A NUMERICAL IMPLEMENTATION FOR THE HIGH-ORDER 2D VIRTUAL ELEMENT METHOD IN MATLAB

CÉSAR HERRERA [*], RICARDO CORRALES-BARQUERO [†], JORGE ARROYO-ESQUIVEL [‡], AND JUAN G. CALVO [§]

Abstract. We present a numerical implementation for the Virtual Element Method that incorporates high order spaces. We include all the required computations in order to assemble the stiffness and mass matrices, and right hand side. Convergence of method is verified for different polygonal partitions. An specific mesh-free application that allows to approximate harmonic functions is discussed, based on high-order projections. This approach significantly improves running times compared to usual finite or virtual element methods, and can be modified for different virtual spaces and elliptic partial differential equations.

Key words. Virtual element methods, Polygonal mesh, Matlab implementation

AMS subject classifications. 65-04, 65N30, 65D15.

1. Introduction. In the past few years, there has been an increasing interest on the novel Virtual Element Methods (VEM) introduced by Beirão da Veiga et al. in [5]. General element geometries can be considered and local spaces contain polynomials and virtual functions. Even though in general it is not possible to construct local basis functions, the stiffness matrix can be computed by knowing the degrees of freedom. Higher-order continuity conditions can also be achieved by this method and it can be established for different applications; see, e.g., [11, 6, 3, 2, 4, 9, 18, 23].

In this paper we extend the work presented in [21] where the lowest order VEM is implemented in `MATLAB`. Other libraries as `VEMLab` [19] and `iFEM` [15] include VEM examples, but they are limited to particular elements and low order spaces. In our case, we present an open code written in `MATLAB` for general meshes and high order spaces, in the pursuit of an easy-to-follow didactic implementation. We confirm convergence of the method for different type of meshes and a include experiments with two choices for the monomial basis. We also include an application where the projectors defined in VEM can be used as an alternative to construct accurate solutions to Laplace's equation. The library can be found in [12].

Besides solving elliptic partial differential equations via VEM, virtual spaces have been used for the construction of coarse spaces for preconditioners based on Domain Decomposition Methods in the presence of irregular subdomains; see for example [13]. There are also recent studies on iterative solvers for VEM; see, e.g., [13, 14, 16, 24]. We remark that it is required to construct high-order virtual spaces in order to implement the algorithm studied in [13].

The rest of this paper is organized as follows. In Section 2 we introduce the notation that will be used throughout our manuscript. In Section 3, we present the basic theory of VEM. A `MATLAB` implementation is presented in Section 4, where we present with detail the computation of the stiffness and mass matrices, and the right

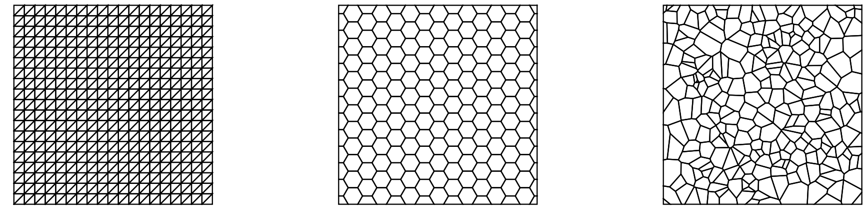
---

[*] Escuela de Matemática, Universidad de Costa Rica, San Jose, Costa Rica. `cesar.herreragarro@ucr.ac.cr`

[†] Escuela de Matemática, Universidad de Costa Rica, San Jose, Costa Rica. `ricardo.corralesbarquero@ucr.ac.cr`

[‡] Department of Mathematics, University of California Davis, California, USA. `jarroyoe@ucdavis.edu`

[§] CIMPA - Escuela de Matemática, Universidad de Costa Rica, San Jose, Costa Rica `juan.calvo@ucr.ac.cr`



(a) Triangular mesh (b) Hexagonal mesh (c) Voronoi mesh

Fig. 2.1: Examples of polygonal partitions $\mathcal{T}_h$ for $\Omega = [0,1]^2$ used in Section 5.

hand side. In Section 5, we present some numerical results that confirm convergence estimates and an application that allows to approximate harmonic functions efficiently. Finally, in Section 6 we include relevant conclusions and final remarks.

2. Notation.

We follow the notation introduced in [7] for the description of the method. Let $\{\mathcal{T}_h\}_h$ be a family of polygonal partitions of a given domain $\Omega \subset \mathbb{R}^2$; see [5, Section 3] for assumptions on these decompositions and Figure 2.1 for three particular examples. Given an element $E \subset \mathbb{R}^2$, denote by $\boldsymbol{b}_E$, h_E, $|E|$ and $\mathcal{S}_E$ the barycenter, diameter, area and set of edges of E, respectively. Let $\mathcal{P}_k(E)$ be the space of polynomials defined in E of degree at most k, and define

$$n_k = \dim \mathcal{P}_k(E) = \frac{(k+1)(k+2)}{2}.$$

For any point $\boldsymbol{x} = (x_1, x_2)^T$ in $\mathbb{R}^2$, we use the standard multi-index notation

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2), \quad \boldsymbol{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} x_2^{\alpha_2}, \quad |\boldsymbol{\alpha}| = \alpha_1 + \alpha_2.$$

Given a multi-index $\boldsymbol{\alpha}$, the scaled monomial $m_{\boldsymbol{\alpha}} : E \to \mathbb{R}$ is defined as

$$m_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \left(\frac{\boldsymbol{x} - \boldsymbol{b}_E}{h_E} \right)^{\boldsymbol{\alpha}}. \tag{2.1}$$

These monomials are ordered via the natural correspondence

$$1 \leftrightarrow (0,0), 2 \leftrightarrow (1,0), 3 \leftrightarrow (0,1), 4 \leftrightarrow (2,0), 5 \leftrightarrow (1,1) \ldots,$$

and we then consider the ordered set

$$\mathcal{M}_k(E) = \{m_1, \ldots, m_{n_k}\} \tag{2.2}$$

of scaled monomials of degree at most k. Finally, for a given vector $\boldsymbol{x} = [x_1, x_2]^T$ we denote by $\boldsymbol{x}^\perp = [x_2, -x_1]$ its clockwise 90° rotation.

3. Virtual Element Methods.

We briefly describe the VEM for nodal elliptic problems in two dimensions; we refer to [5, 7] and references therein for a more detailed exposition. For simplicity, we consider Poisson's equation in a given domain $\Omega \subset \mathbb{R}^2$ with homogeneous Dirichlet boundary conditions on $\partial\Omega$. Different boundary conditions can be incorporated straightforwardly.

Given an integer $k \geq 1$, the local virtual space of degree k is defined for every element $E \in \mathcal{T}_h$ as

$$V_k(E) := \{v \in H^1(E) : v|_{\partial E} \in C^0(\partial E),\ v|_e \in \mathcal{P}_k(e)\,\forall e \in \mathcal{S}_E,\ \Delta v \in \mathcal{P}_{k-2}(E)\}, \tag{3.1}$$

where $\mathcal{P}_{-1}(E) = \{0\}$. Given $v \in V_k(E)$, its local degrees of freedom (dof) can be chosen as:

1. The value of v at each vertex of E.
2. The value of v at the $k-1$ internal Gauss-Lobatto quadrature nodes of every edge of E.
3. The moments of v up to order $k-2$ given by

$$\frac{1}{|E|}\int_E v(\boldsymbol{x})\ m_j(\boldsymbol{x})\ d\boldsymbol{x} \quad (j = 1, 2, \ldots, n_{k-2});$$

see [5, Equation 2.4]. The total number of local dof is then

$$N^E_{\text{dof}} = \dim V_k(E) = kN^E_V + n_{k-2},$$

where N^E_V is the number of vertices of E.

For each element E, its local dof are numbered as follows. First consider its N^E_V vertices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N^E_V}$ (ordered counterclockwise). Second, consider the interior nodes on each edge (starting from $\boldsymbol{x}_1$ in the counterclockwise direction) and denote them by $\boldsymbol{x}_{N^E_V+1}, \ldots, \boldsymbol{x}_{kN^E_V}$; see Figure 3.1a. Finally, consider the moments ordered as in (2.2). We then define the functional

$$\text{dof}^E_i : V_k(E) \to \mathbb{R}$$

where $\text{dof}^E_i(v)$ is the i-th degree of freedom of v. Hence, for $v \in V_k(E)$ it holds that

$$\text{dof}^E_i(v) = \begin{cases} v(\boldsymbol{x}_i) \quad (1 \leq i \leq kN^E_V), \\ \dfrac{1}{|E|}\displaystyle\int_E v(\boldsymbol{x}) m_{i-kN^E_V}(\boldsymbol{x})\ d\boldsymbol{x} \quad (kN^E_V + 1 \leq i \leq kN^E_V + n_{k-2}). \end{cases} \tag{3.2}$$

The canonical local basis is then defined as the set $\{\varphi^E_i\}_{i=1}^{N^E_{\text{dof}}}$ such that $\text{dof}^E_i(\varphi^E_j) = \delta_{ij}$. Define $P_0 : V_k(E) \to \mathcal{P}_0(E)$ by

$$P_0 v = \frac{1}{N^E_V}\sum_{i=1}^{N^E_V} v(\boldsymbol{x}_i), \quad \text{for} \ \ k = 1;$$

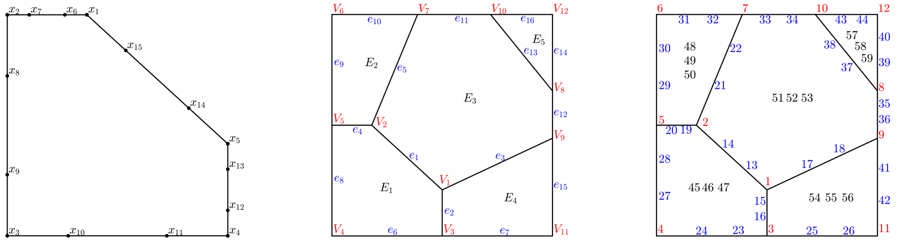
$$P_0 v = \frac{1}{|E|}\int_E v(\boldsymbol{x})\ d\boldsymbol{x}, \quad \text{for} \ \ k \geq 2.$$

A projection operator $\Pi^\nabla_{E,k} : V_k(E) \to \mathcal{P}_k(E)$ is defined as follows. Let $w = \Pi^\nabla_{E,k} v$ be the polynomial of degree at most k such that

$$a^E(w, p) = a^E(v, p) \quad \forall\, p \in \mathcal{P}_k(E) \quad \text{and} \quad P_0 v = P_0 w,$$

where $a^E(u, v) = \displaystyle\int_E \nabla u(\boldsymbol{x}) \cdot \nabla v(\boldsymbol{x})\ d\boldsymbol{x}$. The local bilinear form in the VEM is defined as

$$a^E_h(u, v) = \int_E \nabla\Pi^\nabla_{E,k} u \cdot \nabla\Pi^\nabla_{E,k} v + \sum_{r=1}^{N^E_{\text{dof}}} \text{dof}_r((I - \Pi^\nabla_{E,k})u)\text{dof}_r((I - \Pi^\nabla_{E,k})v);$$



(a) (b) (c)

Fig. 3.1: (a) An element with 5 vertices. The local order of dof on ∂E is shown for $k = 3$. There are $n_1 = 3$ additional internal dof related to the monomials m_1, m_2, m_3. (b) A mesh with $N = 59$ dof: $N^V = 12$ vertices (V_i), $N^e = 16$ edges (e_i), and $N^E = 5$ elements (E_i). (c) Global order of dof, corresponding to vertices (red), edges (blue) and moments (black).

see [7, Section 3.3] for further details. Therefore, the entries corresponding to the local stiffness matrix are given by

$$(A_h^E)_{i,j} = a_h^E(\varphi_j, \varphi_i). \tag{3.3}$$

Globally, the virtual element space $V_h \subset H_0^1(\Omega)$ is defined as

$$V_h = \{v \in H_0^1(\Omega) : v|_E \in V_k(E) \ \text{ for all } E\}.$$

The total number of global dof is given by

$$N = \dim(V_h) = N^V + N^e(k-1) + N^E n_{k-2},$$

where N^V, N^e and N^E represent the total number of vertices, edges and elements in the triangulation, respectively. The global dof are ordered in a natural way. First, the nodal values on the vertices; second, the nodal values on the internal points on each edge (ordered from the node with smaller global index to the remaining node); third, the internal moments for each element; see Figures 3.1b and 3.1c.

The global bilinear form is then obtained by assembling the local bilinear forms:

$$a_h(u, v) = \sum_E a_h^E(u, v).$$

For the local right hand side, different approaches have been proposed; see, e.g., [7, Section 6] and references therein. In our implementation we consider the $L^2(E)-$projection operator $\Pi_k^0 : V_k(E) \to \mathcal{P}_k(E)$. Then, the local contribution for the load term is defined as

$$(\boldsymbol{b}_E^h)_i = \int_E f \ \Pi_k^0 \varphi_i \quad (i = 1, \ldots, N_{\text{dof}}^E). \tag{3.4}$$

After assembling the local stiffness matrices and right hand sides, we solve the global linear system of equations for which standard optimal estimates can be obtained; see e.g., [1, 5, 8].

Remark 3.1. We could replace $V_k(E)$ by the space $W_k(E)$, where the condition $\Delta v \in \mathcal{P}_{k-2}(E)$ given in (3.1) is replaced by the conditions

$$\Delta v \in \mathcal{P}_k(E) \text{ and } \int_E v m_{\boldsymbol{\alpha}} = \int_E \Pi^{\nabla}_{E,k} v \, m_{\boldsymbol{\alpha}} \text{ for } |\boldsymbol{\alpha}| \in \{k-1, k\};$$

see [7, Section 5]. From a practical point of view, degrees of freedom are the same for both spaces, but using the space W_k allows us to compute the L^2-projector and the mass matrix; see [7, Remarks 5.1, 5.2].

4. Implementation.

In this section we describe a numerical implementation in `MATLAB` [22] for the Virtual Element Method with local spaces of degree $k \geq 2$. The case $k = 1$ was implemented in [21] and we omit further details for such case.

4.1. Mesh.

Consider a polygonal mesh stored as a `MATLAB` structure with the following fields:

- `mesh.verts`: a $N^V \times 2$ matrix with the (x_1, x_2) coordinates of all the N^V vertices of the mesh. The point `mesh.verts(j,:)` corresponds to the $j-$th global node of the mesh.
- `mesh.elems`: a $N^E \times 1$ cell with the connectivity of the elements. Each vector `mesh.elems{j}` has $N_V^{E_j}$ indices, corresponding to the $N_V^{E_j}$ vertices of the j-th polygon E ordered counterclockwise.
- `mesh.bndry`: a column vector with the indices of the vertices on $\partial\Omega$.

The edge connectivity is stored in the $N^e \times 2$ matrix `mesh.edges`. The j-th global edge connects the vertex `mesh.edges(j,1)` to the vertex `mesh.edges(j,2)`. This array is created by computing all the edges of the triangulation and then eliminating repeated entries. For each element E, we create the field `mesh.dof` with the mapping that assigns the global dof to each local dof. We store the global indices of all the dof related to nodes on $\partial\Omega$ in the vector `mesh.bdDOF`; see Listing 1.

```
function mesh = meshSetup(mesh,k)
nNodes = size(mesh.verts,1);            % global number of nodes
nVerts = cellfun(@numel,mesh.elems);    % vertices per element
% create array with all edges [e1 e2] (smaller index first)
e1 = cell2mat(mesh.elems);
e2 = cell2mat(cellfun(@(y)y([2:end,1]),mesh.elems,'UniformOutput',0));
[edges,index] = sort([e1 e2],2);
signEdges = index(:,2)-index(:,1);      % reversed edges
[mesh.edges, ¬, indexEdges] = unique(edges,'rows');
numEdges = size(mesh.edges,1);          % global number of edges
% find edges on boundary
uniqueEdges = hist(indexEdges,1:size(mesh.edges,1))';
edgesBd = uniqueEdges==1;
edgesBd = nNodes+(find(edgesBd)-1)*(k-1)+repmat(1:k-1,sum(edgesBd),1);
mesh.bdDOF = sort([mesh.bndry; edgesBd(:)]);
% create array with global dof for each element
index = [0 cumsum(nVerts)'];
for j = 1:size(mesh.elems,1)
    range = index(j)+1:index(j+1);
    increment = repmat(1:k-1,nVerts(j),1);
    % reverse order on edges that changed orientation
    sign  = signEdges(range);
    increment(sign<0,:) = fliplr(increment(sign<0,:));
    % find global dof for local vertices, edges and moments
    dofVert=mesh.elems{j};
    dofEdgs=(nNodes+(indexEdges(range)-1)*(k-1)+increment)';
    dofElem=nNodes+numEdges*(k-1)+(j-1)*k*(k-1)/2+(1:k*(k-1)/2)';
    mesh.dof{j} = [dofVert; dofEdgs(:); dofElem];
end
end
```

Listing 1: Mesh setup: edge connectivity, boundary dof and local to global mapping

As an example, Figure 3.1b corresponds to the array

$$\texttt{mesh.edges} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 5 & 6 & 7 & 8 & 8 & 8 & 9 & 10 \\ 2 & 3 & 9 & 5 & 7 & 4 & 11 & 5 & 6 & 7 & 10 & 9 & 10 & 12 & 11 & 12 \end{bmatrix}^T.$$

The first element satisfies $\texttt{mesh.elems}\{1\} = [2, 5, 4, 3, 1]$. Thus,

$$\texttt{mesh.dof}\{1\} = [2, 5, 4, 3, 1, 19, 20, 28, 27, 24, 23, 16, 15, 13, 14, 45, 46, 47].$$

It also holds that `mesh.bdDOF = [3:12,23:36,39:44]`^{T}; see Figure 3.1c.

4.2. Preliminaries. In this section we describe five auxiliary routines. First, for Gauss-Lobatto integration, the nodes and weights for the interval $[0, 1]$ are obtained as described in [17]; see Listing 2.

```
function quadrature = quadInfoGL(k)
% GL quadrature rule in [0,1]
j = 1:k-2;
v = sqrt(j.*(j+2)./((2*j+1).*(2*j+3)));
A = diag(v,-1)+diag(v,1);
[V,D] = eig(A);
x   = diag(D);                          % quad nodes in [-1,1]
w   = V(1,:).^2*4/3;                    % quad weights
w1  = 1-sum(w./(1-x'))/2;               % weight initial node
w2  = 1-sum(w./(1+x'))/2;               % weight final node
quadrature.nodes   = ([-1; x; 1]+1)/2;  % translate to [0,1]
quadrature.weights = [w1, w./((1-x').*(1+x')), w2]'/2;
end
```

Listing 2: Nodes and weights for Gauss-Lobatto quadrature

We then create an array with the exponents of the scaled monomials up to degree k ordered as in (2.2); see Listing 3.

```
function polys = createPolynomials(k)
polys = zeros((k+1)*(k+2)/2,2);
pointer = 0;
for j = 0:k
    polys(pointer+1:pointer+j+1,:) = [j:-1:0; 0:j]';
    pointer = pointer + j + 1;
end
```

Listing 3: List of exponents for the scaled monomials as given in (2.2)

We also need to compute the area A_E and barycenter $\boldsymbol{b}_E$ for each element E. If

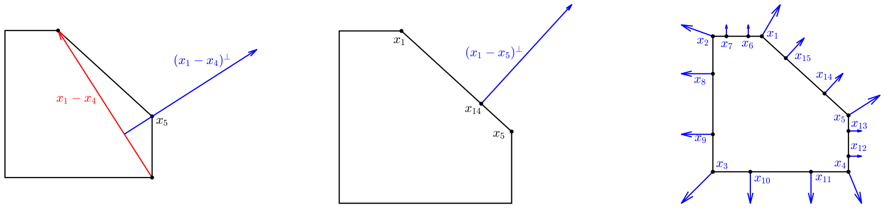


Fig. 4.1: (left) For a vertex $\boldsymbol{x}_j$ of E, we compute $(\boldsymbol{x}_{j+1}-\boldsymbol{x}_{j-1})^\perp$ (center) For the interior nodes of an edge with endpoints $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}$, we compute $(\boldsymbol{x}_{i+1}-\boldsymbol{x}_i)^\perp$ (right) Required vectors for each nodal dof on ∂E; see Section 4.3.

E has vertices (x_i, y_i) $(i = 1, \ldots, N_E^V)$, it is well-known that

$$A_E = \frac{1}{2}\sum_{i=1}^{N_E^V}(x_i y_{i+1} - x_{i+1} y_i),$$

$$\boldsymbol{b}_E = \frac{1}{6A}\left(\sum_{i=1}^{N_E^V}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \sum_{i=1}^{N_E^V}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\right),$$

see Listing 4.

```
function [diameter,area,centroid] = geomElement(verts)
% compute diameter (maximum distance between two vertices)
allPairs  = combnk(1:size(verts,1),2);
diffVerts = verts(allPairs(:,1),:)-verts(allPairs(:,2),:);
diameter  = sqrt(max(sum(diffVerts.*diffVerts,2)));
% formulas for area and centroid
x  = verts(:,1); y  = verts(:,2);
x1 = x([2:end,1]);  y1 = y([2:end,1]);  % cyclic ordering
area_sum = x.*y1 - x1.*y;
area     = sum(area_sum)/2;
centroid = sum([(x+x1), (y+y1)].*area_sum)/(6*area);
end
```

Listing 4: Diameter, area and centroid for a polygon with a given set of vertices

In order to evaluate efficiently the monomials on the boundary nodes of each element, we create a matrix `nodesBd` with the coordinates of the GL nodes on ∂E as in Figure 3.1a. We also require to compute normal vectors for each edge and their averages (one per dof on ∂E); see Figure 4.1 and Section 4.3 for further details. The code is shown in Listing 5.

```
function [nodesBd,vectorBd] = bndryNodesVectors(verts,quadNode)
k = numel(quadNode)-1;   % degree
nVerts = size(verts,1);  % number of vertices
% dof on boundary (vertices and GL nodes)
initPts = repelem(verts,k-1,1);
direc   = repelem(diff([verts; verts(1,:)]),numel(quadNode)-2,1);
nodesBd = [verts; initPts+repmat(quadNode(2:end-1),nVerts,1).*direc];
% required vectors on boundary
```

```
tangV  = diff([verts; verts(1,:)]);      % tangential vectors
normal = [tangV(:,2), -tangV(:,1)];      % normal vectors
avNorm = normal+normal([end,1:end-1],:);% average normal
% vectors: average on vertices, normal on edges
vectorBd = [avNorm;reshape(reshape(repmat(normal,1,k-1)',1,2*(k-1)*...
    nVerts),2,[])'];
end
```

Listing 5: Coordinates of nodes related to dof on ∂E and vectors shown in Figure 4.1

We then precompute all the integrals of the form

$$I(\alpha_1,\alpha_2) = \int_E \left(\frac{x_1 - b_1^E}{h_E}\right)^{\alpha_1} \left(\frac{x_2 - b_2^E}{h_E}\right)^{\alpha_2} \, dx_2 \, dx_1, \tag{4.1}$$

that can be evaluated by using the divergence theorem:

$$\begin{aligned} I(\alpha_1,\alpha_2) &= \frac{h_E}{1+\alpha_1}\int_E \operatorname{div}\left[\left(\frac{x_1-b_1}{h_E}\right)^{1+\alpha_1}\left(\frac{x_2-b_2}{h_E}\right)^{\alpha_2}, 0\right]^T \\ &= \frac{h_E}{1+\alpha_1}\sum_{e\in\partial E}\int_e \left(\frac{x_1-b_1}{h_E}\right)^{1+\alpha_1}\left(\frac{x_2-b_2}{h_E}\right)^{\alpha_2} n_1, \end{aligned}$$

where $\boldsymbol{n} = [n_1, n_2]^T$ is the outward normal of the edge $e \in \partial E$. Each integral over an edge e includes a polynomial that can be integrated exactly by using the Gauss-Lobato quadrature if its degree is at most $2k-1$, as we require in the construction of the matrix D. In Listing 6 we compute all the integrals of the form (4.1) with $0 \le \alpha_1 + \alpha_2 \le 2k$ because the case $|\alpha| = 2k$ is required to compute the right hand side. The value $I(\alpha_1,\alpha_2)$ is stored in the entry (α_1+1, α_2+1) of the matrix `intMon`.

```
function intMon = precomputeIntegrals(verts,centroid,h,area,k)
x  = verts(:,1); x(end+1) = x(1);
y  = verts(:,2); y(end+1) = y(1);
xb = centroid(1);   yb = centroid(2);
quadrature   = quadInfoGL(k+1);         % GL for degree 2k
polys        = createPolynomials(2*k); % monomials of deg 2k
intMon = nan(2*k+1,2*k+1);       % preallocate matrix
intMon(1,1) = area;
for index_monomial = 2:(2*k+1)*(k+1) % integral for monomials
   int = 0;
   poly = polys(index_monomial,:);
   for i = 1:length(x)-1
    fun = @(t) (((x(i).*(1-t)+x(i+1).*t-xb)/h).^(poly(1)+1))...
       .*(((y(i).*(1-t)+y(i+1).*t-yb)/h).^(poly(2))).*(y(i+1)-y(i));
    int = int + sum(fun(quadrature.nodes).*quadrature.weights);
   end
   intMon(poly(1)+1,poly(2)+1) = h/(poly(1)+1)*int;
end
end
```

Listing 6: Computing integrals of the form (4.1) up to degree $2k$

4.3. Local stiffness matrix. We then compute the local stiffness matrix (3.3) by performing the following steps [5, Section 3.5]:

Step 1 Define constants and allocate structures for storage; see Listing 7. The stiffness and mass matrices are built via `A = sparse(ii,jj,aa)` and `M = sparse(ii,jj,mm)`, respectively.

```
% preliminaries
mesh   = meshSetup(mesh,k);
quad   = quadInfoGL(k);
polys = createPolynomials(k);
% constants and arrays for data
nk2       = (k-1)*k/2;
nPolys   = (k+1)*(k+2)/2;
ndofEl   = k*cellfun(@numel,mesh.elems)+nk2; % dof per element
sizeMat = sum(ndofEl.^2); sizeRHS = sum(ndofEl);
aa   = nan(sizeMat,1); ii   = nan(sizeMat,1);
mm   = nan(sizeMat,1); jj   = nan(sizeMat,1);
rhs = nan(sizeRHS,1); irhs = nan(sizeRHS,1);
```

Listing 7: Preliminaries and arrays to store matrix entries

Step 2 For each element, we compute its vertices, diameter, area, centroid, coordinates of dof on boundary, vectors shown in Figure 4.1, integrals given by (4.1), $(\boldsymbol{x} - \boldsymbol{b}_E)/h_E$ and N^E_{dof}; see Listing 8.

```
    verts   = mesh.verts(mesh.elems{E},:);
    nVerts = size(verts,1);
    [h,area,centroid] = geomElement(verts);
    [nodesBd,vectorBd]= bndryNodesVectors(verts,quad.nodes);
    intMon = precomputeIntegrals(verts,centroid,h,area,k);
    base      = (nodesBd-centroid)/h;
    n_dof     = k*nVerts + nk2;
```

Listing 8: Initialize variables for each element

Step 3 Compute the matrix

$$D = \begin{bmatrix} \text{dof}_1(m_1) & \text{dof}_1(m_2) & \cdots & \text{dof}_1(m_{n_k}) \\ \text{dof}_2(m_1) & \text{dof}_2(m_2) & \cdots & \text{dof}_2(m_{n_k}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{dof}_{N^{\text{dof}}}(m_1) & \text{dof}_{N^{\text{dof}}}(m_2) & \cdots & \text{dof}_{N^{\text{dof}}}(m_{n_k}) \end{bmatrix}.$$

From (3.2) we can write explicitly

$$D_{ij} = \begin{cases} m_j(\boldsymbol{x}_i), & i \in \{1, \ldots, N^V_E k\}, \\ \dfrac{1}{|E|} \displaystyle\int_E m_{i - N^V_E k}(\boldsymbol{x})\ m_j(\boldsymbol{x})\ d\boldsymbol{x}, & i \in \{N^V_E k + 1, \ldots, N^E_{\text{dof}}\}. \end{cases} \tag{4.2}$$

The code for computing D is shown in Listing 9.

```
    D = ones(n_dof, nPolys);   % first column is one
    for poly_id = 2:nPolys     % nodal dof
        D(1:k*nVerts,poly_id) = prod(base.^polys(poly_id,:),2);
    end
    for dof_id = k*nVerts+1:n_dof % moments
        poly = polys(dof_id-k*nVerts,:)+polys;
        D(dof_id,:) = intMon(sub2ind(size(intMon), poly(:,1)+1, poly...
             (:,2)+1))/area;
    end
```

Listing 9: Computing D as given in (4.2)

Step 4 Compute the matrix

$$B = \begin{bmatrix} P_0\varphi_1 & \cdots & P_0\varphi_{N^{\text{dof}}} \\ a^E(m_2, \varphi_1) & \cdots & a^E(m_2, \varphi_{N^{\text{dof}}}) \\ \vdots & \ddots & \vdots \\ a^E(m_{n_k}, \varphi_1) & \cdots & a^E(m_{n_k}, \varphi_{N^{\text{dof}}}) \end{bmatrix}.$$

Since $P_0\varphi_j$ gives the mean value of φ_j on E, all the entries in the first row are zero except for the moment of $\varphi_{N_E^V k+1}$; this is,

$$B_{1,j} = \begin{cases} 1, & j = N_E^V k + 1, \\ 0, & j \neq N_E^V k + 1. \end{cases}$$

For $i = 2, \ldots, n_k$, we need to compute

$$B_{ij} = \int_E \nabla m_i \ \varphi_j = -\int_E \Delta m_i \ \varphi_j + \int_{\partial E} \frac{\partial m_i}{\partial n} \ \varphi_j. \tag{4.3}$$

We then have two possibilities:

1. For $j = 1, \ldots, N_E^V k$, all the internal moments are zero and (4.3) simplifies to

   $$B_{ij} = \int_{\partial E} \nabla m_i \cdot \boldsymbol{n} \ \varphi_j \quad (j = 1, \ldots, N_E^V k),$$

   which can be evaluated exactly by the Gauss-Lobatto quadrature:

   (a) If φ_j corresponds to a vertex $\boldsymbol{x}_j$ of E, we then obtain that

   $$B_{ij} = \omega \nabla m_i(\boldsymbol{x}_j) \cdot (\boldsymbol{x}_{j+1} - \boldsymbol{x}_{j-1})^{\perp}.$$

   (b) If φ_j corresponds to a vertex in the interior of an edge $e_k \in \partial E$ with endpoints V_k, V_{k+1}, then

   $$B_{ij} = \omega \nabla m_i(\boldsymbol{x}_j) \cdot (V_{k+1} - V_k)^{\perp}.$$

   where ω is the Gauss-Lobatto weight associated to node $\boldsymbol{x}_j$.
2. For $j = N_E^V k + 1, \ldots, N^{\text{dof}}$, the integral on ∂B vanishes. Writing $m_i(\boldsymbol{x}) = ((\boldsymbol{x} - \boldsymbol{b}_E)/h_E)^{\boldsymbol{\alpha}}$, we conclude that

   $$B_{ij} = -\int_E \Delta m_i \ \varphi_j = -\frac{\alpha_1(\alpha_1 - 1)}{h_E^2} \int_E \left(\frac{\boldsymbol{x} - \boldsymbol{b}_E}{h_E}\right)^{[\alpha_1 - 2, \alpha_2]} \varphi_j \\ - \frac{\alpha_2(\alpha_2 - 1)}{h_E^2} \int_E \left(\frac{\boldsymbol{x} - \boldsymbol{b}_E}{h_E}\right)^{[\alpha_1, \alpha_2 - 2]} \varphi_j$$

   Note that the first term is equal to zero, unless $m_{j - N_E^V k}$ has exponent $[\alpha_1 - 2, \alpha_2]$; similarly the second integral vanishes unless $m_{j - N_E^V k}$ has exponent $[\alpha_1, \alpha_2 - 2]$. Therefore,

   $$B_{ij} = \begin{cases} -\dfrac{\alpha_1(\alpha_1 - 1)|E|}{h_E^2}, & \text{if } m_{j - N_E^V k}(\boldsymbol{x}) = \left(\dfrac{\boldsymbol{x} - \boldsymbol{b}_E}{h_E}\right)^{[\alpha_1 - 2, \alpha_2]} \\ -\dfrac{\alpha_1(\alpha_1 - 1)|E|}{h_E^2}, & \text{if } m_{j - N_E^V k}(\boldsymbol{x}) = \left(\dfrac{\boldsymbol{x} - \boldsymbol{b}_E}{h_E}\right)^{[\alpha_1, \alpha_2 - 2]} \\ 0, & \text{otherwise.} \end{cases}$$

The code for computing B is shown in Listing 10.

```
B = zeros(nPolys, n_dof); B(1,k*nVerts+1) = 1;
for poly_id = 2:nPolys % non-constant polynomials
    poly_degree = polys(poly_id,:);
    g1 = prod(base.^[max(poly_degree(1)-1,0) poly_degree(2)],2);
    g2 = prod(base.^[poly_degree(1) max(poly_degree(2)-1,0)],2);
    gradM_a = poly_degree.*[g1 g2]/h;
    B(poly_id, 1:k*nVerts) = sum(gradM_a.*vectorBd,2).*[repmat(...
        quad.weights(1),nVerts,1); repmat(quad.weights(2:end-1),...
        nVerts,1)];
end
for dof_id = k*nVerts+1:n_dof
    varphi_j = polys(dof_id-k*nVerts,:);
    poly_degree = polys(2:end,:);
    B(2:end,dof_id) = - area/h^2 * ...
        (poly_degree(:,1).*(poly_degree(:,1)-1).*prod(poly_degree...
            -[2,0] == varphi_j,2)+poly_degree(:,2).*(poly_degree...
            (:,2)-1).*prod(poly_degree-[0,2] == varphi_j,2));
end
```

Listing 10: Computing B

Step 5 Compute $G = BD$.
Step 6 Compute the matrices corresponding to the projection operators:

$$\Pi_*^\nabla = G^{-1}B, \quad \Pi^\nabla = D\Pi_*^D.$$

Step 5 Compute the local stiffness matrix:

$$A_E^h = (\Pi_*^\nabla)^T \widetilde{G}(\Pi_*^\nabla) + (I - \Pi^\nabla)^T(I - \Pi^\nabla),$$

where $\widetilde{G}$ is the matrix obtained from G by replacing the first row by zeros.

The last three steps are straightforward to compute; see Listing 11.

```
G = B*D;
PiStar = G\B;
G(1, :) = 0;
I = eye(n_dof);
A = PiStar'*G*PiStar + sigma*(I-D*PiStar)'*(I-D*PiStar);
```

Listing 11: Computing A_E^h

4.4. Mass matrix and right hand side. For each basis function, we compute its L^2-projection onto the space of polynomials of degree at most k. First, we compute the matrix $H \in \mathbb{R}^{n_k \times n_k}$ with entries

$$H_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \int_E m_{\boldsymbol{\alpha}} m_{\boldsymbol{\beta}}.$$

We remark that these entries were already computed in Listing 6. Second, we compute the matrix $C \in \mathbb{R}^{n_k \times N_{\text{dof}}^E}$ given by

$$C_{\boldsymbol{\alpha},i} = \begin{cases} \int_E m_{\boldsymbol{\alpha}} \varphi_i & \text{if} \quad 1 \le \alpha \le n_{k-2}, \\ \int_E m_{\boldsymbol{\alpha}} \Pi^\nabla \varphi_i & \text{if} \quad n_{k-2}+1 \le \alpha \le n_k. \end{cases}$$

It holds that $C_{\alpha,i} = (HG^{-1}B)_{\alpha,i}$ for $n_{k-2}+1 \leq \alpha \leq n_k$; see [5, Remark 5.4]. The L^2-projector acting on the monomial basis and in the virtual element local basis are given by $\Pi^0_{*,k} = H^{-1}C$ and $\Pi^0_k = DH^{-1}C$, respectively.

After straighforward computations, the local right hand side (3.4) is given by

$$b_E = (\Pi^0_{*,k})^T \left[\int_E f m_1, \int_E f m_2, \ldots, \int_E f m_{n_k} \right]^T .$$

For the sake of simplicity, each integral $\int_E f(\boldsymbol{x}) m_i(\boldsymbol{x})\, d\boldsymbol{x}$ is computed by triangulating E and applying a change of basis to a reference triangle. We have tested different approaches in order to improve accuracy with similar results. Finally, the local mass matrix with entries

$$(M^h_E)_{ij} = \int_E \varphi_i \varphi_j$$

can be computed via the formula

$$M^h_E = C^T H^{-1} C + |E|(I - \Pi^0_k)^T (I - \Pi^0_k);$$

see [7, Section 6.2] and Listing 12.

```
H = zeros(nPolys,nPolys);             % matrix H
for i = 1:nPolys
    poly  = polys(i,:)+polys;
    H(i,:)= intMon(sub2ind(size(intMon),poly(:,1)+1,poly(:,2)+1));
end
C = zeros(nPolys,n_dof);              % matrix C
C(1:nk2,k*nVerts+(1:nk2)) = area*eye(nk2);
Ctem = H*PiStar;
C(nk2+1:nPolys,:) = Ctem(nk2+1:nPolys,:);
PiStar0 = H\C;                        % L2 projections
Pi0 = D*PiStar0;
M = C'*(H\C)+area*(I-Pi0)'*(I-Pi0);   % mass matrix
int_fm = zeros(nPolys,1);             % compute (f,Pi*m_alpha)
tri = delaunay(verts(:,1),verts(:,2));
for j = 1:nPolys
    fun = @(x,y) (((x-centroid(1))./h).^polys(j,1)).*(((y-centroid...
        (2))./h).^polys(j,2)).*f(x,y);
    int_fm(j) = integrate(fun,tri,verts);
end
rhsLoc = PiStar0'*int_fm;
```

Listing 12: Computing the local right hand side and mass matrix

4.5. Assembling the global stiffness matrix.

For each element, the mass matrix, stiffness matrix and right hand side are stored in the vectors `mm`, `aa` and `rhs`, respectively, with the local to global dof mapping stored in the vectors `ii`, `jj` and `irhs`; see Listing 13.

```
[dofi,dofj] = meshgrid(mesh.dof{E},mesh.dof{E});
mm(ptMat+1:ptMat+n_dof^2) = M(:);
aa(ptMat+1:ptMat+n_dof^2) = A(:);
rhs(ptRHS+1:ptRHS+n_dof)  = rhsLoc;
ii(ptMat+1:ptMat+n_dof^2) = dofi(:);
```

```
    jj(ptMat+1:ptMat+n_dof^2) = dofj(:);
    irhs(ptRHS+1:ptRHS+n_dof) = mesh.dof{E};
    ptRHS = ptRHS + n_dof;
    ptMat = ptMat + n_dof^2;
```

Listing 13: Storing local information for each element

Finally, the global assembly is performed by using the `sparse` and `accumarray` functions. Dirichlet boundary conditions are explicitly imposed in the system.

5. Numerical results. In this section we present different numerical results in order to validate the implementation when solving Poisson's equation with homogeneous Dirichlet boundary conditions.

5.1. Convergence. We first confirm convergence rates as a function of h and k. We consider triangular, hexagonal and Voronoi partitions for $\Omega = [0,1]^2$; see Figures 2.1a, 2.1b and 2.1c for an example of each mesh, respectively. Voronoi elements are obtained with the script `voronoiLimit`; see [20]. We report the diameter h, total number of vertices N^V, edges N^e and elements N^E for hexagonal and Voronoi meshes in Table 5.1.

Table 5.1: Diameter h, number of vertices N_v, edges N_e and elements N_E for three different hexagonal and Voronoi partitions considered in Section 5.

	Hexagons				Voronoi			
i	h_i	N_v	N_e	N_E	h_i	N_v	N_e	N_E
1	$3{,}85 \cdot 10^{-1}$	38	55	18	$3{,}44 \cdot 10^{-1}$	138	205	68
2	$1{,}92 \cdot 10^{-1}$	106	157	52	$1{,}77 \cdot 10^{-1}$	522	781	260
3	$9{,}62 \cdot 10^{-2}$	378	565	188	$8{,}94 \cdot 10^{-2}$	2058	3085	1028

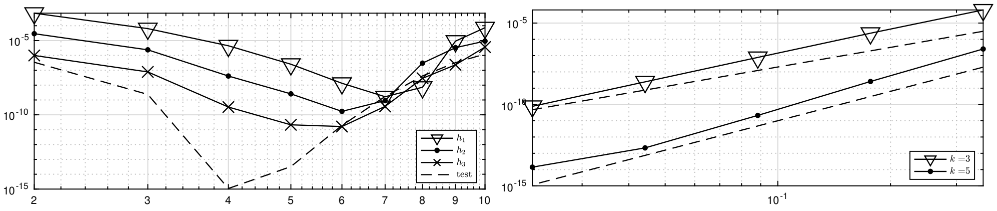
For simplicity we use direct solvers to obtain the numerical solution u_h and we estimate the error $\|u-u_h\|^2_{L^2(\Omega)}$ by $\sum\limits_{E} \|u-\Pi^{\nabla}_{E,k}u_h\|^2_{L^2(E)}$. We choose the load term such that the exact solution is $u(x,y) = \sin(\pi x)\sin(\pi y)$. We remark that in this case it holds that

$$\|u-u_h\|_{L^2(\Omega)} \leq C\frac{h^{k+1}}{k^{r+1}}\|u\|_{H^{r+1}(\Omega)};$$

see [8, Remark 4.4].

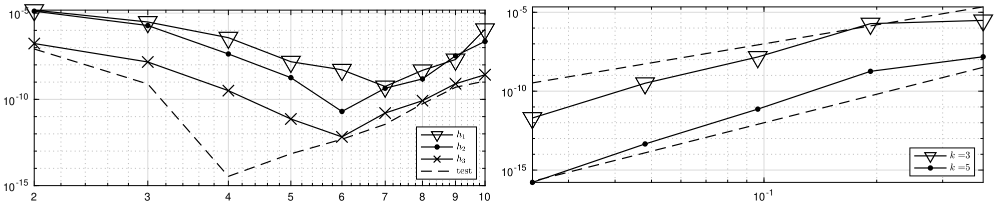
Convergence as a function of k is shown in Figures 5.1a, 5.1c, 5.1e for different partitions and values of h (as given in Table 5.1). Following [8], we include a test (dashed lines) where the exact solution is $u(x,y) = (x-x^2)(y-y^2)$. In this case, the numerical method should recover the exact solution for $k \geq 4$. Therefore, the error is only due to the algebraic solver since it should be zero in exact arithmetic.

5.2. Approximating harmonic functions. As an application, we approximate harmonic functions u with prescribed boundary data g on a given polygonal domain Ω with a different approach as standard FEM or VEM. A similar idea was considered in [13] when constructing a two-level overlapping Schwarz preconditioner; it is shown that coarse spaces of degree $k = 2$ or $k = 3$ are enough to obtain scalable preconditioners.



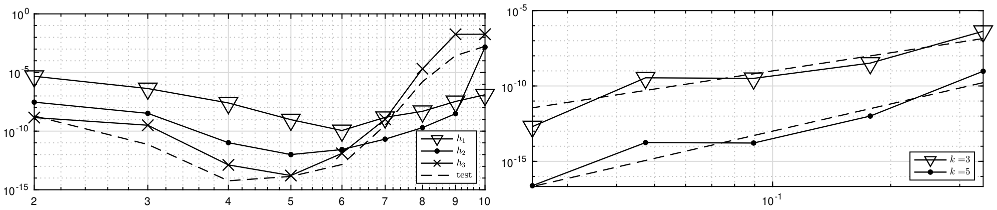
(a) Triangular elements.

(b) Triangular elements.



(c) Hexagonal elements.

(d) Hexagonal elements.



(e) Voronoi elements.

(f) Voronoi elements.

Fig. 5.1: L^2-error for Poisson problem with exact solution $u(x, y) = \sin(\pi x)\sin(\pi y)$ (left) as a function of k and (right) as a function of h (dashed lines correspond to a slope of $k + 1$).

Given $k \geq 1$, we want to approximate u by $u_h \in V_k(\Omega)$. It is clear that the nodal degrees of freedom are known from the boundary data and it is required to compute the internal moments of u_h; we denote the vectors with nodal dof and moments as $\boldsymbol{d}_{\texttt{nod}}$ and $\boldsymbol{d}_{\texttt{mom}}$, respectively. Since u minimizes $\int_\Omega |\nabla u|^2$, we seek $\boldsymbol{d}_{\texttt{mom}} \in \mathbb{R}^{n_{k-2}}$ such that $\int_\Omega |\nabla \Pi^\nabla_{\Omega,k} u_h|^2$ attains its minimum. We remark that this minimum is found by solving just a $n_{k-2} = k(k-1)/2$ linear system.

Explicitly, consider the matrix Q with entries $Q_{ij} = \int_\Omega \nabla m_i \cdot \nabla m_j$ for $i, j \in \{1, 2, \ldots, n_{k-2}\}$ that can be straightforwardly computed from Listing 6. Denote by $\widehat{Q}$ the matrix obtained from Q by removing its first row and first column (that corresponds to the constant monomial). Let $\widehat{P}$ be the matrix obtained from Π^∇_* by removing its first row and let $M = \widehat{P}^T \widehat{Q} \widehat{P}$. If we write

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

where $M_{22} \in \mathbb{R}^{n_{k-2} \times n_{k-2}}$, the minimum is then attained for

$$\boldsymbol{d}_{\texttt{mom}} = -M_{22}^{-1} M_{21} \boldsymbol{d}_{\texttt{nod}}.$$

Therefore, the coefficients of u_h in the monomial basis are given by

$$\Pi_*^\nabla[\boldsymbol{d}_{\mathtt{nod}}\ \boldsymbol{d}_{\mathtt{mom}}]^T.$$

We consider four harmonic functions: $u_1(\boldsymbol{x}) = \sin(2x_2)(e^{2x_1} - e^{-2x_1}), u_2(\boldsymbol{x}) = x_1^6 - 15x_1^4x_2^2 + 15x_1^2x_2^4 - x_2^6$, $u_3(r,\theta) = r^{2/3}\sin(2\theta/3)$ and $u_4(r,\theta) = r^{5/2}\sin(5\theta/2)$. For u_1, u_2 we take $\Omega = (0,1)^2$ and for u_3, u_4 we take $\Omega = (-1,1)^2 \setminus ([0,1]\times[-1,0])$. For simplicity, we compute the pointwise error at the point $\boldsymbol{x} = (1/5, 2/5)$. Numerical results are shown in Figure 5.2, where we confirm convergence as k increases, depending on the regularity of u. These figures were obtained in less than a tenth of a second. For u_1 we have exponential convergence and for u_3, u_4 convergence of order $1/k$ and $1/k^2$, respectively; see [8] for theoretical bounds. Since $u_2 \in V_6(\Omega)$, for $k = 6$ we obtain an accuracy close to machine precision. We then conclude that it is possible to obtain accurate approximations for the solution of Laplace equation. Similar approaches can be defined for different partial differential equations by considering appropriate virtual spaces and its variational form.

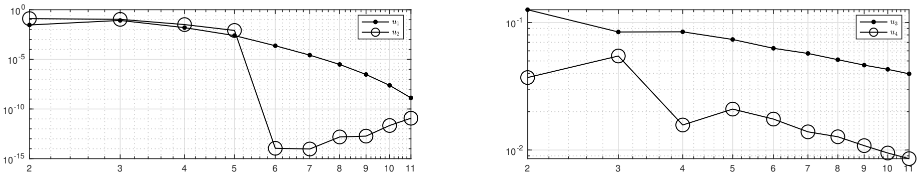
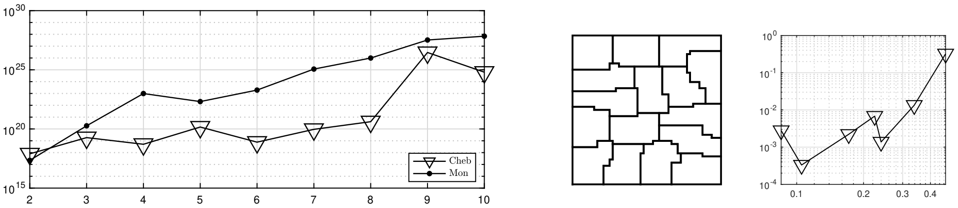


Fig. 5.2: $|u(\boldsymbol{x}) - u_h(\boldsymbol{x})|$ for a particular point $\boldsymbol{x} \in \Omega$ for different harmonic functions.

5.3. Chebyshev basis.

In this section, we show the effect of replacing the monomial basis given in (2.1) by a basis of Chebyshev polynomials. Similar efforts have been considered in [8, 10], where a scaled, Legendre-type and orthogonal basis are considered. We present in Figure 5.3a the effect on the condition number of the linear system. Even though there is a slightly improvement, error estimates are similar for small values of k, which is the same effect observed in [8].



(a)

(b)

Fig. 5.3: (left) Condition number for an hexagonal mesh with standard and Chebyshev monomials. (right) A METIS mesh and the L^2 error as a function of h for such partitions.

6. Final remarks. We have presented an implementation for higher order VEM in the pursuit of a didactic implementation. An open code can be found in [12] with the experiments shown in this paper. We remark that high-order virtual spaces have been used as components for domain decomposition preconditioners [13] and as an alternative to approximate numerical solutions for differential equations as shown in Section 5.2. There is an increasing number of studies on preconditioning linear systems that arise from VEM. Particularly, for METIS-type meshes such as the mesh shown in Figure 5.3b we cannot confirm convergence of the method, since this family of partitions has edges that are not comparable to the diameter of the elements.

Acknowledgments. The authors gratefully acknowledges the institutional support for the project B8290 subscribed to the Vice-Rectory for Research, University of Costa Rica.

Conflict of interest. The authors declare that they have no conflict of interest.

REFERENCES

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

[2] S. N. Alvarez, V. Bokil, V. Gyrya, and G. Manzini, *The virtual element method for resistive magnetohydrodynamics*, Comput. Methods Appl. Mech. Engrg., 381 (2021), p. 113815.

[3] P. F. Antonietti, L. Beirão da Veiga, D. Mora, and M. Verani, *A stream virtual element formulation of the Stokes problem on polygonal meshes*, SIAM J. Numer. Anal., 52 (2014), pp. 386–404.

[4] E. Artioli, L. Beirão da Veiga, and F. Dassi, *Curvilinear virtual elements for 2D solid mechanics applications*, Comput. Methods Appl. Mech. Engrg., 359 (2020), pp. 112667, 19.

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

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

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

[8] L. Beirão da Veiga, A. Chernov, L. Mascotto, and A. Russo, *Basic principles of hp virtual elements on quasiuniform meshes*, Math. Models Methods Appl. Sci., 26 (2016), pp. 1567–1598.

[9] L. Beirão da Veiga, A. Pichler, and G. Vacca, *A virtual element method for the miscible displacement of incompressible fluids in porous media*, Comput. Methods Appl. Mech. Engrg., 375 (2021), pp. 113649, 35.

[10] S. Berrone and A. Borio, *Orthogonal polynomials in badly shaped polygonal elements for the virtual element method*, Finite Elem. Anal. Des., 129 (2017), pp. 14–31.

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

[12] J. A. C. Herrera, R. Corrales and J.G.Calvo, *VEM Library.* `www.github.com/jgcalvo`, 2021.

[13] J. G. Calvo, *On the approximation of a virtual coarse space for domain decomposition methods in two dimensions*, Mathematical Models and Methods in Applied Sciences, 28 (2018), pp. 1267–1289.

[14] ———, *An overlapping Schwarz method for virtual element discretizations in two dimensions*, Comput. Math. Appl., 77 (2019), pp. 1163–1177.

[15] L. Chen, *iFEM: an integrated finite element methods package in MATLAB*, tech. report, University of California at Irvine, 2008.

[16] P. D., B. S., P. M., and L. M., *FETI-DP Preconditioners for the Virtual Element Method on General 2D Meshes*, in Numerical Mathematics and Advanced Applications ENUMATH 2017, F. A. Radu, K. Kumar, I. Berre, J. M. Nordbotten, and I. S. Pop, eds., vol. 126 of Lect. Notes Comput. Sci. Eng., Springer, Cham, 2019, pp. 157–164.

[17] G. H. Golub and J. H. Welsch, *Calculation of gauss quadrature rules*, Mathematics of Computation, 23 (1969), pp. 221–221.

[18] F. Lepe and G. Rivera, *A virtual element approximation for the pseudostress formulation of the Stokes eigenvalue problem*, Comput. Methods Appl. Mech. Engrg., 379 (2021), p. 113753.

[19] A. Ortiz-Bernardin, *VEMLab version 2.3*, Department of Mechanical Engineering, Universidad de Chile, Santiago, Chile, 2020.

[20] J. Sievers, *`VoronoiLimit(varargin)`*, 2019 (MATLAB Central File Exchange. Retrieved June 15, 2019.).

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

[22] I. The Mathworks, *MATLAB version 9.8.0.1451342 (R2020a)*, The Mathworks, Inc., Natick, Massachusetts, 2020.

[23] F. Wang and J. Zhao, *Conforming and nonconforming virtual element methods for a Kirchhoff plate contact problem*, IMA J. Numer. Anal., 41 (2021), pp. 1458–1483.

[24] Y. Zhu, *Auxiliary Space Preconditioners for Linear Virtual Element Method*, in Domain decomposition methods in science and engineering XXV, R. Haynes, S. MacLachlan, X.-C. Cai, L. Halpern, H. H. Kim, A. Klawonn, and O. Widlund, eds., vol. 138 of Lect. Notes Comput. Sci. Eng., Springer, Cham, 2020, pp. 383–390.