Programming of linear virtual element methods in three dimensions

Yue Yu

School of Mathematical Sciences, Institute of Natural Sciences, MOE-LSC, Shanghai Jiao Tong University, Shanghai, 200240, P. R. China.

Abstract
We present a simple and efficient MATLAB implementation of the linear virtual element method for the three dimensional Poisson equation. The purpose of this software is primarily educational, to demonstrate how the key components of the method can be translated into code.

Keywords: Polyhedral meshes, MATLAB implementation, Virtual element method, Three dimensions, Poisson equation

1. Introduction
The virtual element method (VEM), first introduced and analyzed in [1], is a generalization of the standard finite element method on general polytopal meshes. In the past few years, people have witnessed rapid progresses of virtual element methods (VEMs) for numerically solving various PDEs, see [2–5] for examples. One can refer to [6] for a transparent MATLAB implementation of the conforming 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 [7–10]. However, to the best of knowledge, no related implementation is publicly available in the literature.

In this paper, we are intended to present a simple and efficient MATLAB implementation of the linear virtual element method for the three dimensional Poisson equation on general polyhedral meshes. All important blocks are described in detail step by step. Although the current procedure is only for first-order virtual element spaces, the design ideas can be directly generalized to higher-order cases.

2. Virtual element methods for the 3-D Poisson equation
Let $\Omega \subset \mathbb{R}^3$ be a polyhedral domain and let $\Gamma$ denote a subset of its boundary consisting of some faces. We consider the following model problem

\[
\begin{aligned}
-\Delta u &= r \quad \text{in } \Omega, \\
u &= g_D \quad \text{on } \Gamma, \\
\partial_n u &= g_N \quad \text{on } \Gamma' = \partial \Omega \setminus \Gamma,
\end{aligned}
\]  

(2.1)

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.

Email address: terenceyue@sjtu.edu.cn (Yue Yu)
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 [10] for (2.1) is to find $u_h \in V^k_T$ such that

$$a_h(u_h, v_h) = \ell_h(v_h), \quad v_h \in V^0,$$

where

$$a_h(u_h, v_h) = \sum_{K \in T_h} a^K_h(u_h, v_h), \quad \ell_h(v_h) = \int_{\Omega} r_h v_h \, dx + \int_{\Gamma'} g_h v_h \, ds.$$

The local bilinear form is split into two parts:

$$a^K_h(v, w) = (\nabla v, \nabla w)_K + h_K S^K(v - \Pi_h \nabla v, w - \Pi_h w),$$

where $\Pi_h : V^k(K) \to \mathbb{P}_h(K)$ 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

$$\ell^K_h(v_h) = \int_K r \Pi_h v_h \, dx + \sum_{f \subset \Gamma' \cap \partial K} \int_f g_N \Pi^K_{h,f} v_h \, ds,$$

where $\Pi^K_{h,f} : V^k(f) \to \mathbb{P}_h(f)$ is the elliptic projector defined on the face $f$.

For the detailed introduction of the virtual element spaces, please refer to Section 2 in [10]. 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.

3. 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 $N \times 3$ 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. 1(a) is shown in Fig. 1(b), which is still represented by a cell array since the faces may have different numbers of vertices.

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

```matlab
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

$$\text{ErrL}^2 = \sum_{K \in T_h} \| u - \Pi^\nabla_1 u_h \|_{0,K}, \quad \text{ErrH}^1 = \sum_{K \in T_h} | u - \Pi^\nabla_1 u_h |_{1,K}$$

by using respectively the subroutines `getL2error3.m` and `getH1error3.m`. The procedure is completed by verifying the rate of convergence through `showrateErr.m`.

```matlab
%% Parameters
maxIt = 3;
for k = 1:maxIt
    % load mesh
    load(['mesh3data', num2str(k), '.mat']);
    % get boundary information
    bdStruct = setboundary3(node3, elem3, bdNeumann);
    % solve
    [uh, info] = PoissonVEM3(node3, elem3, pde, bdStruct);
    % compute errors in discrete L2, H1 and energy norms
end
```

Fig. 1. Polyhedral mesh of a cube
In the following sections, we shall go into the details of the implementation of the 3-D VEM in PoissonVEM3.m.

4. 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 \( \Pi^{V_1}_{(f)} \phi_f \) ready in advance, where \( \phi_f \) is the nodal basis of the enhanced virtual element space \( V^1(f) \) (see Subsection 2.3 in [10]). To this end, it may be necessary to establish local coordinates \((s,t)\) on the face \( f \).

As shown in Fig. 2, the boundary of the polygon is oriented in a counterclockwise order as \( a_1, \cdots, a_n \). Let \( e = a_1 a_2 \) 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

\[
(s, t) = (x - x_1, y - y_1, z - z_1) \begin{bmatrix} n_1 & n_2 & n_3 \\ t_1 & t_2 & t_3 \end{bmatrix}^{-1},
\]

with the inverse understood in the least squares sense. When translating 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 $\phi_1^T = [\phi_{f,1}, \ldots, \phi_{f,n}]$ 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

\[
\begin{aligned}
(G_f \Pi_{1,f}^\nabla \phi_f^T) = (\nabla f m_f, \nabla f \phi_f^T)_f, \\
P_0(\Pi_{1,f}^\nabla \phi_f^T) = P_0(\phi_f^T).
\end{aligned}
\tag{4.1}
\]

where

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

Since $P_1(f) \subset V^1(f)$, we can write

\[
m_f^T = \phi_f^T D_f, \quad D_f = (D_\alpha), \quad D_\alpha = \chi_{f,i}(m_{f,\alpha}),
\]

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

\[
\Pi_{1,f}^\nabla \phi_f^T = \phi_f^T \Pi_{1,f}^\nabla, \quad \Pi_{1,f}^\nabla \phi_f^T = m_f^T \Pi_{1,*,f}^\nabla.
\]

One easily finds that

\[
\Pi_{1,f}^\nabla = D_f \Pi_{1,*,f}^\nabla
\]

and (4.1) can be rewritten in matrix form as

\[
\begin{aligned}
(G_f \Pi_{1,*,f}^\nabla) = B_f, \\
P_0(m_f^T) \Pi_{1,*,f}^\nabla = P_0(\phi_f^T),
\end{aligned}
\]

where

\[
G_f = (\nabla f m_f, \nabla f m_f^T)_f, \quad B_f = (\nabla f m_f, \nabla f \phi_f^T)_f.
\]

Note that the following consistency relation holds

\[
G_f = B_f D_f, \quad \Pi_{1,f}^\nabla = D_f.
\]

Let $\text{face}$ 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

\[
\text{Pi} = \text{faceEllipticProjection}(P),
\]

where $P$ is the coordinates of the face $f$ and $\text{Pi}$ is the matrix representation $\Pi_{1,*,f}^\nabla$ 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 $\text{face}$. 

5
Derive elliptic projections of all faces

```matlab
for s = 1:NF
    faces = face(s); P = node3(faces,:);
    Pifs = faceEllipticProjection(P);
    [~,idx] = sort(faces);
    faceProj(s) = Pifs(:,idx);
end
```

Note that in the last step we sort the columns of $\Pi_{1s,f}^{\nabla}$ 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 34-38 in the code of the next section).

The face integral is then given by

$$
\int_f \Pi_{1s,f}^{\nabla} \phi_f^T d\sigma = \int_f m_f^T d\sigma \Pi_{1s,f}^{\nabla} = (|f|, 0, 0) \Pi_{1s,f}^{\nabla},
$$

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

5. 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 $D, G, \tilde{G}, B$ and $\tilde{B}$. For example, the transition matrix is given by

$$
D = (D_\alpha), \quad D_\alpha = \chi_i(m_\alpha) = m_\alpha(p_i).
$$

The most involved step is to compute the matrix

$$
B = \int_K \nabla m \cdot \nabla \phi^T dx = -\int_K \Delta m \cdot \phi^T dx + \sum_{f \subset \partial K} \int_f (\nabla m \cdot n_f) \phi^T d\sigma
$$

$$
= \sum_{f \subset \partial K} (\nabla m \cdot n_f) \phi^T d\sigma,
$$

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

$$
\int_f (\nabla m \cdot n_f) \phi^T d\sigma = (\nabla m \cdot n_f) \int_f \phi^T d\sigma = (\nabla m \cdot n_f) \int_f \Pi_{1f}^{\nabla} \phi^T d\sigma,
$$

and the last term is available from (4.2). Obviously, for the vertex $p_i$ away from the face $f$ there holds $\Pi_{1f}^{\nabla} \phi_i = 0$. 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.
% ------- element information --------

% faces
elemf = elem3(iel); indexFace = elem2face(iel);
% global index of vertices
index3 = unique(horzcat(elemf{:}));
% local index of elemf
Idx = min(index3):max(index3);
elemLocal = cellfun(@(face) Idx(face), elemf, 'UniformOutput', false);
% centroid and diameter
Nv = length(index3); Ndo = Nv;
V = node3(index3,:);
xK = centroid3(iel,1); yK = centroid3(iel,2); zK = centroid3(iel,3);
hK = diameter3(iel);
% ------- 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
gradMat = [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,Ndo);
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] = sort(faces);
Pifs = Pifs(:,idx);
% normal vector
e1 = P(2,:) - P(1,:); en = P(1,:) - P(end,:);
f = cross(e1,en); nf = f/norm(f);
% area
areaF = polyarea3(P);
% --- integral of Pifs
intFace = [areaF,0,0]*Pifs; % local
intProj = zeros(1,Ndo); % global adjustment
faceLocal = elemfLocal{s};
intProj(faceLocal) = intFace;
% add grad(m)*nf
Bf = dot(gradMat, repmat(nf,4,1),2)*intProj;
B = B + Bf;
end
% constraint
Bs = B; Bs(1,:) = 1/Ndo;
% consistency relation
G = B*B; Gs = Bs*D;
6. Computation of the right hand side and assembly of the linear system

The right-hand side is approximated as

\[ F_K = \int_K f \Pi_Y \phi dx = (\Pi_Y^\top)^T \int_K f m dx, \]

where \( \Pi_Y \) is the elliptic projector on the element \( K \) and \( \Pi_Y^\top \) is the matrix representation in the basis \( m^T \). The integral \( \int_K f m^T dx \) can be approximated by

\[ \int_K f m^T dx = |K| f(x_K) m(x_K) = |K| f(x_K)[1,0,0,0]^T, \quad x_K = (x_K, y_K, z_K). \]

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

\[ A_K = (\Pi_Y^\top)^T G \Pi_Y + h_K (I - \Pi_Y^\top)^T (I - \Pi_Y). \]

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.

```matlab
for iel = 1:NT
  % --------- local stiffness matrix ---------
  Pis = Gs\Bs; Pi = D*Pis; I = eye(size(Pi));
  AK = Pis'*G*Pis + hK*(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 elliptic projection -----------
  indexDof = index3;
  ii(ia+1:ia+Ndof^2) = reshape(repmat(indexDof,Ndof,1),[],1);
  jj(ia+1:ia+Ndof^2) = repmat(indexDof(:,1),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 \( \Pi_Y^\top \) and the assembly index `elem2dof` in the M-file so as to compute the discrete \( L^2 \) and \( H^1 \) errors.
7. 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

$$F_f = \int_g g_N \Pi_{1,f} \phi_f \, d\sigma = (\Pi_{1,f})^T \int_g g_N m_f \, d\sigma,$$

where $g_N = \partial_n u = \nabla u \cdot n_f$. 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.

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

The Dirichlet boundary conditions are easy to handle. The code is given as follows.
In the above codes, \texttt{bdStruct} stores all necessary information of boundary faces. We finally derive the linear system \( \mathbf{k} \mathbf{k}^* \mathbf{u} = \mathbf{f} \), where \( \mathbf{k} \) is the resulting coefficient matrix and \( \mathbf{f} \) 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.

1
2
3
4
5
6
7
8
9
10

Here, the subroutine \texttt{amg.m} can be found in iFEM — a MATLAB software package for the finite element methods [11].

8. Summary of the code

The complete M-file is \texttt{PoissonVEM3.m}. For the sake of the length, we only provide the main structure:

| \texttt{function} \[u, \text{info}\] = PoissonVEM3(node3, elem3, pde, bdStruct) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 |
| \% Get auxiliary data |
| 5 | 6 | 7 | 8 |
| \% Derive elliptic projections of all faces |
| 9 | 10 | 11 | 12 |
| \% Compute and assemble the linear system |
| 13 | 14 | 15 | 16 |
| \% Assemble Neumann boundary conditions |
| 17 | 18 | 19 | 20 |
| \% Apply Dirichlet boundary conditions |
| 21 | 22 | 23 | 24 |
| \% Set solver |
| 25 | 26 | 27 | 28 |
| \% Store information for computing errors |

To execute the M-file, one can refer to the example file \texttt{main_PoissonVEM3.m}. We provide several simple meshes, such as mesh3data1.mat and mesh3data2.mat. All examples are implemented in MATLAB R2019b. The results for the test case 1 in [10] are displayed in Fig. 3 and Tab. 1, from which we observe the optimal rate of convergence both for the \( H^1 \) norm and \( L^2 \) norm.

Tab. 1. Discrete \( L^2 \) and \( H^1 \) errors for polyhedral meshes

| \( N \) | \( h \) | \( \text{ErrL2} \) | \( \text{ErrH1} \) |
|---|---|---|---|
| 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 |

Our code is available from GitHub (https://github.com/Terenceyuyue/mVEM) as part of the mVEM package which contains efficient and easy-following codes for various VEMs published in the literature. The test script \texttt{main_PoissonVEM3.m} is stored in \texttt{vem3} folder for the current version.
Fig. 3. Convergence rates in $L^2$ and $H^1$ norms for the 3-D Poisson equation

Conflict of Interest

The authors declare that they have no conflict of interest.

References

[1] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

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

[9] H. Chi, L. Beirão Da Veiga, and G. Paulino. Some basic formulation of the virtual element method (VEM) for finite deformations. *Math. Models Methods Appl. Sci.*, 318(99):148–192, 2017.

[10] 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.

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