Computers and Mathematics with Applications 180 (2025) 144–165

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

Vectorized implementation of primal hybrid FEM in MATLAB

Harish Nagula Mallesham [a], Kamana Porwal [b], Jan Valdman [c,d], Sanjib Kumar Acharya [a,*]

[a] Institute of Chemical Technology Mumbai, IndianOil Odisha Campus, Bhubaneswar, Odisha, 751013, India
[b] Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, Delhi, 110016, India
[c] Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 16000 Prague, Czech Republic
[d] The Czech Academy of Sciences, Institute of Information Theory and Automation, Pod vodárenskou věží 4, 18208, Prague 8, Czech Republic

ARTICLE INFO

MSC:
35-04
65N30
65N06
35J25
35K20
35K57

Keywords:
Finite elements
Primal hybrid method
Elliptic problem
Parabolic problem
Vectorization
MATLAB

ABSTRACT

We present efficient MATLAB implementations of the lowest-order primal hybrid finite element method (FEM) for linear second-order elliptic and parabolic problems with mixed boundary conditions in two spatial dimensions. We employ backward Euler and the Crank-Nicolson finite difference scheme for the complete discrete setup of the parabolic problem. All the codes presented are fully vectorized using matrix-wise array operations. Numerical experiments are conducted to show the performance of the software.

1. Introduction

MATLAB is a popular computing platform in academia and industry, with many built-in functions and toolboxes developed by Mathworks Computer Software Corporation. However, codes containing for loops, particularly the assembly of finite element stiffness and mass matrices in MATLAB are extremely slow compared to C and FORTRAN (cf. [1,2]). To overcome this, Rahman and Valdman [3] introduced an efficient and flexible MATLAB assembly procedure of nodal element stiffness and mass matrices for 2D and 3D problems. They vectorized for-loops by extending the element-wise array operations into matrix-wise array operations, where the array elements are matrices rather than scalars, resulting in a faster and more time-scalable algorithm. This idea is exploited for the MATLAB assembly of 2D and 3D edge elements by Anjam and Valdman in [4]. In [5], Funken et al. adopted a similar approach while assembling the element stiffness matrices. For more detailed literature, we refer to Cuvelier et al.; see [6], Moskovka and Valdman [7] and the references therein.

The code (and data) in this article has been certified as Reproducible by Code Ocean: https://codeocean.com/.

* Corresponding author.
E-mail addresses: mat21h.nagula@stuiocb.ictmumbai.edu.in (H. Nagula Mallesham), kamana@maths.iitd.ac.in (K. Porwal), jan.valdman@utia.cas.cz (J. Valdman), sk.acharya@iocb.ictmumbai.edu.in (S.K. Acharya).

https://doi.org/10.1016/j.camwa.2024.12.017
Received 22 March 2024; Received in revised form 8 December 2024; Accepted 16 December 2024

Available online 30 December 2024

We present an efficient MATLAB implementation procedure of the primal hybrid finite element method, which is based on an extended variational principle introduced by Raviart and Thomas in [8] for elliptic problems. Acharya and Porwal [9,10] extended this technique to parabolic second- and fourth-order problems and established optimal-order error estimates. The benefit of this method is that two unknown variables (primal and hybrid) can be computed simultaneously without losing accuracy. The constrained space using Lagrange multipliers leads to a non-conforming space; for instance, the lowest-order primal hybrid FEM (using unconstrained space) is equivalent to the Crouzeix-Raviart nonconforming (using constrained space) finite element space. It is also called the generalized nonconforming method. For a detailed review of the literature and applications, see [9–13] and the references therein. Our main contributions are as follows.

1. Edge generation function edge.m (cf. [2]) and a uniform mesh refinement function based on red-refinement (cf. [14,15]) have been vectorized using matrix-wise array operations.
2. Vectorized implementation of the lowest-order primal hybrid FEM for a second-order general elliptic problem with mixed boundary conditions is presented.
3. Vectorized implementation for the general second-order parabolic problem with mixed boundary conditions using backward Euler and the Crank-Nicolson finite difference scheme in the temporal direction is presented. This directly allows for the extension of the calculations in [12].

Numerical experiments have been conducted, and run-time has been presented, confirming linear-like time-scaling of our implementations. The experiments were carried out using MATLAB R2020a on a computer equipped with a i5 − 10210U CPU operating at a frequency of 1.60GHz-2.10 GHz, with 16GB RAM and a x64-based processor with 1TB of system memory. The software can be downloaded through the following links: https://codeocean.com/capsule/5776148/tree [16] or https://www.mathworks.com/matlabcentral/fileexchange/136359.

The outline of this paper is as follows. In Section 2, we present the data structure required to present the primal hybrid FEM and its implementation procedures. The primal hybrid finite element algebraic formulation for the elliptic problem and its MATLAB implementation have been discussed in Section 3. The implementation of the primal hybrid FEM with the backward Euler and Crank-Nicolson scheme in the time direction for parabolic problems is presented in Section 4. We give our concluding remarks in Section 5.

2. Triangulation and geometric structure

Let $\Omega = (0,1)^2$ be the computational domain with boundary $\Gamma = \overline{\Gamma}_D \cup \overline{\Gamma}_N$, where $\overline{\Gamma}_D$ and $\Gamma_N = \Gamma / \overline{\Gamma}_D$ are the Dirichlet and Neumann boundary, respectively. We use the Sobolev spaces $H^m(\Omega)$, $H(\mathrm{div}, \Omega)$ in the domain Ω, $m \in \mathbb{R}_{\pm}$ [17]. With the mesh parameter h, let $\mathcal{T}_h$ be a regular family of triangulations of the set $\overline{\Omega}$ in the sense of Ciarlet [17] with shape regular triangles T whose diameters $h_T \le h$ are such that $\overline{\Omega} = \bigcup_{T \in \mathcal{T}_h} \overline{T}$.

For any triangle T, let ∂T denote the boundary of T and let ν_T be the outward unit normal to ∂T. Let

$$\mathcal{E}^h = \mathcal{E}^h_\Omega \cup \mathcal{E}^h_D \cup \mathcal{E}^h_N$$

be the set of all edges E of $\mathcal{T}_h$ where $\mathcal{E}^h_\Omega$ denotes the set of all interior element edges,

$$\mathcal{E}^h_D = \{ E \in \mathcal{E}^h : E \subset \overline{\Gamma}_D \}$$

denotes the set of edges on the Dirichlet boundary and

$$\mathcal{E}^h_N = \{ E \in \mathcal{E}^h : E \subset \overline{\Gamma}_N \}$$

denotes the set of edges on the Neumann boundary. The following notations will be used throughout the article:

$$\begin{aligned}
\mathcal{N} &= \text{set of vertices of element } T \in \mathcal{T}_h, & \mathtt{nE} &= \mathrm{card}(\mathcal{T}_h),\\
m_E &= \text{mid-point of } E \in \mathcal{E}^h, & N &= 3 \times \mathtt{nE},\\
\mathcal{N}_m &= \text{set of mid-points } m_E \text{ of the edges } E \in \mathcal{E}^h, & L &= \mathrm{card}(\mathcal{E}^h_\Omega \cup \mathcal{E}^h_D),\\
\mathcal{N}_m(\Gamma_D) &= \text{set of midpoints of the edges } E \in \mathcal{E}^h_D, & \mathtt{nEd} &= \mathrm{card}(\mathcal{E}^h),\\
\mathcal{N}_m(\Gamma_N) &= \text{set of midpoints of the edges } E \in \mathcal{E}^h_N, & N_D &= \mathrm{card}(\mathcal{E}^h_D),\\
\mathrm{card}(S) &= \text{cardinality of the set } S, & N_\Omega &= \mathrm{card}(\mathcal{E}^h_\Omega),\\
\mathrm{conv}\{A,B\} &= \text{convex hull of } A \text{ and } B, & N_n &= \mathrm{card}(\mathcal{E}^h_N).
\end{aligned}$$

The intersection of two distinct triangles of $\mathcal{T}_h$ is either empty or non-empty. They share one complete edge $E = \mathrm{conv}\{z_1, z_2\}$ or a node z_1 in the non-empty case. Fig. 1 depicts two adjacent triangles T_+ and T_- with a common edge $E = \mathrm{conv}\{A, B\}$. Let $\mathcal{N} = \{z_1, \ldots, z_{\mathrm{card}(\mathcal{N})}\}$ be the set of vertices (nodes) z_l, $1 \le l \le \mathrm{card}(\mathcal{N})$ with coordinates $(x_l, y_l) \in \mathbb{R}^2$.

For an edge E with initial node z_i and end node z_j, its outward unit normal is defined as:

$$\nu_E = (y_j - y_i, x_i - x_j)/|E|,$$

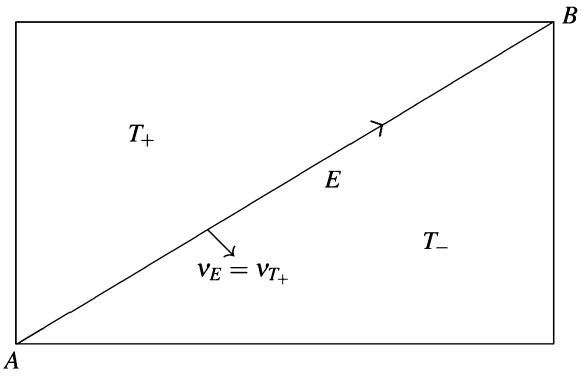

Fig. 1. A pair of adjacent triangles (T_+, T_-) with a common edge $E = \partial T_+ \cap T_-$, initial node A, end node B and outward unit normal $\nu_E = \nu_{T_+}$.

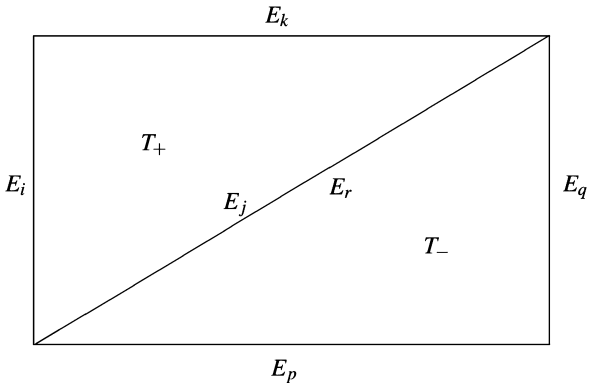

Fig. 2. Depiction of a general numbering of edges in a mesh with two adjacent triangles.

where $|E|$ is the length of E.

Let $E \in \mathcal{E}_\Omega^h$ be such that $E = \partial T_+ \cap \partial T_-$ where T_+ and T_- are adjacent triangles in $\mathcal{T}_h$ (as shown in Fig. 1) with $\nu_{T_+} = \nu_E$ denoting the unit normal of E pointing from T_+ to T_- and $\nu_{T_-} = -\nu_E$, we define the jump of a scalar-valued function v on the edge E as

$$[\![v]\!]_E = (v|_{T_+})|_E - (v|_{T_-})|_E.$$

Further, for an edge $E \in \mathcal{E}_D^h$ such that $E = \partial T \cap \overline{\Gamma}_D$ for some $T \in \mathcal{T}_h$, we define $[\![v]\!]_E = (v|_T)|_E$.

Let i, j, k represent the global numbering of nodes of a triangle $T = \text{conv}\{z_i, z_j, z_k\}$. Let E_i, E_j, E_k be edges of triangle T_+ and E_p, E_q, E_r be edges of triangle T_- for two adjacent triangular elements $T_+ = \text{conv}\{z_i, z_j, z_k\}$ and $T_- = \text{conv}\{z_p, z_q, z_r\}$ in $\mathcal{T}_h$ as in Fig. 2.

For a triangular element, $T = \text{conv}\{P_1, P_2, P_3\}$, where

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2), \quad P_3 = (x_3, y_3)$$

are vertices of T, opposite edges are

$$E_1 = \text{conv}\{P_2, P_3\}, \quad E_2 = \text{conv}\{P_1, P_3\}, \quad E_3 = \text{conv}\{P_1, P_2\},$$

mid-point of edges are

$$mE_1 = (P_2 + P_3)/2, \quad mE_2 = (P_1 + P_3)/2, \quad mE_3 = (P_1 + P_2)/2,$$

and area of the element is denoted as $|T|$.

2.1. Data representation of the triangulation

We present the data structures for the initial triangulation of Ω with triangles in Table 1 and Fig. 3. The nodes and elements are stored in the rows of matrices `"coordinates.dat"` and `"elements.dat"`, respectively. Information about the boundary edges in $\mathcal{E}_D^h$ and $\mathcal{E}_N^h$ is stored in `"Dirichlet.dat"` and `"Neumann.dat"`, respectively.

Table 1
`coordinates.dat`, `elements.dat`, `Neumann.dat` (consists of 2 boundary edges which are plotted in red) and `Dirichlet.dat` (remaining 2 boundary edges) data for the triangulation displayed in Fig. 3.

coordinates.dat	
0	0
1	0
0	1
1	1
0.5	0.5

elements.dat		
5	1	2
5	3	1
5	4	3
5	2	4

Dirichlet.dat	
1	2
2	4

Neumann.dat	
4	3
3	1

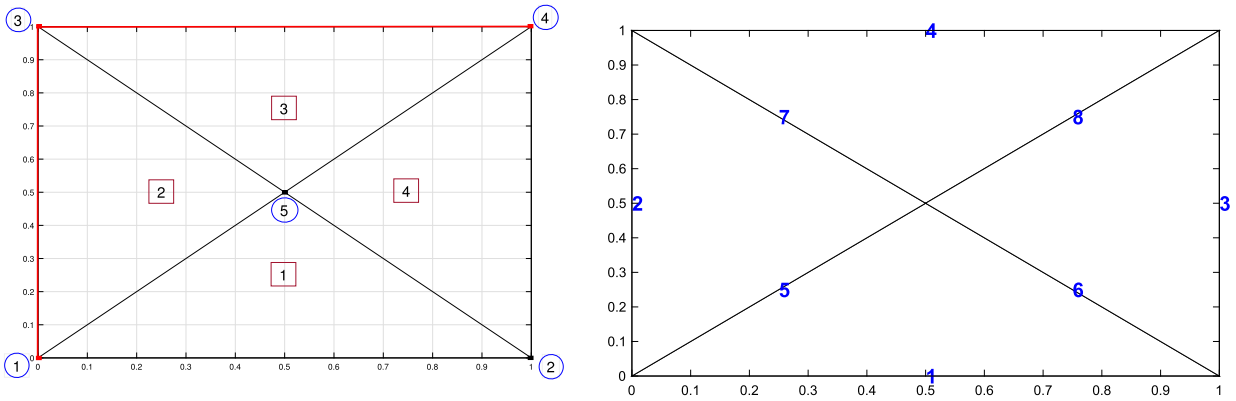

Fig. 3. Depiction of initial (level 0) triangulation (left) and the corresponding edges generated (right). The Neumann edges are plotted in red.

2.2. *Vectorization of* **edge.m**

In this subsection, we present a modified version of the function **edge.m** (whose inputs are `elements` and `coordinates` and outputs are `nodes2element(n2el)`, `nodes2edge(n2ed)` and `edge2element(ed2el)` (cf. Bahriawati and Carstensen, [2]). The following example intuitively explains the meaning of output matrices.

Example 1. Given the triangulation of Fig. 3, we have the following full versions of sparse matrices

$$\texttt{nodes2element} = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 4 & 1 \\ 2 & 0 & 0 & 0 & 3 \\ 0 & 0 & 3 & 0 & 4 \\ 1 & 4 & 2 & 3 & 0 \end{pmatrix}, \qquad \texttt{nodes2edge} = \begin{pmatrix} 0 & 1 & 2 & 0 & 5 \\ 1 & 0 & 0 & 3 & 6 \\ 2 & 0 & 0 & 4 & 7 \\ 0 & 3 & 4 & 0 & 8 \\ 5 & 6 & 7 & 8 & 0 \end{pmatrix}. \tag{2.1}$$

The 1st row of `nodes2element` tells us that:

the directed edge $1 \to 2$ belongs to element 1,
the directed edge $1 \to 5$ belongs to element 2.

All directed edges are oriented counterclockwise in their elements. Therefore, the directed edge $3 \to 1$ is stored in the third raw and belongs to element 2. There is no directed edge $1 \to 3$.

The symmetric matrix `nodes2edge` provides actual indices of undirected edges. The first raw of `nodes2edge` reveals that:

the undirected edge $1 \leftrightarrow 2$ has number 1,
the undirected edge $1 \leftrightarrow 3$ has number 2,
the undirected edge $1 \leftrightarrow 5$ has number 5.

Finally, the matrix `edge2element` contains nodes belonging to the edges (the first two columns) and the connection of the edges to the elements (the remaining two columns). It reads

```
nE=size(elements,1); %number of elements
nC=size(coordinates,1); % number of nodes
I=elements(:);
J=reshape(elements(:,[2 3 1]), numel(elements), 1);
S=reshape([1:nE;1:nE;1:nE]', numel(elements), 1);
nodes2element=sparse(I, J, S, nC, nC);
```

Listing 1: Vectorization of the `nodes2element` matrix.

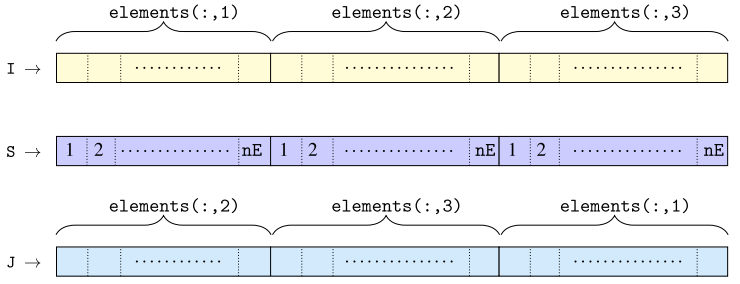

Fig. 4. Geometrical representation of `I`, `J`, and `S` vectors.

$$\texttt{edge2element} = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 2 & 0 \\ 2 & 4 & 4 & 0 \\ 4 & 3 & 3 & 0 \\ 5 & 1 & 1 & 2 \\ 2 & 5 & 1 & 4 \\ 5 & 3 & 2 & 3 \\ 5 & 4 & 3 & 4 \end{pmatrix}. \tag{2.2}$$

The fifth row of `edge2element` tells us that:

the undirected edge $5 \leftrightarrow 1$ belongs to elements 1, 2.

Clearly, the rows of `edge2element` with the fourth column entry equal to zero describe boundary edges that belong to one element only.

By removing for-loops of **edge.m** function, we provide a modified function:
`[nodes2element, nodes2edge, edge2element]` = **edge_vec**`(elements, coordinates)`
in which the interior edges (`intE`), exterior edges (`extE`), local edge numbering data for T_+ elements (`led`), and local edge numbering data for T_- elements (`ledTN`) can additionally be output.

2.2.1. *nodes2element*

The `nodes2element` function describes an element's global number as a function of its two vertices, with dimension $\text{card}(\mathcal{N}) \times \text{card}(\mathcal{N})$ defined as,

$$\texttt{nodes2element}\,(k,l) = \begin{cases} j & \text{if } (k,l) \text{ are numbers of the nodes of element number } j; \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

For the matrix `nodes2element`, the vectorized computation in MATLAB is as follows:

Here `nodes2element` is a sparse matrix from the triplets `I`, `J`, and `S` such that, `nodes2element( I(`k`), J(`k`))` = `S(`k`)`, where `I`, `J` and `S` are column-vectors as shown in Fig. 4.

The local edge numbering of element T is $\{E_3,\ E_1,\ E_2\}$, that is $[3,1,2]$ in **edge_vec.m**. The following line computes the local edge numbering $\forall T \in \mathcal{T}_h$ using vectors `I`, `J` in Fig. 4.

```
[~,~,localE]=find(sparse(I,J,repmat([3,1,2],nE,1),nC,nC)); % position of edge in element
```

2.2.2. *nodes2edge*

The sparse matrix `node2edge` describes the global number of edges, with dimension $\text{card}(\mathcal{N}) \times \text{card}(\mathcal{N})$ defined as,

$$\texttt{nodes2edge}\,(k,l) = \begin{cases} j & \text{if } j-\text{th edge is connected between the} \\ & \text{two nodes with global numbering } (k,l); \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

```
B=nodes2element+nodes2element';
[i,j]=find(triu(B));
nodes2edge=sparse(i,j,1:size(i,1),nC,nC);
nodes2edge=nodes2edge+nodes2edge';
```

Listing 2: Vectorization of the `nodes2edge` matrix.

```
%% to generate element of edge
[Inode,Enode,ielems]=find(nodes2element);
nodes2elementSparsity=sparse(Inode,Enode,ones(size(Inode,1),1),nC, nC);
nodes2edgesDirected=nodes2edge.*nodes2elementSparsity;
[~,~,iedges]=find(nodes2edgesDirected);
data=[iedges ielems, Inode, Enode,localE];        % edge, element, Inode, Enode, position in element
dataSort=sortrows(data);
idxl=[true; diff(dataSort(:,1))>0];
edge2element=zeros(nEd,4);
edge2element(:,1:3)=dataSort(idxl,[3 4 2]);   % first occurence of an edge
led=dataSort(idxl,5); ledTN=led;
IdxTN=find(idxl==0);
edge2element(dataSort(IdxTN,1),4)=dataSort(IdxTN,2); % repetition of the edge
ledTN(dataSort(IdxTN,1),1)=dataSort(IdxTN,5);
intE=find(edge2element(:,4)); % to produce the interior edges
extE=find(edge2element(:,4)==0); % to produce the exterior edges
```

Listing 3: Vectorization of the `edge2element` matrix.

The MATLAB code for the calculation of the matrix `nodes2edge` is as follows: In the above code, the variables `i` and `j` represent the row and column indices for the nonzero entries of the upper triangular part of the matrix `B`. The number of edges is denoted by `nEd=size(i,1)`.

2.2.3. *edge2element*

The row number $p1$ of the matrix `edge2element` represents the edge's initial node k and end node l, as well as the numbers m, n of elements T_+, T_- that share the edge, and the matrix `edge2element` has a dimension of `nEd` $\times 4$.

$$\texttt{edge2element}\ (p1,[1,2]) = [k\ \ l] \qquad (k,l) \text{ are initial node and end node of edge } p1.$$

The Fig. 1 convention is used to specify the neighbor elements T_+, T_- with indices m and n. The entry of the matrix `edge2element`($p1$, 3) defines the element T_+, which also specifies the orientation of the unit normal $\nu_E = \nu_{T_+}$ of the edge $E = T_+ \cap T_-$. If the edge $E \in \mathcal{E}_D^h \cup \mathcal{E}_N^h$, then the fourth entry `edge2element`($p1$,4)) is zero, indicating that ν_E lies on $\partial\Omega$.

$$\texttt{edge2element}\ (p1,[3,4]) = \begin{cases} [m\ n] & \text{if the common edge } p1 \text{ belongs to element } m \text{ and } n; \\ [m\ 0] & \text{if the boundary edge } p1 \text{ belongs to element } m. \end{cases} \tag{2.5}$$

The MATLAB code for the vectorized calculation of the matrix `edge2element` is as follows: In the above code for $\forall E \in \mathcal{E}^h$,

- Line 2-5:
  - The vectors `Inode` and `Enode`, specify the initial and end nodes of edge E through their respective indices,
  - The vectors `ielems` and `iedges` contain element and edge indices corresponding to the edge E.
- Line 6-14: The matrix `data` concatenates the `iedges`, `ielems`, `Inode`, `Enode`, and `localE`, potentially resulting in the repetition of edges.
  - With the help of unique edges, we compute the local edge numbering for T_+ elements in `led` and `edge2element`$(:, 1:3)$.
  - With the help of repeated edges, we calculate the local edge numbering for T_- elements in `ledTN` and `edge2element`$(:, 4)$.

2.3. *Vectorization of* **redrefine.m**

The function **redrefine_vec.m** performs uniform refinement of the computational domain.

- Line 5-7: The index of all new nodes is stored in a vector called `Marker`. The new midpoint coordinates are added to `coordinates` data by using concatenation.
- Line 8-13: For all triangles in the original mesh, three new triangles are created by the mid-point of the edges. Mark the new nodal numbers in `M1`, `M2`, and `M3` vectors. The connectivity of all these new triangles is stored in the `elements` data.
- Line 14-23: If there are $\mathcal{E}_D^h$ in the original mesh, a new node is created at the mid-point of the Dirichlet edge. Then, we find the corresponding new mid-point node for each Dirichlet edge using `Marker`. Update the `Dirichlet` data with these new nodes. For the Neumann boundary, the code does the same thing as for the Dirichlet boundary, but with the `Neumann` data.

```
function [coordinates,elements,Dirichlet,Neumann]= ...
    redrefine_vec(coordinates,elements,n2ed,ed2el,Dirichlet,Neumann)
nE=size(elements,1); % number of elements
nEd=size(ed2el,1); % number of edges
%% Coordinates
Marker=(size(coordinates,1)+1:size(coordinates,1)+nEd)';
coordinates=[coordinates;(coordinates(ed2el(:,1),:)+coordinates(ed2el(:,2),:))/2];
%% Elements
M1= Marker(n2ed(sub2ind(size(n2ed),elements(:,2),elements(:,3))));
M2= Marker(n2ed(sub2ind(size(n2ed),elements(:,3),elements(:,1))));
M3= Marker(n2ed(sub2ind(size(n2ed),elements(:,1),elements(:,2))));
elements(reshape(nE+1:4*nE,3,nE)',:) = [elements(:),[M3;M1;M2],[M2;M3;M1]];
elements((1:nE),:)=[M1 M2 M3];
%% Dirichlet and Neumann boundary
if (~isempty(Dirichlet)), Dirichlet=refineEdges(Dirichlet); end
if (~isempty(Neumann)), Neumann=refineEdges(Neumann); end
%% subfunction refineEdges
    function ed2n=refineEdges(ed2n)
        nEd=size(ed2n,1); % number of edges
        M=Marker(n2ed(sub2ind(size(n2ed),ed2n(:,1),ed2n(:,2))));
        ed2n(nEd+1:2*nEd,:)=[M ed2n(1:nEd,2)];
        ed2n(1:nEd,2)=M;
    end
end
```

Listing 4: Vectorization of redrefine function.

Table 2
Run-times: Vectorized code of **redrefine.m** and **edge.m**.

Level	nE	nEd	run-time (in sec)	
			redrefine_vec.m	**edge_vec.m**
4	1024	1568	0.001	0.002
5	4096	6208	0.001	0.003
6	16384	24704	0.002	0.013
7	65536	98560	0.006	0.049
8	262144	393728	0.025	0.245
9	1048576	1573888	0.140	0.974
10	4194304	6293504	0.507	3.986

2.4. *Run-time of* **redrefine_vec.m** *and* **edge_vec.m**

Table 2 presents the run-time of vectorized versions of the functions **redrefine.m** and **edge.m** in MATLAB, as we increase the level of refinement. The level 0 corresponds to the initial triangulation of the domain (Fig. 3), which consists of four triangular elements. In contrast, level 1 is the first uniform refinement of level 0, resulting in 16 triangular elements. For any $n \in \mathbb{N}$, level n is the uniform refinement of level $n-1$, and it contains 4^{n+1} triangular elements.

Remark 1. Run-times for non-vectorized codes are much higher compared to vectorized codes; for instance, the run-times of **redrefine.m** and **edge.m** are 2861.10 and 17114.35 seconds, respectively, at level 9.

3. Implementation of second order elliptic model problem

3.1. *Problem formulation*

Consider the general second-order elliptic problem

$$-\nabla \cdot (A\nabla u + up) + \delta u = f \qquad \text{in } \Omega, \tag{3.1}$$

$$u = u_D \qquad \text{on } \Gamma_D, \tag{3.2}$$

$$(A\nabla u + up)\cdot \nu = g \qquad \text{on } \Gamma_N, \tag{3.3}$$

where A is a symmetric and positive definite matrix whose entries are constants, p is a constant vector and δ is a non-negative constant. Furthermore, f, u_D, and g are assumed to be sufficiently smooth.

We consider the broken Sobolev space (cf. [8]), i.e., a space of functions that belong to Sobolev space $H^1(T)$ on each element T,

$$X = \{v \in L^2(\Omega) : v|_T \in H^1(T), \forall T \in \mathcal{T}_h\} = \prod_{T\in\mathcal{T}_h} H^1(T).$$

The space X is equipped with the norm

$$\|v\|_X = \left(\sum_{T\in\mathcal{T}_h} \|\nabla v\|^2_{L^2(T)} + h_T^{-2}\|v\|^2_{L^2(T)} \right)^{\frac{1}{2}}.$$

We shall denote by $H^{1/2}(\Gamma)$ the space of traces $v|_\Gamma$ over Γ of the functions $v \in H^1(\Omega)$ and $H^{-1/2}(\Gamma)$ denotes its dual space, and $\langle\cdot,\cdot\rangle_E$ denotes the corresponding duality pairing.

The Lagrange multiplier space is defined as

$$M = \left\{ \chi \in \prod_{T\in\mathcal{T}_h} H^{-1/2}(\partial T/\Gamma_N) : \text{there exists } \mathbf{q}\in H(\mathrm{div},\Omega) \text{ such that, } \mathbf{q}\cdot\nu_T = \chi \text{ on } \partial T/\Gamma_N, \forall T\in\mathcal{T}_h \right\}.$$

The primal hybrid formulation of (3.1)-(3.3) is to seek a pair of solutions $(u,\kappa)\in X\times M$ such that

$$a(u,v) + b(v,\kappa) = (f,v) + \langle g, v\rangle_{\Gamma_N} \qquad \forall v\in X, \tag{3.4}$$

$$b(u,\chi) \qquad = \langle u_D, \chi\rangle, \qquad \forall \chi\in M, \tag{3.5}$$

where

$$a(u,v) = \sum_{T\in\mathcal{T}_h}\int_T ((A\nabla u + up)\cdot\nabla v + \delta uv)\,\mathrm{d}x,$$

$$b(v,\chi) = -\sum_{E\in\mathcal{E}^h/\mathcal{E}^h_N} \langle \chi, v\rangle_E,$$

$$\langle u_D,\chi\rangle = -\sum_{E\in\mathcal{E}^h/\mathcal{E}^h_N} \langle \chi, u_D\rangle_E.$$

The Lagrange multiplier associated with the constraint $u\in\left\{v\in H^1(\Omega) : v = u_D \text{ on } \Gamma_D\right\}$ is

$$\kappa = (A\nabla u + up)\cdot\nu_T \qquad \text{on } \partial T/\Gamma_N,\ \forall T\in\mathcal{T}_h. \tag{3.6}$$

3.2. Discrete problem and algebraic formulation

Let P_k be the space of polynomials of degree $\le k$, for integer $k\ge 0$. We define the finite-dimensional spaces $X_h\subset X, M_h\subset M$ as

$$X_h = \prod_{T\in\mathcal{T}_h} P_1(T),$$

$$M_h = \left\{ \chi_h \in \prod_{E\in\mathcal{E}^h/\mathcal{E}^h_N} P_0(E) : \chi_h|_{\partial T_+} + \chi_h|_{\partial T_-} = 0 \text{ on } T_+\cap T_-, \text{ for any adjacent pair } T_\pm\in\mathcal{T}_h \right\}.$$

We seek a pair $(u_h,\kappa_h)\in X_h\times M_h$ such that for all pairs $(v_h,\chi_h)\in X_h\times M_h$

$$\sum_{T\in\mathcal{T}_h}\int_T ((A\nabla u_h + u_h p)\cdot\nabla v_h + \delta u_h v_h)\,\mathrm{d}x - \sum_{E\in\mathcal{E}^h/\mathcal{E}^h_N}\int_E \kappa_h[\![v_h]\!]_E\,\mathrm{d}\gamma$$

$$= \sum_{T\in\mathcal{T}_h}\int_T f v_h\,\mathrm{d}x + \int_{\Gamma_N} g v_h\,\mathrm{d}\gamma, \tag{3.7}$$

$$-\sum_{E\in\mathcal{E}^h/\mathcal{E}^h_N}\int_E \chi_h[\![u_h]\!]_E\,\mathrm{d}\gamma = -\sum_{E\in\mathcal{E}^h/\mathcal{E}^h_N}\int_E \chi_h u_{Dh}\,\mathrm{d}\gamma. \tag{3.8}$$

Note that the problem (3.7)-(3.8) has a unique pair of solutions $(u_h,\kappa_h)\in X_h\times M_h$, and optimal order error estimates of the displacement variable and the Lagrange multiplier in their respective norms can be found in [8].

By defining a norm over the space $\prod_{T\in\mathcal{T}_h} L^2(\partial T/\Gamma_N)$ (cf. (6.28) in [8]) as

$$\|\chi\|_h = \left(\sum_{T\in\mathcal{T}_h} h_T\|\chi\|^2_{L^2(\partial T/\Gamma_N)} \right)^{\frac{1}{2}}, \tag{3.9}$$

providing an explicit bound on the Lagrange multiplier error (cf. (6.30) in [8]).

With $N = \dim(X_h)$, let $X_h = \text{span}\{\phi_1, \dots, \phi_N\}$ and $U = [x_1, \dots, x_N]'$ be the components of

$$u_h = \sum_{i=1}^{N} x_i \phi_i \in X_h.$$

Let E_1, E_2, E_3 be the edges of the triangle T, and let ν_{E_k} denote the unit normal vector of E_k chosen with a global fixed orientation while ν_k denotes the outer unit normal of T along E_k. We define the basis functions ψ_E of M_h, where $E \in \mathcal{E}^h / \mathcal{E}^h_N$ as below:

$$\psi_{E_k}(x) = \sigma_k \qquad \text{for } \ k = 1, 2, 3 \text{ and } \ x \in T,$$

where $\sigma_k = \nu_{E_k} . \nu_k$ is $+1$ if ν_{E_k} points outward and otherwise -1. Globally, let for any edge $E = T_+ \cap T_- \in \mathcal{E}^h_\Omega$

$$\psi_E(x) = \begin{cases} 1 & \text{for } \ x \in T_+, \\ -1 & \text{for } \ x \in T_-, \\ 0 & \text{elsewhere;} \end{cases}$$

and for $E \in \mathcal{E}^h_D$

$$\psi_E(x) = \begin{cases} 1 & \text{for } \ x \in E, \\ 0 & \text{elsewhere.} \end{cases}$$

With $L = \dim(M_h)$, let $M_h = \text{span}\{\psi_l\}_1^L$ with $\psi_l = \psi_{E_l}$, where $l = 1, \dots, L$ is an enumeration of edges

$$\mathcal{E}^h_\Omega \cup \mathcal{E}^h_D = \{E_1, \dots, E_L\}.$$

Let $\Lambda = [x_{N+1}, \dots, x_{N+L}]'$ be the components of κ_h such that

$$\kappa_h = \sum_{l=1}^{L} x_{N+l} \psi_l.$$

The problem (3.7)-(3.8) can be written in a linear system of equations for unknowns U and Λ as: for $j = 1, \dots, N$ and $l = 1, \dots, L$.

$$\sum_{i=1}^{N} x_i \sum_{T \in \mathcal{T}_h} \int_T ((A \nabla \phi_j + \phi_j p) \cdot \nabla \phi_i + \delta \phi_j \phi_i) \, \mathrm{d}x - \sum_{l=1}^{L} x_{N+l} \int_{E_l} \psi_l [\![\phi_j]\!]_{E_l} \, \mathrm{d}\gamma = \sum_{T \in \mathcal{T}_h} \int_T f \phi_j \, \mathrm{d}x + \int_{\Gamma_N} g \phi_j \, \mathrm{d}\gamma, \tag{3.10}$$

$$- \sum_{i=1}^{N} x_i \int_{E_l} \psi_l [\![\phi_i]\!]_{E_l} \, \mathrm{d}\gamma = - \int_{E_l} \psi_l u_{Dh} \, \mathrm{d}\gamma. \tag{3.11}$$

3.3. Local matrices $\mathbb{B}_T$, $\mathbb{D}_T$ $\mathbb{C}_T$, $\mathbb{M}_T$

With local numbers $i \in \{1, 2, 3\}$, let $\{a_i\}_{i=1}^3$ be the set of vertices of an element $T \in \mathcal{T}_h$ and $\{\lambda_i(x)\}_{i=1}^3$ be the set of barycentric coordinates of a point $x \in T$ (cf. [17]). Let $\{\phi_i^T(x)\}_{i=1}^3$ be the set of basis functions of $P_1(T)$. It should be noted that the basis functions of X_h are not globally continuous over Ω but locally continuous, which means that for any triangle $T \in \mathcal{T}_h$ with global node numbers i, j, k, $\phi_i = \lambda_1^T; \phi_j = \lambda_2^T; \phi_k = \lambda_3^T$.

Definition 3.1. The local matrices are defined for $i, j, l = 1, 2, 3$ as follows:

$$(\mathbb{B}_T)_{ji} = \int_T A \nabla \phi_j \cdot \nabla \phi_i \ \mathrm{d}x, \qquad (\mathbb{D}_T)_{ji} = \int_T (\phi_j p) \cdot \nabla \phi_i \, \mathrm{d}x,$$

$$(\mathbb{C}_T)_{lj} = \int_{E_l} \psi_l [\![\phi_j]\!]_{E_l} \, \mathrm{d}\gamma, \qquad (\mathbb{M}_T)_{ji} = \int_T \delta \phi_j \phi_i \, \mathrm{d}x.$$

The elaborate computation provides forms of all local matrices:

$$\mathbb{B}_T = |T| \begin{pmatrix} (A\nabla\lambda_1^T)\cdot\nabla\lambda_1^T & (A\nabla\lambda_1^T)\cdot\nabla\lambda_2^T & (A\nabla\lambda_1^T)\cdot\nabla\lambda_3^T \\ (A\nabla\lambda_2^T)\cdot\nabla\lambda_1^T & (A\nabla\lambda_2^T)\cdot\nabla\lambda_2^T & (A\nabla\lambda_2^T)\cdot\nabla\lambda_3^T \\ (A\nabla\lambda_3^T)\cdot\nabla\lambda_1^T & (A\nabla\lambda_3^T)\cdot\nabla\lambda_2^T & (A\nabla\lambda_3^T)\cdot\nabla\lambda_3^T \end{pmatrix}, \tag{3.12}$$

$$\mathbb{D}_T = \frac{|T|}{3} \begin{pmatrix} p\cdot\nabla\lambda_1^T & p\cdot\nabla\lambda_1^T & p\cdot\nabla\lambda_1^T \\ p\cdot\nabla\lambda_2^T & p\cdot\nabla\lambda_2^T & p\cdot\nabla\lambda_2^T \\ p\cdot\nabla\lambda_3^T & p\cdot\nabla\lambda_3^T & p\cdot\nabla\lambda_3^T \end{pmatrix}, \tag{3.13}$$

$$\mathbb{M}_T = \delta \begin{pmatrix} \int_T \lambda_1^T\lambda_1^T\,\mathrm{d}x & \int_T \lambda_1^T\lambda_2^T\,\mathrm{d}x & \int_T \lambda_1^T\lambda_3^T\,\mathrm{d}x \\ \int_T \lambda_2^T\lambda_1^T\,\mathrm{d}x & \int_T \lambda_2^T\lambda_2^T\,\mathrm{d}x & \int_T \lambda_2^T\lambda_3^T\,\mathrm{d}x \\ \int_T \lambda_3^T\lambda_1^T\,\mathrm{d}x & \int_T \lambda_3^T\lambda_2^T\,\mathrm{d}x & \int_T \lambda_3^T\lambda_3^T\,\mathrm{d}x \end{pmatrix} = \delta|T| \begin{pmatrix} \frac{1}{6} & \frac{1}{12} & \frac{1}{12} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{12} & \frac{1}{12} & \frac{1}{6} \end{pmatrix}, \tag{3.14}$$

$$\mathbb{C}_T = \begin{pmatrix} \int_{E_1} \psi_1[\![\lambda_1^T]\!]_{E_1}\,\mathrm{d}\gamma & \int_{E_1} \psi_1[\![\lambda_2^T]\!]_{E_1}\,\mathrm{d}\gamma & \int_{E_1} \psi_1[\![\lambda_3^T]\!]_{E_1}\,\mathrm{d}\gamma \\ \int_{E_2} \psi_2[\![\lambda_1^T]\!]_{E_2}\,\mathrm{d}\gamma & \int_{E_2} \psi_2[\![\lambda_2^T]\!]_{E_2}\,\mathrm{d}\gamma & \int_{E_2} \psi_2[\![\lambda_3^T]\!]_{E_2}\,\mathrm{d}\gamma \\ \int_{E_3} \psi_3[\![\lambda_1^T]\!]_{E_3}\,\mathrm{d}\gamma & \int_{E_3} \psi_3[\![\lambda_2^T]\!]_{E_3}\,\mathrm{d}\gamma & \int_{E_3} \psi_3[\![\lambda_3^T]\!]_{E_3}\,\mathrm{d}\gamma \end{pmatrix} = \begin{pmatrix} |E_1|\sigma_1 R_1 \\ |E_2|\sigma_2 R_2 \\ |E_3|\sigma_3 R_3 \end{pmatrix}, \tag{3.15}$$

where $R_1 = \left[0, \frac{1}{2}, \frac{1}{2}\right]$, $R_2 = \left[\frac{1}{2}, 0, \frac{1}{2}\right]$, $R_3 = \left[\frac{1}{2}, \frac{1}{2}, 0\right]$.

3.4. Right-hand side and boundary conditions

The computation of the right-hand side in (3.10)-(3.11) includes the numerical integration over elements and edges of the functions f, g, and u_D. The right-hand side $F = \{F_j\}_{j=1}^N$ in (3.10) is defined as $F_j = b_j + LN_j$, where

$$b_j = \sum_{T\in\mathcal{T}_h} \int_T f\phi_j\,\mathrm{d}x, \qquad LN_j = \sum_{E\in\mathcal{E}_N^h} \int_E g\phi_j\,\mathrm{d}\gamma.$$

Both parts are approximated by

$$b_j \approx \sum_{T\in\mathcal{T}_h} \frac{|T|}{3} \sum_{i=1}^{3} f(m_{E_i})\,\phi_j(m_{E_i}), \tag{3.16}$$

$$LN_j \approx \sum_{E\in\mathcal{E}_N^h} |E| g(m_E)\,\phi_j(m_E), \tag{3.17}$$

where $m_E \in \mathcal{N}_m(\Gamma_N)$. The right hand side $b_D = \{b_{D_l}\}_{l=1}^L$ in (3.11) is defined and approximated as

$$\begin{aligned} b_{D_l} = -\int_{E_l} \psi_l u_{Dh}\,\mathrm{d}\gamma &\approx -|E_l|\,\sigma_l u_{Dh}(m_{E_l}), && \text{where } m_{E_l} \in \mathcal{N}_m(\Gamma_D) \\ &= -|E_l|\,u_{Dh}(m_{E_l}), && \text{since } \sigma_l = 1 \text{ for } l = 1,\dots,L. \end{aligned} \tag{3.18}$$

3.5. Assembly of the global matrices

In vector-matrix form, (3.10)-(3.11) can be written as

$$\mathrm{M}_{\mathrm{vec}}\mathrm{W}_{\mathrm{vec}} = \mathrm{F}_{\mathrm{vec}}\ , \tag{3.19}$$

```
function [B,D,M,b]=StiffMassConv_PH(coordinates,elements,params)
A=params.A;p=params.p;delta=params.delta;
nE=size(elements,1);  % number of elements
%% Vectorization of B matrix
P1=coordinates(elements(:,1),:); % collection of the first nodes of all triangles
P2=coordinates(elements(:,2),:); % collection of the second nodes of all triangles
P3=coordinates(elements(:,3),:); % collection of the third nodes of all triangles
e1=P3-P2; e2=P3-P1; e3=P2-P1; % represents the edges of an element within a mesh
mp23=(P2+P3)/2;  mp13=(P1+P3)/2;   mp12=(P1+P2)/2; % midpoints of the edges
area=(e2(:,2).*e3(:,1)-e2(:,1).*e3(:,2))/2; % Area
GI=reshape([1:3*nE; 1:3*nE; 1:3*nE],9,nE);  % 3*nE =Number of nodes
I3=reshape(GI([1 4 7 2 5 8 3 6 9],:),9*nE,1); J3=reshape(GI,9*nE,1);
G1=[-e1(:,2),e1(:,1)]./(2*area);
G2=[e2(:,2),-e2(:,1)]./(2*area);
G3=[-e3(:,2),e3(:,1)]./(2*area);
BT=[[dot(G1*A,G1,2),dot(G1*A,G2,2),dot(G1*A,G3,2),dot(G2*A,G1,2),dot(G2*A,G2,2),...
    dot(G2*A,G3,2),dot(G3*A,G1,2),dot(G3*A,G2,2),dot(G3*A,G3,2)].*area]';
B=sparse(I3,J3,BT,3*nE,3*nE);
%% Vectorization of mass matrix (M)
MT=reshape(delta*(area/12)'.*[2;1;1;1;2;1;1;1;2],9*nE,1);
M=sparse(I3,J3,MT,3*nE,3*nE);
%% Vectorization of D matrix
h1=(-p(1)*e1(:,2)+p(2)*e1(:,1))'/6;
h2=(p(1)*e2(:,2)-p(2)*e2(:,1))'/6;
h3=(-p(1)*e3(:,2)+p(2)*e3(:,1))'/6;
DT=[h1;h2;h3;h1;h2;h3;h1;h2;h3];
D=sparse(I3,J3,DT,3*nE,3*nE);
%% Vectorization of load vector b
f1=f(mp13,params)/2 +f(mp12,params)/2;
f2=f(mp23,params)/2+f(mp12,params)/2;
f3=f(mp13,params)/2+f(mp23,params)/2;
fT=(area.*[f1 f2 f3])/3;
b=accumarray(reshape(1:3*nE,3*nE,1),reshape(fT',3*nE,1),[3*nE 1]);
end
```

Listing 5: Vectorization of the assembly of stiffness matrix $\mathbb{B}$, convection matrix $\mathbb{D}$, mass matrix $\mathbb{M}$ and load vector b.

where global matrices and vectors read

$$\mathrm{M}_{\mathrm{vec}} = \begin{pmatrix} \mathbb{B}+\mathbb{D}+\mathbb{M} & -\mathbb{C}' \\ -\mathbb{C} & 0 \end{pmatrix}_{(N+L)\times(N+L)}, \quad \mathrm{W}_{\mathrm{vec}} = \begin{pmatrix} U \\ \Lambda \end{pmatrix}_{(N+L)\times 1}, \quad \mathrm{F}_{\mathrm{vec}} = \begin{pmatrix} F \\ b_D \end{pmatrix}_{(N+L)\times 1}$$

with

$$\mathbb{B} = \left(\mathbb{B}_{ij}\right)_{N\times N}, \quad \mathbb{B}_{ij} = \sum_{T\in\mathcal{T}_h} \int_T (A\nabla\phi_i)\cdot\nabla\phi_j \ \mathrm{d}x,$$

$$\mathbb{D} = \left(\mathbb{D}_{ij}\right)_{N\times N}, \quad \mathbb{D}_{ij} = \sum_{T\in\mathcal{T}_h} \int_T (\phi_i p)\cdot\nabla\phi_j \ \mathrm{d}x,$$

$$\mathbb{M} = \left(\mathbb{M}_{ij}\right)_{N\times N}, \quad \mathbb{M}_{ij} = \sum_{T\in\mathcal{T}_h} \int_T \phi_i\cdot\phi_j \ \mathrm{d}x,$$

$$\mathbb{C} = (\mathbb{C}_{N+l,i})_{L\times N}, \quad \mathbb{C}_{N+l,i} = \int_{E_l} \psi_l [\![\phi_i]\!]_{E_l} \mathrm{d}\gamma,$$

$$F = (F_j)_{N\times 1}, \quad b_D = (b_{D_l})_{L\times 1}.$$

3.6. Implementation

3.6.1. MATLAB code of vectorized **StiffMassConv_PH.m**

Here we explain the vectorized code **StiffMassConv_PH.m**, where we compute the matrices $\mathbb{B}$, $\mathbb{D}$, $\mathbb{M} \in \mathbb{R}^{N\times N}$, and $b \in \mathbb{R}^{N\times 1}$. The vectorization of stiffness matrix $\mathbb{B}$ and load vector b are similar to the MATLAB code Listing 3 in [5] with an adequate amount of modifications.

- Line 11-12: We create a $9\times$ nE matrix called `GI` by reshaping the matrix $[1:3\times\mathrm{nE}; 1:3\times\mathrm{nE}; 1:3\times\mathrm{nE}]$ such that each column of `GI` (cf. Fig. 5) corresponds to the column index of local stiffness matrices for elements in the global assembly. `I3` and `J3` (cf. Fig. 6) contain index pairs for the global stiffness matrix, constructed using `GI`. Note that, `I3` is based on the modified `GI` with specific row indices [1 4 7 2 5 8 3 6 9], while `J3` is based on the original `GI`.

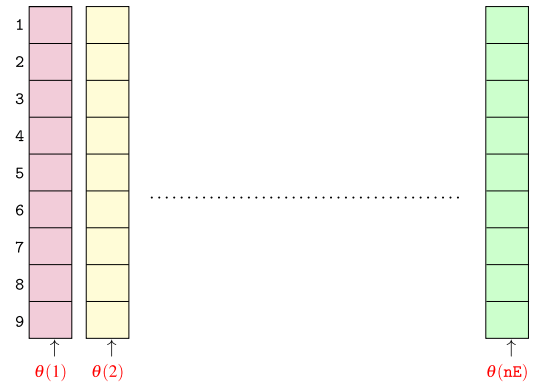

Fig. 5. Geometrical representation of the columns of the GI matrix.

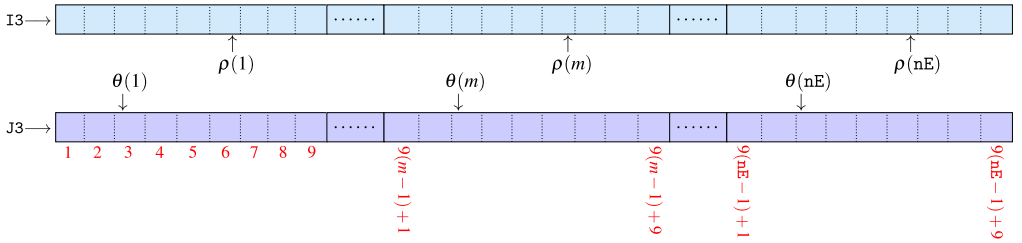

Fig. 6. Geometrical representation of I3 and J3 column vectors.

Define, the row and column index pair of entries in the stiffness matrix corresponding to the m^{th} element as

$$\rho(m) = \begin{bmatrix} 3(m-1)+1 \\ 3(m-1)+2 \\ 3(m-1)+3 \\ 3(m-1)+1 \\ 3(m-1)+2 \\ 3(m-1)+3 \\ 3(m-1)+1 \\ 3(m-1)+2 \\ 3(m-1)+3 \end{bmatrix} \text{ and } \theta(m) = \begin{bmatrix} 3(m-1)+1 \\ 3(m-1)+1 \\ 3(m-1)+1 \\ 3(m-1)+2 \\ 3(m-1)+2 \\ 3(m-1)+2 \\ 3(m-1)+3 \\ 3(m-1)+3 \\ 3(m-1)+3 \end{bmatrix}, \quad \text{where } m = 1, 2, \ldots, \mathtt{nE}.$$

Remark 2. The index pair vectors for conforming (cf. [5]) and non-conforming FEM differ in vectorization because of the distinct element-wise assembly of the local stiffness matrices. For primal hybrid FEM, we generate I3 and J3 index pair vectors that are compatible with non-conforming P1-FEM.

- Line 13-18: The gradients of the basis functions G1, G2, and G3 over element T are

$$\mathtt{G1}|_T = \frac{1}{2|T|}\begin{bmatrix} y_2 - y_3 & x_3 - x_2 \end{bmatrix}, \quad \mathtt{G2}|_T = \frac{1}{2|T|}\begin{bmatrix} y_3 - y_1 & x_1 - x_3 \end{bmatrix}, \quad \mathtt{G3}|_T = \frac{1}{2|T|}\begin{bmatrix} y_1 - y_2 & x_2 - x_1 \end{bmatrix}.$$

- Line 19-21: The element-wise assembly of $\mathbb{M}$ uses nine updates of the mass matrix per element T. That is, the vector MT has length $9\times$ nE.
- Lines 22-27: $\mathtt{h1}|_T$, $\mathtt{h2}|_T$, and $\mathtt{h3}|_T$ are entries of the matrix $\mathbb{D}_T$ (3.13),

$$\mathtt{h1}|_T = \frac{1}{6}\, p \cdot \begin{bmatrix} -(y_3 - y_2) \\ x_3 - x_2 \end{bmatrix}, \quad \mathtt{h2}|_T = \frac{1}{6}\, p \cdot \begin{bmatrix} y_3 - y_1 \\ -(x_3 - x_1) \end{bmatrix}, \quad \mathtt{h3}|_T = \frac{1}{6}\, p \cdot \begin{bmatrix} -(y_2 - y_1) \\ x_2 - x_1 \end{bmatrix}.$$

  We assemble the convection matrix $\mathbb{D}$ by using **sparse** command in MATLAB with I3(row index) and J3(column index).
- Line 28-33: We evaluate the volume forces $f(m_{E_i})$, $i = 1, 2, 3$ for all triangle $T \in \mathcal{T}_h$, then (3.16) and assemble the load vector b using **accumarray**.

3.6.2. *MATLAB code of vectorized* **Lambda_PH.m**

```
1  function [C,e11,v,R11]= Lambda_PH(coordinates,ed2el,intE,Dbed,Redges,led,ledTN)
2  nEd=size(ed2el,1); % number of edges
3  num_intE=size(intE,1); % number of interior edges
4  nDb=size(Dbed,1); % number of Dirichlet boundary edges
5  edges=ed2el(1:nEd,1:2);   % all edges
6  %% Vectorization of matrix C
7  v=vecnorm((coordinates(edges(:,1),:)-coordinates(edges(:,2),:))')';   % length of edges
8  e11=ed2el(:,3); e12=nonzeros(ed2el(:,4));
9  i=find(led==1); j=find(led==2); k=find(led==3);
10 p=find(ledTN(intE)==1); q=find(ledTN(intE)==2); r=find(ledTN(intE)==3);
11 R1=[0, 1/2, 1/2]; R2= [1/2, 0,  1/2]; R3= [1/2, 1/2, 0];
12 R11(i,:)=repmat(R1,size(i));
13 R11(j,:)=repmat(R2,size(j));
14 R11(k,:)=repmat(R3,size(k));
15 R22(p,:)=repmat(R1,size(p));
16 R22(q,:)=repmat(R2,size(q));
17 R22(r,:)=repmat(R3,size(r));
18 I1=reshape([Dbed' intE';Dbed' intE';Dbed' intE'],3*(num_intE+nDb),1);
19 J1=reshape(((e11([Dbed' intE'])-ones(num_intE+nDb,1))*3 +[1 2 3])'...
20         ,3*(num_intE+nDb),1);
21 S1=reshape(( v([Dbed' intE']).* R11([Dbed' intE'],:))',3*(num_intE+nDb),1);
22 I2=reshape([intE';intE';intE'],3*num_intE,1);
23 J2=reshape(((e12(:)-ones(num_intE,1))*3 +[1 2 3])',3*num_intE,1);
24 S2=reshape(( (-1)*v(intE).*(R22))',3*num_intE,1);
25 I=[I1;I2]; J=[J1;J2]; S=[S1;S2]; % appending vectors
26 C=sparse(I, J, S);
27 C=C(Redges, :);   % eliminate extra rows
28 end
```

Listing 6: Vectorized and efficient MATLAB implementation of the assembly of matrix $\mathbb{C}$.

- Line 1: The function **Lambda_PH.m** takes the `coordinates` data, `ed2el`, `intE`, Dirichlet boundary edges(`Dbed`), `Redges`($\mathcal{E}_N^h$), `led`, and `ledTN`. It computes the matrix $\mathbb{C} \in \mathbb{R}^{L\times N}$ in (3.19) and other required data `e11`, `v`, `R11` for **Main.m** described in the next subsection.
- Line 7-8: The length of all edges is computed by using the MATLAB function **vecnorm** and stored in a vector `v`. Vector `e11` = `ed2el(:,3)`, which contains the index of all T_+ elements, and vector `e12` = **nonzeros**(`ed2el(:,4)`), which contains the index of all T_- elements.
- Line 9-10: For each triangular element $T_+ \in \mathcal{T}_h$, the vectors `i`, `j`, `k` store the indices of the first edge (E_1), second edge (E_2), third edge (E_3), respectively. Similarly, the vectors `p`, `q`, `r` store the indices of E_1, E_2, E_3 for T_- elements.
- Line 11-17: We construct the matrices `R11` $\in \mathbb{R}^{L\times 3}$ and `R22` $\in \mathbb{R}^{N_\Omega\times 3}$ based on the values of the `led` and `ledTN` vectors, as well as the R_1, R_2, and R_3 vectors (Definition 3.1). The rows of `R11` corresponding to E_1, E_2, and E_3 of each $T_+ \in \mathcal{T}_h$ are populated with the vectors R_1, R_2, and R_3, respectively. This is accomplished by using the **repmat** function to replicate each vector along the rows of `R11` at the indices stored in `i`, `j`, and `k`, respectively. Similarly, for each $T_- \in \mathcal{T}_h$, we build the rows of `R22` are R_1, R_2, and R_3 vectors with respect to the indices stored in `p`, `q`, and `r`.
- Line 18-24: We calculate the row and column indexing vectors `I1` and `J1` $\in \mathbb{R}^{3L\times 1}$ (as shown in Fig. 7) for the matrix $\mathbb{C}$ based on the T_+ elements. Similarly, we obtain the row and column indexing vectors `I2` and `J2` $\in \mathbb{R}^{3N_\Omega\times 1}$ (as shown in Fig. 8) for the matrix $\mathbb{C}$ based on the T_- elements. The entries of matrix $\mathbb{C}$ corresponding to T_+ and T_- elements are represented by `S1` $\in \mathbb{R}^{3L\times 1}$ (as shown in Fig. 7) and `S2` $\in \mathbb{R}^{3N_\Omega\times 1}$ (as shown in Fig. 8), respectively, and are arranged in column vectors:

$$\texttt{S1}_{|_{T_+}} = \begin{pmatrix} \texttt{v}(i)\,\texttt{R11}(i,:) \\ \texttt{v}(j)\,\texttt{R11}(j,:) \\ \texttt{v}(k)\,\texttt{R11}(k,:) \end{pmatrix}, \qquad \text{where } i,\ j,\ \text{and } k \text{ are the global edge number of } T_+,$$

$$\text{and}\quad \texttt{S2}_{|_{T_-}} = \begin{pmatrix} -\texttt{v}(p)\,\texttt{R22}(p,:) \\ -\texttt{v}(q)\,\texttt{R22}(q,:) \\ -\texttt{v}(r)\,\texttt{R22}(r,:) \end{pmatrix}, \qquad \text{where } p,\ q,\ \text{and } r \text{ are the global edge number of } T_-.$$

Let $\alpha_1, \alpha_2, \ldots, \alpha_{N_D}$ be the global numbers of $\mathcal{E}_D^h$ for the mesh $\mathcal{T}_h$. Let $\beta_1, \beta_2, \ldots, \beta_{N_\Omega}$ be the global numbers of $\mathcal{E}_\Omega^h$ for the mesh $\mathcal{T}_h$. Define,

$$\gamma_m = (\texttt{e11}(m) - 1) * 3 + [1\ 2\ 3], \qquad \text{where } m = 1, 2, \ldots, \texttt{nEd},$$

$$\text{and}\quad \zeta_l = (\texttt{e12}(l) - 1) * 3 + [1\ 2\ 3], \qquad \text{where } l = 1, 2, \ldots, N_\Omega.$$

- Line 25-27: We are appending the vectors `I=[I1;I2]`, `J=[J1;J2]`, `S=[S1;S2]`. We assemble the matrix $\mathbb{C}$ by using **sparse** with `I`, `J`, and `S`, but it has extra zero rows with respect to the $\mathcal{E}_N^h$. We eliminate the extra rows from matrix $\mathbb{C}$ in line 27.

3.6.3. *MATLAB code of* **Main.m**

```
load('coordinates.dat');   load('elements.dat');
load('Dirichlet.dat');     load('Neumann.dat');
[n2el,n2ed,ed2el,intE,extE,led,ledTN]=edge_vec(elements,coordinates);
params.A=[1 0; 0 1]; params.p=[1,1]; params.delta=1; % problem parameters
nt=4;
 for level=1:nt
    fprintf('Level = %d\n',level);
    tic
    [coordinates,elements,Dirichlet,Neumann]=redrefine_vec(coordinates,elements,n2ed,ed2el,Dirichlet,Neumann);
    fprintf('run-time (in sec) for the function redrefine_vec.m  is = %3.3f\n',toc);
    tic
    [n2el,n2ed,ed2el,intE,extE,led,ledTN]=edge_vec(elements,coordinates);
    fprintf('run-time (in sec) for the function edge_vec.m  is = %3.3f\n',toc);
    h(level)=sqrt(det([1 1 1;coordinates(elements(1,:),:)'])*2); % space parameter
    grad4e = getGrad4e(coordinates,elements);
    nE=size(elements,1);  % Number of elements
    nEd=size(ed2el,1); % Number of Edges
    fprintf('number of elements = %d, number of edges = %d \n',nE,nEd);
    Dbed=n2ed(sub2ind(size(n2ed),Dirichlet(:,1),Dirichlet(:,2))); %Dirichlet boundary edges
    if (~isempty(Neumann))
        Nbed=n2ed(sub2ind(size(n2ed),Neumann(:,1),Neumann(:,2))); %Neumann boundary edges
    else
        Nbed=[];
    end
    Redges=setdiff(1:nEd,Nbed); % removing Neumann edges from all edges
    L=length(Redges);
    ndf=3*nE;            %degree of freedom
    tic
    [B,D,M,b]=StiffMassConv_PH(coordinates,elements,params);
    fprintf('run-time (in sec) for the function StiffMassConv_PH.m  is = %3.3f\n',toc);
    tic
    [C,e11,v,R11]= Lambda_PH(coordinates,ed2el,intE,Dbed,Redges,led,ledTN);
    fprintf('run-time (in sec) for the function Lambda_PH.m  is = %3.3f\n',toc);
    %% Vectorization of Neumann BC
    LN=sparse(ndf,1);
    if (~isempty(Neumann))
        nNb=size(Nbed,1); % number of Neumann boundary edges
        NP1=coordinates(Neumann(:,1),:); NP2=coordinates(Neumann(:,2),:);
        Nmd=(NP1+NP2)/2; % mid-point of Neumann edges
        Nnorm=[NP2(:,2)-NP1(:,2) NP1(:,1)-NP2(:,1)]./vecnorm((NP1-NP2)')'; %normal to Neumann edge
        JN=reshape(((e11(Nbed)-ones(nNb,1))*3+[1 2 3])',3*nNb,1);
        S=reshape(((diag(v(Nbed).*g(Nmd,params)*Nnorm')).*R11(Nbed,:))',3*nNb,1);
        LN= accumarray (JN, S ,[ndf 1]);
    end
    %% Dirichlet boundary condition
    uhb1=sparse(L,1);
    [ed,~]=find(Redges==Dbed);  % extract the linear indices of Dirichlet edge number
    me=(coordinates(Dirichlet(:,2),:)+coordinates(Dirichlet(:,1),:))/2; % mid-point of Dirichlet edges
    uhb1(ed)=-v(Dbed).*uD(me);
%%%%%%% Solving saddle point problem %%%%%%
    tic
    Mvec=[ B + D + M         -C'
          -C          sparse(L,L)];
    Fvec=[b+LN ; uhb1];
    Wvec=Mvec\Fvec;
    uh=Wvec(1:ndf);
    Kh=Wvec(1+ndf:length(Wvec));
    u=ue(coordinates(elements',:)); % exact solution
    K=exactLambda(coordinates,ed2el(Redges,1:2),params);
    fprintf('run-time (in sec) for the linear solver is = %3.3f\n',toc);
    tic
    [L2e(level),Xe(level),herr(level)]=PHerror(uh,Kh,ed2el(Redges,1:2),grad4e,elements,coordinates,h(level),v(Redges),
     params);
    fprintf('run-time (in sec) for the function PHerror.m is = %3.3f\n',toc);
    fprintf('\n');
 end
%% order of convergence
 for j=1:level-1
    ocX(j)=log(Xe(j)/Xe(j+1))/log(h(j)/h(j+1));
    ocL2(j)=log(L2e(j)/L2e(j+1))/log(h(j)/h(j+1));
    och(j)=log(herr(j)/herr(j+1))/log(h(j)/h(j+1));
 end
fprintf('\n order of convergences w.r.t h \n in   X-norm   L2-norm   h-norm \n');
disp([ocX',ocL2',och']);
```

Listing 7: **Main.m**.

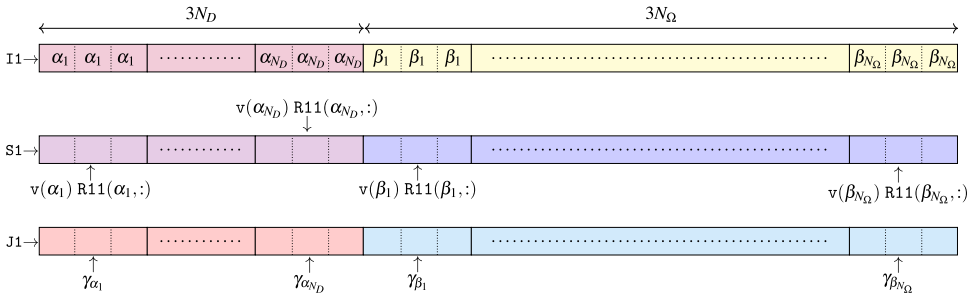

Fig. 7. Geometrical representation of `I1`, `J1`, and `S1` vectors.

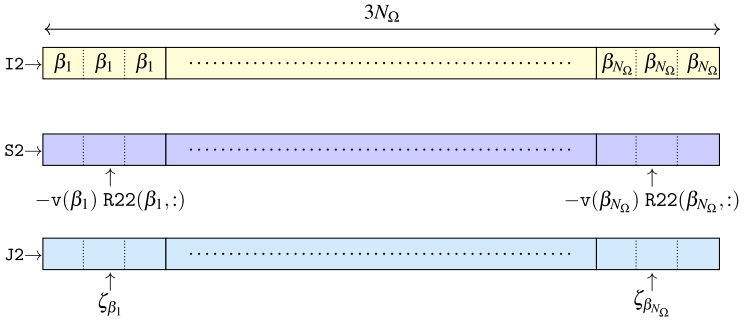

Fig. 8. Geometrical representation of `I2`, `J2`, and `S2` vectors.

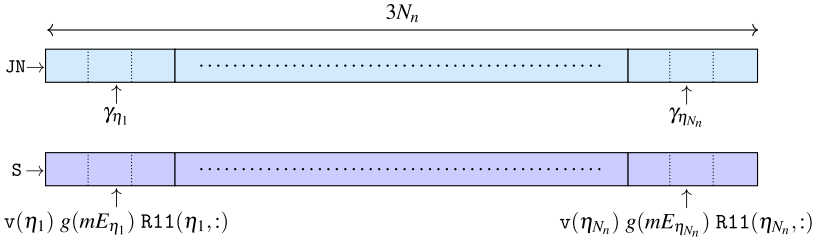

Fig. 9. Geometrical representation of `JN`, and `S` vectors.

- The computation of $LN \in \mathbb{R}^{N\times 1}$ in MATLAB reads lines 35-44, incorporating the Neumann boundary conditions. The Neumann boundary condition in (3.17) at every mid-point $m_E \in \mathcal{N}_m(\Gamma_N)$ is evaluated. `JN` and `S` $\in \mathbb{R}^{3N_n\times 1}$ (Fig. 9) indicate the row indexing vector and entries for the matrix LN. Let $\eta_1, \eta_2, \ldots, \eta_{N_n}$ be the Neumann boundary edge global numbers for the mesh $\mathcal{T}_h$. The assembly of the matrix LN is done in line 43 by using the **accumarray** with `JN` and `S`.
- Line 46-49: In taking care the Dirichlet boundary condition (3.18), the integral is approximated by the mid-point rule.
- Line 52-54: Assembly of M_{vec} (see, (3.19)) in which zero $(L\times L)$ matrix defined as a **sparse** matrix, which helps to save run-time and storage space for zero-valued entries.
- Line 59: The κ (3.6) is calculated using the **exactLambda.m** function.
- Line 62: The errors $u - u_h$ and $\kappa - \kappa_h$ are computed using the **PHerror.m** function.

Example 2. Let $A = I_{2\times 2}$, $p = [1, 1]$, $\delta = 1$. We take the exact solution of (3.1)-(3.3) as

$$u(x_1, x_2) = (x_1 - x_1^2)(x_2 - x_2^2) \qquad \text{in } \Omega = (0,1)\times(0,1) \tag{3.20}$$

and corresponding load function $f(x_1, x_2)$ in the form

$$f(x_1, x_2) = (1 + 2x_1)(x_2 - x_2^2) + (x_1 - x_1^2)(1 + 2x_2) + u(x_1, x_2) \qquad \text{in } \Omega. \tag{3.21}$$

The Dirichlet and Neumann boundary conditions (cf. Fig. 10) read

$$u_D(x_1, x_2) = 0 \qquad \text{on } \Gamma_D,$$
$$g(x_1, x_2) = [(1 - x_1 - x_1^2)(x_2 - x_2^2),\ (x_1 - x_1^2)(1 - x_2 - x_2^2)] \cdot \nu \qquad \text{on } \Gamma_N,$$

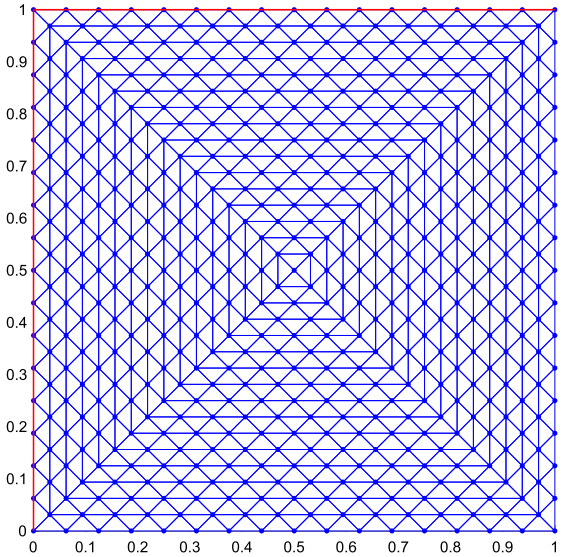

Fig. 10. Example of (level 4, $h = 1/16$) red-refined mesh used in both elliptic and parabolic model computations. Left and top red-colored sides denote Γ_N and the remaining right and bottom sides Γ_D.

Table 3
Run-time of the **StiffMassConv_PH.m** and **Lambda_PH.m** for second order elliptic model problem.

Level	nE	nEd	run-time (in sec)	
			StiffMassConv_PH.m	**Lambda_PH.m**
4	1024	1568	0.002	0.002
5	4096	6208	0.004	0.005
6	16384	24704	0.017	0.030
7	65536	98560	0.063	0.084
8	262144	393728	0.290	0.395
9	1048576	1573888	1.092	1.408
10	4194304	6293504	4.215	6.103

Table 4
Order of convergence in L^2, X and h-norms for second order elliptic model problem.

Level	$\|u-u_h\|_X$	Order of convergence	$\|u-u_h\|_{L^2}$	Order of convergence	$\|\kappa-\kappa_h\|_h$	Order of convergence
1	0.0841		0.0111		0.1304	
2	0.0414	1.0226	0.0026	2.0987	0.0557	1.2268
3	0.0205	1.0087	0.0006	2.0282	0.0264	1.0783
4	0.0103	1.0027	0.0002	2.0072	0.0130	1.0213
5	0.0051	1.0007	3.94e-05	2.0018	0.0065	1.0054
6	0.0026	1.0002	9.85e-06	2.0004	0.0032	1.0014
7	0.0013	1.0000	2.46e-06	2.0001	0.0016	1.0003
8	0.0006	1.0000	6.15e-07	2.0000	0.0008	1.0001
9	0.0003	1.0000	1.53e-07	2.0000	0.0004	1.0000

where ν denotes the outer normal vector.

Table 3 displays the run-time (in seconds) for vectorized implementations of functions **StiffMassConv_PH.m** and **Lambda_PH.m**, which vary with the number of elements in the mesh.

In Table 4, the errors $u - u_h$ in the L^2 and X-norm, and the error $\kappa - \kappa_h$ in the h-norm (cf. (3.9)) are presented. In addition, we present the order of convergence in the respective norms.

Remark 3. We notice that the run-time of Table 3 is comparable to Table 2. Thus, the setup of FEM matrices is only a few factors more expensive than mesh refinement and edge ordering. The complete FEM computation is performed by the script **Main.m** given in Listing 7. For the levels 8 and 9, size of the M_{vec} (3.19) is 1179648×1179648 and 4718592×4718592. The run-time of **Main.m** is significantly higher due to the costs of the default MATLAB linear solver (6.879 and 37.247 seconds for levels 8 and 9).

4. Implementation of second order parabolic model problem

4.1. Parabolic problem and algebraic formulation

Consider a second-order parabolic model problem with mixed boundary and initial conditions

$$u_t(x,t) - \nabla\cdot(A\nabla u(x,t) + u(x,t)p) + \delta u(x,t) = f(x,t) \qquad \text{in } \Omega\times(0,t_{\tilde{N}}], \tag{4.1}$$

$$u(x,t) = u_D(x,t) \qquad \text{on } \Gamma_D\times(0,t_{\tilde{N}}], \tag{4.2}$$

$$(A\nabla u(x,t) + u(x,t)p)\cdot\nu = g(x,t) \qquad \text{on } \Gamma_N\times(0,t_{\tilde{N}}] \tag{4.3}$$

$$u(x,0) = u_0(x) \qquad \text{in } \Omega. \tag{4.4}$$

Here, f, u_D, g and u_0 are suitable smooth functions, A is a symmetric and positive definite matrix with constant entries, p is a constant vector, and δ is a non-negative constant.

Primal hybrid formulation of (4.1)-(4.4) is to seek a pair of solutions $(u,\kappa) : [0,t_{\tilde{N}}] \to X\times M$ such that

$$(u_t, v) + a(u,v) + b(v,\kappa) = (f,v) + \langle g, v\rangle_{\Gamma_N}, \qquad \forall v\in X, \tag{4.5}$$

$$b(u,\chi) = \langle u_D, \chi\rangle, \qquad \forall \chi\in M, \tag{4.6}$$

$$u(0) = u_0, \tag{4.7}$$

Let the time interval $[0,t_{\tilde{N}}]$ be partitioned into equally spaced points

$$0 = t_0 < t_1 < \cdots < t_{\tilde{N}}$$

with time step parameter k such that $\tilde{N} = t_{\tilde{N}}/k$ and $t_n = nk$ for $n\in\{1,2,\dots,\tilde{N}\}$. For $\varphi\in C[0,t_{\tilde{N}}]$, we set the backward difference operator ∂ as

$$\partial\varphi^n = (\varphi^n - \varphi^{n-1})/k$$

and for $t_{n-1/2} = (n-1/2)k$ the midpoint value as

$$\varphi^{n-1/2} = (\varphi^n + \varphi^{n-1})/2.$$

We employ two methods for the fully discrete set-up: (i) the backward Euler explicit method and (ii) the Crank-Nicolson implicit method in the time direction.

- By employing the backward Euler explicit method in the time direction, we seek a solution pair $(u_h^n,\kappa_h^n)\in X_h\times M_h$ satisfying

$$\sum_{T\in\mathcal{T}_h}\int_T \partial u_h^n v_h\,\mathrm{d}x + \sum_{T\in\mathcal{T}_h}\int_T A\nabla u_h^n\cdot\nabla v_h\,\mathrm{d}x + \sum_{T\in\mathcal{T}_h}\int_T (u_h^n p)\cdot\nabla v_h\,\mathrm{d}x$$
$$+ \sum_{T\in\mathcal{T}_h}\int_T \delta u_h^n v_h\,\mathrm{d}x - \sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \kappa_h^n[\![v_h]\!]_E\,\mathrm{d}\gamma = F^{t_{n-1}}(v_h),$$
$$- \sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \chi_h[\![u_h^n]\!]_E\,\mathrm{d}\gamma = -\sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \chi_h u_{Dh}^{n-1}\,\mathrm{d}\gamma,$$

  for any testing pair $(v_h,\chi_h)\in X_h\times M_h$. Here,

$$F^{t_{n-1}}(v_h) = \sum_{T\in\mathcal{T}_h}\int_T f(t_{n-1})v_h\,\mathrm{d}x + \sum_{E\in\mathcal{E}_N^h}\int_E g(t_{n-1})v_h\,\mathrm{d}\gamma.$$

- By employing the Crank-Nicolson implicit scheme in the time direction, we seek a solution pair $(u_h^n,\kappa_h^n)\in X_h\times M_h$ satisfying

$$\sum_{T\in\mathcal{T}_h}\int_T \partial u_h^n v_h\,\mathrm{d}x + \sum_{T\in\mathcal{T}_h}\int_T A\nabla u_h^{n-1/2}\cdot\nabla v_h\,\mathrm{d}x + \sum_{T\in\mathcal{T}_h}\int_T (u_h^{n-1/2} p)\cdot\nabla v_h\,\mathrm{d}x$$
$$+ \sum_{T\in\mathcal{T}_h}\int_T \delta u_h^{n-1/2} v_h\,\mathrm{d}x - \sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \kappa_h^{n-1/2}[\![v_h]\!]_E\,\mathrm{d}\gamma = F^{t_{n-1/2}}(v_h),$$
$$- \sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \chi_h[\![u_h^{n-1/2}]\!]_E\,\mathrm{d}\gamma = -\sum_{E\in\mathcal{E}^h/\mathcal{E}_N^h}\int_E \chi_h u_{Dh}^{n-1/2}\,\mathrm{d}\gamma,$$

  for any testing pair $(v_h,\chi_h)\in X_h\times M_h$. Here,

$$F^{t_{n-1/2}}(v_h) = \sum_{T\in\mathcal{T}_h}\int_T f(t_{n-1/2})v_h\,\mathrm{d}x + \sum_{E\in\mathcal{E}_N^h}\int_E g(t_{n-1/2})v_h\,\mathrm{d}\gamma.$$

Note that the above problems have unique solutions $(u_h^n, \kappa_h^n) \in X_h \times M_h$ (cf. [9]). Optimal order error estimates of the displacement variable and the Lagrange multiplier can be derived as in [9,10].

Using the matrices $\mathbb{M}, \mathbb{B}, \mathbb{D}, \mathbb{C}$ constructed for the elliptic case, we write the above algebraic formulation in matrix-vector form,

$$M_1\ \mathrm{W}^n = M_2\ \mathrm{W}^{n-1} + \mathrm{F}. \tag{4.8}$$

Here,

$$\mathrm{W}^n = \begin{pmatrix} U^n \\ \Lambda^n \end{pmatrix}_{(N+L)\times 1}, \quad \mathrm{W}^{n-1} = \begin{pmatrix} U^{n-1} \\ \Lambda^{n-1} \end{pmatrix}_{(N+L)\times 1},$$

$$M_1 = \begin{cases} \begin{pmatrix} (1+k\delta)\mathbb{M} + k\mathbb{B} + k\mathbb{D} & -k\mathbb{C}' \\ -\mathbb{C} & 0 \end{pmatrix}_{(N+L)\times(N+L)}, & \text{for backward Euler;} \\ \begin{pmatrix} (1+\frac{1}{2}k\delta)\mathbb{M} + \frac{1}{2}k\mathbb{B} + \frac{1}{2}k\mathbb{D} & -\frac{1}{2}k\mathbb{C}' \\ -\frac{1}{2}\mathbb{C} & 0 \end{pmatrix}_{(N+L)\times(N+L)}, & \text{for Crank-Nicolson;} \end{cases} \tag{4.9}$$

$$M_2 = \begin{cases} \begin{pmatrix} \mathbb{M} & 0 \\ 0 & 0 \end{pmatrix}_{(N+L)\times(N+L)}, & \text{for backward Euler;} \\ \begin{pmatrix} (1-\frac{1}{2}k\delta)\mathbb{M} - \frac{1}{2}k\mathbb{B} - \frac{1}{2}k\mathbb{D} & \frac{1}{2}k\mathbb{C}' \\ \frac{1}{2}\mathbb{C} & 0 \end{pmatrix}_{(N+L)\times(N+L)}, & \text{for Crank-Nicolson;} \end{cases} \tag{4.10}$$

$$\mathrm{F} = \begin{cases} \begin{pmatrix} kF^{n-1} \\ b_D^{n-1} \end{pmatrix}_{(N+L)\times 1}, & \text{for backward Euler;} \\ \begin{pmatrix} kF^{n-1/2} \\ b_D^{n-1/2} \end{pmatrix}_{(N+L)\times 1}, & \text{for Crank-Nicolson;} \end{cases} \tag{4.11}$$

where
$$F^{n-1} = (F_j^{n-1})_{N\times 1}, \quad F_j^{n-1} = b_j^{n-1} + LN_j^{n-1},$$
$$F^{n-1/2} = (F_j^{n-1/2})_{N\times 1}, \quad F_j^{n-1/2} = b_j^{n-1/2} + LN_j^{n-1/2},$$
$$b_j^{n-1/2} = \frac{1}{2}b_j^{n-1} + \frac{1}{2}b_j^n, \quad LN_j^{n-1/2} = (LN_j^{n-1} + LN_j^n)/2.$$

For global basis functions ϕ_j,

$$b_j^{n-1} \approx \sum_{T\in\mathcal{T}_h}\int_T f^{n-1}\phi_j\,\mathrm{d}x, \quad LN_j^{n-1} \approx \sum_{E\in\mathcal{E}_N^h}\int_E g^{n-1}\phi_j\,\mathrm{d}\gamma.$$

Here, b_j^{n-1} is obtained as follows

$$b_j^{n-1} \approx \sum_{T\in\mathcal{T}_h}\frac{|T|}{3}\sum_{i=1}^{3} f(m_{E_i}, t_{n-1})\phi_j(m_{E_i}). \tag{4.12}$$

The Neumann boundary term LN_j^{n-1} is obtained by

$$LN_j^{n-1} \approx \sum_{E\in\mathcal{E}_N^h} |E| g(m_E, t_{n-1})\phi_j(m_E), \qquad \text{where } m_E \in \mathcal{N}_m(\Gamma_N). \tag{4.13}$$

The component related to the Dirichlet boundary $b_D^{n-1/2}$ is defined as follows

$$b_D^{n-1/2} = (b_{D_l}^{n-1/2})_{L\times 1}, \quad \text{where} \quad b_{D_l}^{n-1/2} = (b_{D_l}^{n-1} + b_{D_l}^n)/2.$$

Similarly, $b_{D_l}^{n-1}$ is obtained by

$$\begin{aligned} b_{D_l}^{n-1} &\approx -\int_{E_l} \psi_l u_{Dh}^{n-1}\,\mathrm{d}\gamma \approx -|E_l|\sigma_l u_{Dh}^{n-1}(m_{E_l}), && \text{where } m_{E_l} \in \mathcal{N}_m(\Gamma_D) \\ &= -|E_l| u_{Dh}^{n-1}(m_{E_l}), && \text{since } \sigma_l = 1 \text{ for } l = 1, \ldots, L. \end{aligned} \tag{4.14}$$

Table 5
Order of convergence in L^2, X and h-norms for second order parabolic model problem using backward Euler method.

Level	$h=k$	$\|u-u_h\|_X$	Order of convergence	$\|u-u_h\|_{L^2}$	Order of convergence	$\|\kappa-\kappa_h\|_h$	Order of convergence
1	1/2	0.1156		0.0209		0.2784	
2	1/4	0.0565	1.0321	0.0089	1.2329	0.1212	1.2000
3	1/8	0.0279	1.0195	0.0042	1.0629	0.0576	1.0735
4	1/16	0.0139	1.0091	0.0021	1.0182	0.0282	1.0274
5	1/32	0.0069	1.0043	0.0010	1.0060	0.0140	1.0113
6	1/64	0.0034	1.0021	0.0005	1.0023	0.0070	1.0051

Table 6
Order of convergence in L^2, X and h-norms for second order parabolic model problem using Crank-Nicolson scheme.

Level	$h=k$	$\|u-u_h\|_X$	Order of convergence	$\|u-u_h\|_{L^2}$	Order of convergence	$\|\kappa-\kappa_h\|_h$	Order of convergence
1	1/2	0.1108		0.0133		0.2135	
2	1/4	0.0497	1.1583	0.0028	2.2254	0.0729	1.5497
3	1/8	0.0226	1.1356	6.82e-04	2.0627	0.0309	1.2411
4	1/16	0.0106	1.0862	1.62e-04	2.0733	0.0142	1.1235
5	1/32	0.0051	1.0482	3.96e-05	2.0333	0.0068	1.0639
6	1/64	0.0025	1.0255	9.93e-06	1.9955	0.0033	1.0328

The functions **StiffMassConv_PH.m** and **Lambda_PH.m** for the parabolic model problem will be similar to those for the elliptic model problem, which will handle the assembly of the matrices $\mathbb{M}$, $\mathbb{B}$, $\mathbb{C}$, $\mathbb{D}$. But, for the Crank-Nicolson scheme, we need to compute the vector $F^{n-1/2}$ at $t_{n-1/2}$, it is achieved by evaluating at t_n and t_{n-1} time points then taking an average of them. This necessitates the utilization of another function **load_t.m** within **ParabolicMain.m**. The **load_t.m** calculates the values of b^{n-1} (4.12) and LN^{n-1} (4.13) at different time points t_{n-1}.

Remark 4. For the backward Euler finite difference scheme, we obtain a 1st-order convergence. In contrast, we obtain a second-order convergence for the primal variable in the L^2-norm using the Crank-Nicholson scheme. One can construct a complete discrete setup using any other explicit or implicit scheme (cf. [18]).

Example 3. In (4.1)-(4.4) we take a time-dependent exact solution

$$u(x_1,x_2,t) = t\,u(x_1,x_2) \qquad \text{in } \Omega\times[0,1]=(0,1)\times(0,1)\times[0,1], \tag{4.15}$$

where $u(x_1,x_2)$ is given by (3.20). We again consider $A=I_{2\times2}$, $p=[1,1]$, $\delta=1$. Then, the corresponding load function reads

$$f(x_1,x_2,t) = u(x_1,x_2) + t\,f(x_1,x_2), \qquad \text{in } \Omega\times[0,1]=(0,1)\times(0,1)\times[0,1], \tag{4.16}$$

where $f(x_1,x_2)$ is given by (3.21). The Dirichlet and Neumann boundary conditions (cf. Fig. 10) read

$$u_D(x_1,x_2,t) = 0 \qquad \text{on } \Gamma_D,$$
$$g(x_1,x_2,t) = t\,[(1-x_1-x_1^2)(x_2-x_2^2),(x_1-x_1^2)(1-x_2-x_2^2)]\cdot\nu \qquad \text{on } \Gamma_N.$$

Clearly, the initial condition is $u(x_1,x_2,t=0)=0$ in Ω.

Tables 5 and 6 generated by **ParabolicMain.m** show the error $u-u_h$ in the L^2 and X-norms, and the error $\kappa-\kappa_h$ in the h-norm. For optimal order convergence, the space variable h and the time parameter k are considered equal, that is, $h=k$. The order convergence of the Crank-Nicolson scheme is second-order in L^2-norm, and we have first-order accuracy for the backward Euler method. Examples of particular exact and approximate solutions are shown in Figs. 11, 12.

5. Concluding remarks

Primal hybrid FEM has been implemented efficiently in MATLAB for general two-dimensional elliptic and parabolic problems with mixed boundary conditions. Numerical experiments show that the codes are of nearly linear time-scaling, and the method converges optimally. The presented codes are flexible and may be extended to non-linear problems with an adequate amount of modification. We will include the implementations for the three-dimensional case in our future work.

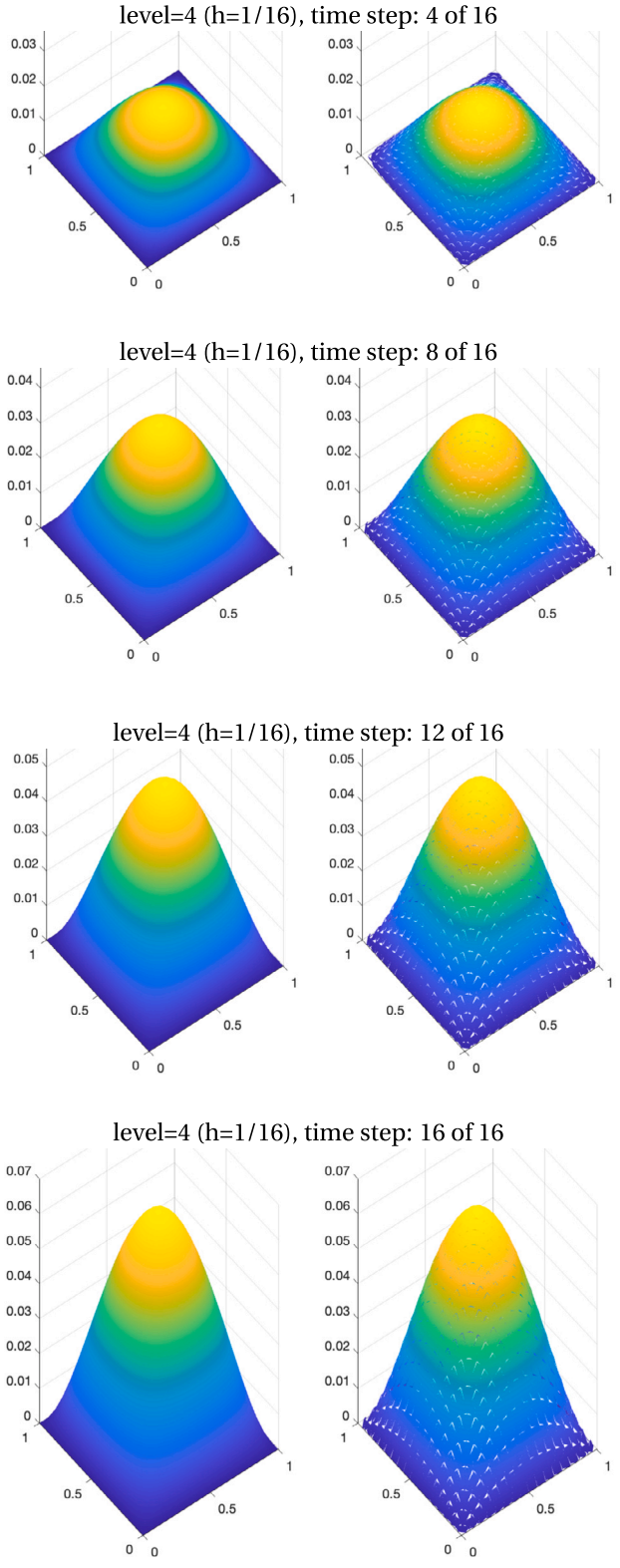

Fig. 11. Exact u (left column) vs. approximate u_h solutions from the broken Sobolev space (right column) of the parabolic model at selected time steps using Crank-Nicolson method.

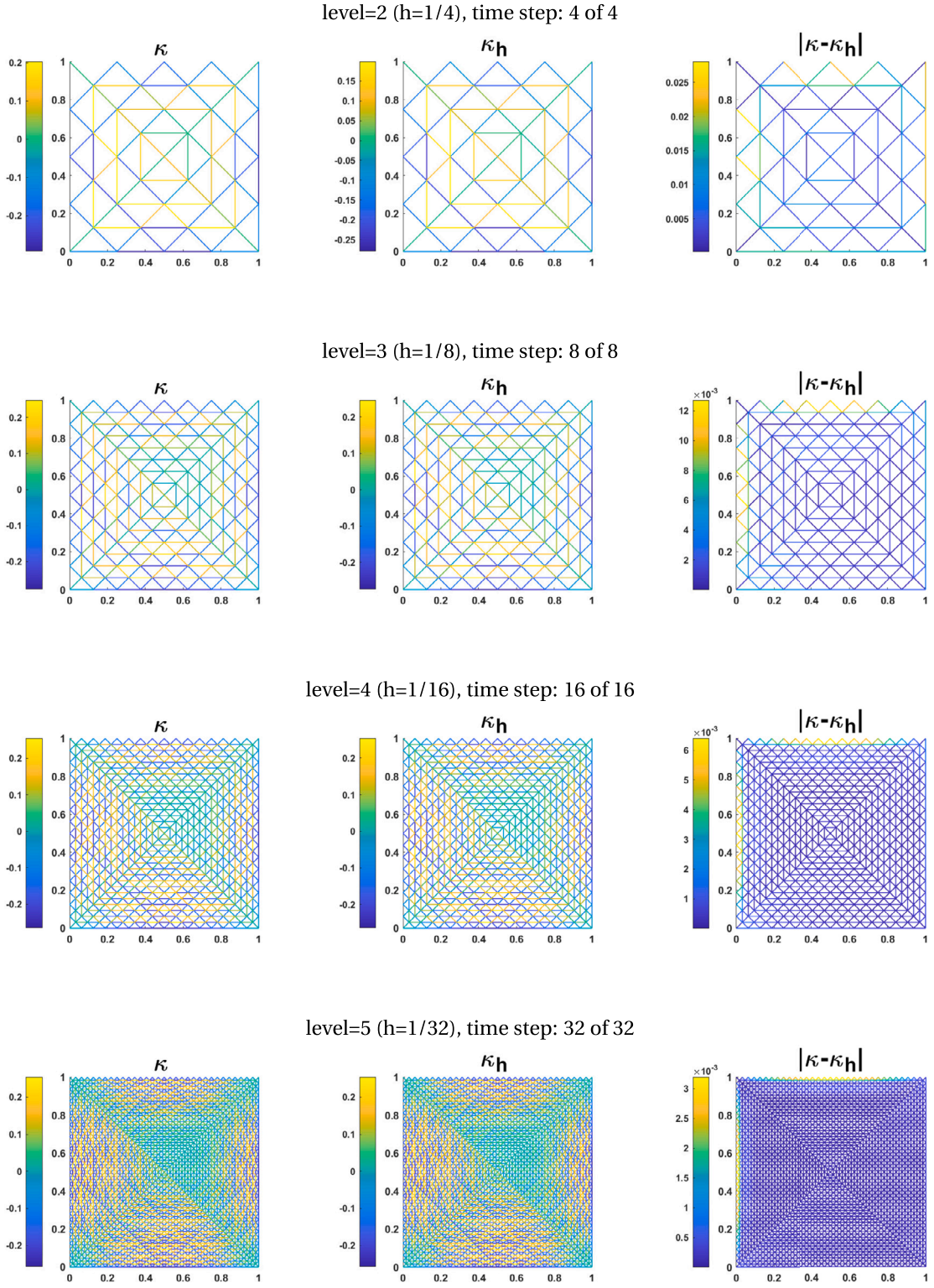

Fig. 12. Exact Lagrange multiplier κ, approximate Lagrange multiplier κ_h and $|\kappa - \kappa_h|$ of the parabolic model in the last time step from level 2 to 5 using Crank-Nicolson method.

Link to the Reproducible Capsule

The code (and data) in this article has been certified as Reproducible by Code Ocean: https://codeocean.com/capsule/5776148/tree

Acknowledgements

The second author's work is supported by the CSIR Extramural Research Grant 25(0297)/19/EMR-II. The third author announces the support of the Czech Science Foundation (GACR) through the GA23-04766S grant Variational approaches to dynamical problems in continuum mechanics. Finally, we express our gratitude to the anonymous reviewer who provided numerous constructive comments and suggestions.

Data availability

We have shared the link of the codes related to this article.

References

[1] J. Alberty, C. Carstensen, S.A. Funken, Remarks around 50 lines of Matlab: short finite element implementation, Numer. Algorithms 20 (1999) 117–137.
[2] C. Bahriawati, C. Carstensen, Three Matlab implementations of the lowest-order Raviart-Thomas mfem with a posteriori error control, Comput. Methods Appl. Math. 5 (2005) 333–361.
[3] T. Rahman, J. Valdman, Fast MATLAB assembly of FEM matrices in 2d and 3d: nodal elements, Appl. Math. Comput. 219 (2013) 7151–7158.
[4] I. Anjam, J. Valdman, Fast MATLAB assembly of FEM matrices in 2d and 3d: edge elements, Appl. Math. Comput. 267 (2015) 252–263.
[5] S. Funken, D. Praetorius, P. Wissgott, Efficient implementation of adaptive P1-FEM in Matlab, Comput. Methods Appl. Math. 11 (2011) 460–490.
[6] F. Cuvelier, C. Japhet, G. Scarella, An efficient way to assemble finite element matrices in vector languages, BIT Numer. Math. 56 (2016) 833–864.
[7] A. Moskovka, J. Valdman, Fast MATLAB evaluation of nonlinear energies using FEM in 2d and 3d: nodal elements, Appl. Math. Comput. 424 (2022) 127048.
[8] P.A. Raviart, J.M. Thomas, Primal hybrid finite element methods for 2nd order elliptic equations, Math. Comput. 31 (1977) 391–413.
[9] S.K. Acharya, A. Patel, Primal hybrid method for parabolic problems, Appl. Numer. Math. 108 (2016) 102–115.
[10] S.K. Acharya, K. Porwal, Primal hybrid finite element method for fourth order parabolic problems, Appl. Numer. Math. 152 (2020) 12–28.
[11] P. Wriggers, C. Carstensen, Mixed Finite Element Technologies, CISM Courses and Lectures, vol. 509, Springer, Wien New York, 2009.
[12] R. Shokeen, A. Patel, A.K. Pani, Primal hybrid method for quasilinear parabolic problems, J. Sci. Comput. 92 (2022) 10.
[13] V.B. Oliari, P.R. Bösing, D. de Siqueira, P. Devloo, A posteriori error estimates for primal hybrid finite element methods applied to Poisson problem, J. Comput. Appl. Math. 441 (2024) 115671.
[14] S. Funken, A. Schmidt, A coarsening algorithm on adaptive red-green-blue refined meshes, Numer. Algorithms 87 (2021) 1147–1176.
[15] J. Bey, Simplicial grid refinement: on freudenthal's algorithm and the optimal number of congruence classes, Numer. Math. 85 (2000) 1–29.
[16] H.N. Mallesham, K. Porwal, J. Valdman, S.K. Acharya, Vectorized implementation of primal hybrid fem in Matlab, Code Ocean, https://doi.org/10.24433/CO.9292176.v1, 2024.
[17] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, North-Holland, Amsterdam, 1978.
[18] V. Thomée, Galerkin Finite Element Methods for Parabolic Problems, Lecture Notes in Mathematics, Springer, Berlin, 1984.