AN EFFICIENT IMPLEMENTATION OF BREZZI-DOUGLAS-MARINI (BDM) MIXED FINITE ELEMENT METHOD IN MATLAB

SHUN ZHANG

Abstract. In this paper, a MATLAB package bdm_mfem for a linear Brezzi-Douglas-Marini (BDM) mixed finite element method is provided for the numerical solution of elliptic diffusion problems with mixed boundary conditions on unstructured grids. BDM basis functions defined by standard barycentric coordinates are used in the paper. Local and global edge ordering are treated carefully. MATLAB build-in functions and vectorizations are used to guarantee the erectness of the programs. The package is simple and efficient, and can be easily adapted for more complicated edge-based finite element spaces. A numerical example is provided to illustrate the usage of the package.

1. Introduction

In recent years, MATLAB is widely used in the numerical simulation and is proved to be an excellent tool for academic educations. For example, Trefethen’s book on spectral methods [15] is extremely popular. In the area of finite element method, there are several papers on writing clear, short, and easily adapted MATLAB codes, for example [1, 2, 10, 11]. Vectorizations are used in [10] and [11] to guarantee the effectiveness of the MATLAB finite elements codes. The mixed finite element [13, 8, 4] is now widely used in many area of scientific computation. For example, in [5, 6, 7, 8, 9], we use RT(Raviart-Thomas)/BDM(Brezzi-Douglas-Marini) space to build recovery-based a posteriori error estimators. On the other side, except for the clear presentation of [2] on RT₀, the implementation of more complicated BDM elements is still somehow confusing for researchers and students. The purpose of this paper is to fill this gap by giving a simple, efficient, and easily adaptable MATLAB implementation of BDM₁-P₀ mixed finite element methods for elliptic diffusion problems with mixed boundary conditions on unstructured grids.

For linear BDM elements, there are several versions to write the basis functions explicitly. In [4, 13], basis functions are defined on each elements using heights and normal vectors. This version of basis functions is less straightforward than the basis functions defined by barycentric coordinates. After all, everyone is very familiar with barycentric coordinates in finite element programmings. Thus, in this implementation, we will use the definition which only uses barycentric coordinates.

There are two basis functions on each edge for linear BDM elements. Unlike the RT element, BDM basis functions depend on the stating and terminal points of the edge. Thus, when assembling the local matrix, for a local edge on each element, we need to make sure we find its correct global stating and terminal points of the edge. There is an implementation...
of BDM element in iFEM package [10] https://bitbucket.org/ifem/ifem/. But in order to make the local ordering of edges in an element is the same as the global ordering of edges, the triangles are not always counterclockwisely oriented. This will cause confusion for programmers. And if we use this ordering, sometime we may need two kinds of element map, one is counterclockwisely ordered, the other is ordered by the indices of vertices. This will make things more complicated. In this implementation, we will use the standard counterclockwisely ordering of vertices of triangles.

Besides these issues, to get the right convergence order, we need to handle the boundary conditions carefully. In this package, we use the basic data structure of iFEM [10], and full MATLAB vectorization is used.

In a summary, in this package:

1. BDM$_1$ basis functions are explicitly defined using barycentric coordinates.
2. Elements are still positively oriented. A function is used to find the right global stating and terminal vertices of an edge in a local element.
3. Mixed type of boundary conditions are handled correctly to guarantee the right order of convergence.
4. MATLAB build-in functions and vectorizations are used to guarantee the erectness of the programs.

The package can be downloaded from http://personal.cityu.edu.hk/~szhang26/bdm-mfem.zip.

The paper is organized as follows. Section 2 describes the model diffusion problem, its BDM$_1$-P$_0$ mixed finite element approximation and the corresponding matrix problem. Section 3 introduces a simple example problem to demonstrate out MATLAB code. In Section 4, edge-based linear BDM basis functions are defined. The main part of the code for the matrix problem is discussed in Section 6. MATLAB functions to checking the errors are discussed in Section 7. Finally, we discuss some related finite elements in Section 8.

2. Model problem and BDM1-P0 mixed finite element method

2.1. Model problem. Let $\Omega$ be a bounded polygonal domain in $\mathbb{R}^2$, with boundary $\partial\Omega = \partial \Gamma_D \cup \partial \Gamma_N$, $\partial \Gamma_D \cap \partial \Gamma_N = \emptyset$, and measure $(\Gamma_D) \neq 0$, and let $n$ be the outward unit vector normal to the boundary. For a vector function $\mathbf{\tau} = (\tau_1, \tau_2)$, define the divergence and curl operators by $\nabla \cdot \mathbf{\tau} = \frac{\partial \tau_1}{\partial x_1} + \frac{\partial \tau_2}{\partial x_2}$ and $\nabla \times \mathbf{\tau} = \frac{\partial \tau_2}{\partial x_1} - \frac{\partial \tau_1}{\partial x_2}$. For a function $v$, define the gradient and rotation operators by $\nabla v = (\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2})^t$ and $\nabla \perp v = (\frac{\partial v}{\partial x_2}, -\frac{\partial v}{\partial x_1})^t$.

Consider diffusion equation
\begin{equation}
-\nabla \cdot (\alpha(x)\nabla u) = f \quad \text{in} \quad \Omega
\end{equation}
with boundary conditions
\begin{equation}
-\alpha \nabla u \cdot n = g_N \quad \text{on} \quad \Gamma_N \quad \text{and} \quad u = g_D \quad \text{on} \quad \Gamma_D.
\end{equation}
We assume that the right-hand side $f \in L^2(\Omega)$, $g_D \in H^{1/2}(\Gamma_D)$, and that $g_N \in L^2(\Gamma_N)$, and that $\alpha(x)$ is a positive piecewise constant function. Define the flux by $\mathbf{\sigma} = -\alpha(x)\nabla u$, then we have
\begin{equation}
\alpha^{-1} \mathbf{\sigma} = -\nabla u \quad \text{and} \quad \nabla \cdot \mathbf{\sigma} = f.
\end{equation}
Define the standard $H(\text{div}; \Omega)$ spaces as

$$H(\text{div}; \Omega) := \{\mathbf{\tau} \in L^2(\Omega)^2 : \nabla \cdot \mathbf{\tau} \in L^2(\Omega)\},$$

$$H_N(\text{div}; \Omega) := \{\mathbf{\tau} \in H(\text{div}; \Omega) : \mathbf{\tau} \cdot \mathbf{n} = 0 \text{ on } \Gamma_N\}.$$ 

Multiply the first equation in (2.3) by a $\mathbf{\tau} \in H_N(\text{div}; \Omega)$ and integrating by parts, we get

$$(\alpha^{-1}\mathbf{\sigma}, \mathbf{\tau}) = -(\nabla u, \mathbf{\tau}) = (\nabla \cdot \mathbf{\tau}, v) - (\mathbf{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D}$$

where $(\cdot, \cdot)_\omega$ is the $L^2$ inner product on a domain $\omega$. If $\omega = \Omega$, we omit the subscript. Then the mixed variational formulation is to find $(\mathbf{\sigma}, u) \in H(\text{div}; \Omega) \times L^2(\Omega)$ with $\mathbf{\sigma} \cdot \mathbf{n} = g_N$ on $\Gamma_N$, such that

$$\begin{align*}
(\alpha^{-1}\mathbf{\sigma}, \mathbf{\tau}) - (\nabla \cdot \mathbf{\tau}, u) &= -(\mathbf{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D} \quad \forall \mathbf{\tau} \in H_N(\text{div}; \Omega), \\
(\nabla \cdot \mathbf{\sigma}, v) &= (f, v) \quad \forall v \in L^2(\Omega).
\end{align*}$$

The existence, uniqueness, and stability results of (2.4) are well-known, and can be found in standard references, for example, [4].

2.2. BDM$_1$-P$_0$ mixed formulation. Let $\mathcal{T} = \{K\}$ be a regular triangulation of the domain $\Omega$. Denote the set of all edges of the triangulation by $\mathcal{E} := \mathcal{E}_I \cup \mathcal{E}_D \cup \mathcal{E}_N$, where $\mathcal{E}_I$ is the set of all interior element edges and $\mathcal{E}_D$ and $\mathcal{E}_N$ are the sets of all boundary edges belonging to the respective $\Gamma_D$ and $\Gamma_N$. For any element $K \in \mathcal{T}$, denote by $P_k(K)$ the space of polynomials on $K$ with total degree less than or equal to $k$. The $H(\text{div}; \Omega)$ conforming Brezzi-Douglas-Marini (BDM) space [3] of the lowest order is defined by

$$\text{BDM}_1 = \{\mathbf{\tau} : \mathbf{\tau}|_K \in \text{BDM}_1(K), \forall K \in \mathcal{T}\} \quad \text{with} \quad \text{BDM}_1(K) = P_1(K)^2.$$ 

and let $\text{BDM}_{1,N} = \text{BDM}_1 \cap H_N(\text{div}; \Omega)$. The piecewise constant space $P_0$ is defined by

$$P_0 = \{v : v|_K \in P_0(K), \forall K \in \mathcal{T}\}.$$ 

For simplicity, we further assume that $\alpha$ is a positive constant in each element $K \in \mathcal{T}$. Let $\mathcal{T}_N$ be the one dimensional mesh induced by $\mathcal{T}$ on $\Gamma_N$. Define

$$P_1(\mathcal{T}_N) = \{v : v|_E \in P_1(E), \forall E \in \mathcal{T}_N\}.$$ 

Let $g_{N,h}$ be the $L^2$ projection of $g_N$ on $P_1(\mathcal{T}_N)$. The BDM$_1$-P$_0$ mixed finite element discrete problem then is to find $(\mathbf{\sigma}_h, u_h) \in \text{BDM}_1 \times P_0$ with $\mathbf{\sigma}_h \cdot \mathbf{n} = g_{N,h}$ on $\Gamma_N$, such that

$$\begin{align*}
(\alpha^{-1}\mathbf{\sigma}_h, \mathbf{\tau}_h) - (\nabla \cdot \mathbf{\tau}_h, u_h) &= -(\mathbf{\tau}_h \cdot \mathbf{n}, g_D)_{\Gamma_D} \quad \forall \mathbf{\tau}_h \in \text{BDM}_{1,N}, \\
(\nabla \cdot \mathbf{\sigma}_h, v_h) &= -(f, v_h) \quad \forall v_h \in P_0.
\end{align*}$$

The discrete problem (2.5) has a unique solution, and the following error estimates hold assuming that the solution $(\mathbf{\sigma}, u)$ has enough regularity:

$$\|\alpha^{-1/2}(\mathbf{\sigma} - \mathbf{\sigma}_h)\|_0 \leq Ch^2\|\alpha^{-1/2}\mathbf{\sigma}\|_2 \quad \text{and} \quad \|u - u_h\|_0 \leq C(h\|u\|_1 + h^2\|\mathbf{\sigma}\|_2),$$

where $h$ is the mesh size and $\| \cdot \|_k$ is the standard norm of Sobolev space $H^k$. Thus, we should expect order 2 convergence of $\mathbf{\sigma}$ and order 1 convergence of $u$ if the solution is smooth enough.
Let $\sigma_N \in BDM_1$ be the interpolation of $g_{N,h}$ on $\Gamma_N$ (The construction of $\sigma_N$ will be explained in Section 2.7 in detail). Then $\sigma_h = \sigma_N + \sigma_0$, with $\sigma_0 \in BDM_{1,N}$ solves the following discrete problem:

$$
\begin{align*}
\left\{ \begin{array}{l}
(\alpha^{-1} \sigma_0, \tau_h) - (\nabla \cdot \tau_h, u_h) = -((\tau_h \cdot n, g_D)_\Gamma) - (\alpha^{-1} \sigma_N, \tau_h) \quad \forall \tau_h \in BDM_{1,N}, \\
-(\nabla \cdot \sigma_0, v_h) = -(f, v_h) + (\nabla \cdot \sigma_N, v_h) \quad \forall v_h \in P_0.
\end{array} \right.
\end{align*}
$$

(2.7) 

2.3. Matrix problem. Suppose that all edges are uniquely defined with fixed initial and terminal vertices, in Section 2.7 we will define two basis functions $\phi_{j,1}$ and $\phi_{j,2}$ for an edge $E_j \in \mathcal{E}$, $j = 1, \ldots, NE$, with $NE$ is the number of edges. For simplicity, we assume that the total number of all edges in $\mathcal{E}_I$ and $\mathcal{E}_D$ is $M$, and $BDM_{1,N} = \text{span}\{\phi_{j,1}, \phi_{j,2}, j = 1 \cdots, M\}$ (The real mesh may has a different order of indices). The basis of $P_0$ is very simple. For the element $K_j$, its basis is $1_j$ which is 1 on $K_j$ and 0 elsewhere.

We want to compute the coefficient vectors $x \in \mathbb{R}^{2NE}$ of $\sigma_h$ and $y \in \mathbb{R}^{NT}$ of $u_h$ with respect to the $BDM_1$ basis and $P_0$ basis

$$
\sigma_h = \sum_{j=1}^{NE} (x_j \phi_{j,1} + x_{NE+j} \phi_{j,2}) \quad \text{and} \quad u_h = \sum_{k=1}^{NT} y_k 1_k,
$$

(2.8) 

and

$$
\sigma_N = \sum_{j=M+1}^{NE} (x_j \phi_{j,1} + x_{NE+j} \phi_{j,2})
$$

(2.9) 

with $x_j$ and $x_{NE+j}$ are determined by $g_{N,h}$ (discussed later in Section 2.7). Then

$$
\begin{align*}
\sum_{j=1}^{M} (x_j(\alpha^{-1} \phi_{i,1}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1} \phi_{i,1}, \phi_{j,2})) - \sum_{k=1}^{NT} y_k (\phi_{i,1}, 1) K_k \\
= - (g_D, \phi_{i,1} \cdot n)_\Gamma - \sum_{j=M+1}^{NE} (x_j(\alpha^{-1} \phi_{i,1}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1} \phi_{i,1}, \phi_{j,2})); \\
\sum_{j=1}^{M} (x_j(\alpha^{-1} \phi_{i,2}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1} \phi_{i,2}, \phi_{j,2})) - \sum_{k=1}^{NT} y_k (\phi_{i,2}, 1) K_k \\
= - (g_D, \phi_{i,2} \cdot n)_\Gamma - \sum_{j=M+1}^{NE} (x_j(\alpha^{-1} \phi_{i,2}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1} \phi_{i,2}, \phi_{j,2})); \\
- \sum_{j=1}^{M} (x_j(\phi_{j,1}, 1) K_i + x_{j+NE}(\phi_{j,2}, 1) K_i) \\
= -(f, 1) K_i + \sum_{j=M+1}^{NE} (x_j(\nabla \phi_{j,1}, 1) K_i + x_{j+NE}(\nabla \phi_{j,2}, 1) K_i);
\end{align*}
$$

(2.9)
for \( i = 1, \cdots, M, \ell = 1, \cdots, NT \). The coefficients \( x_j, x_{j+NE}, j = M + 1, \cdots, NE \) are known from \( g_{N,h} \). Rewrite it as a matrix problem, we have

\[
(2.10) \quad \begin{pmatrix} B & C^t \\ C & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},
\]

where \( B \) is a \( 2M \times 2M \) matrix and \( C \) is a \( NT \times 2M \) matrix with entries:

\[
(2.11) \quad B_{i,j} = (\alpha^{-1} \phi_{i,1}, \phi_{j,1}), \quad B_{i,i+M} = (\alpha^{-1} \phi_{i,1}, \phi_{j,2}), \quad B_{i+M,j+M} = (\alpha^{-1} \phi_{i,2}, \phi_{j,2}), \quad C_{j,j} = (\nabla \cdot \phi_{j,1}, 1), \quad C_{j,i} = (\nabla \cdot \phi_{j,2}, 1),
\]

where \( i = 1, \cdots, M, j = 1, \cdots, M, \) and \( \ell = 1, \cdots, NT \); and the right hand side where \( b_1 \) is a \( 2M \times 1 \) vectors and \( b_2 \) is an \( NT \times 1 \) vector with entries

\[
b_i = -(g_D, \phi_{i,1} \cdot n_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j (\alpha^{-1} \phi_{i,1}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,1}, \phi_{j,2}));
\]

\[
b_{i+M} = -(g_D, \phi_{i,2} \cdot n_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j (\alpha^{-1} \phi_{i,2}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,2}, \phi_{j,2}));
\]

\[
b_{2\ell} = -(f, 1)_{K_\ell} + \sum_{j=M+1}^{NE} (x_j (\nabla \cdot \phi_{j,1}, 1)_{K_\ell} + x_{j+NE} (\nabla \cdot \phi_{j,2}, 1)_{K_\ell})
\]

where \( i = 1, \cdots, M \) and \( \ell = 1, \cdots, NT \); \((x_1, \cdots, x_M, x_{M+NE}, \cdots, x_{M+NE}, y_1, \cdots, y_{NT})^t\) is the \( 2M+NT \times 1 \) solution vector.

3. A Numerical Example

We will demonstrate our MATLAB program by a simple test problem. Let \( \Omega = (-1,1)^2 \), with \( \Gamma_N = \{ x \in (-1,1) \} \times \{ y = 1 \} \), and the rest is \( \Gamma_D \) The mesh is given in Fig. 3. We choose the diffusion coefficient and the exact solution to be

\[
\alpha = \begin{cases} 10 & \text{if } x < 0; \\ 1 & \text{if } x > 0; \end{cases} \quad \text{and} \quad u(x, y) = \begin{cases} \frac{x^2 y^2 + x + y}{10} & \text{if } x < 0; \\ \frac{x^2 y^2 + x + y}{10} & \text{if } x > 0. \end{cases}
\]

The right-hand side \( f = -2(x^2 + y^2) \). The exact \( \sigma \) is

\[
\sigma(x, y) = \begin{cases} -(2xy^2 + 1, 2x^2y + 10)^t & \text{if } x < 0; \\ -(2xy^2 + 1, 2x^2y + 10)^t & \text{if } x > 0. \end{cases}
\]

It’s clear that \( \sigma \) itself is not continuous, but its normal component is continuous. The boundary conditions are

\[
u(x, y) = \begin{cases} \frac{y^2/10 + y - 1/10}{10} & \text{if } x = -1; \\ y^2 + y + 1 & \text{if } x = 1; \\ (x^2 + x)/10 - 1 & \text{if } x < 0 \text{ and } y = -1; \\ x^2 + x - 1 & \text{if } x > 0 \text{ and } y = -1; \end{cases}
\]

and

\[
-\alpha \nabla u \cdot (0,1)^t = g_N = \begin{cases} -2x^2 - 10 & \text{if } x < 0 \text{ and } y = 1; \\ -2x^2 - 1 & \text{if } x > 0 \text{ and } y = 1. \end{cases}
\]

We choose this exact solution to emphasize that the flux \( \sigma \) is in \( H(\text{div}; \Omega) \), but not the \( \nabla u \). The main MATLAB codes of \( \alpha, f, g_D \) and \( g_N \) are given in exactalpha.m, f.m,
LISTING 1. exactalpha, f, gD, and gN

```
function z = exactalpha(p)
ix = (p(:,1)<0); z = ones(size(p,1),1); z(ix) = 10.0;
end
function z = f(p)
x = p(:,1); y = p(:,2); z = -2*(x.*x+y.*y);
end
function z = gD(p)
x = p(:,1); y = p(:,2);
z=(x<0).*(0.1*x.*x.*y.*y+0.1*x+y)+(x>=0).*(x.*x.*y.*y+x+y);
end
function z = gN(p)
x = p(:,1); y = p(:,2); z=(x<0).*(-2*x.*x-10)+(x>=0).*(-2*x.*x-1);
end
```

The MATLAB program to solve this problem is given in main.m. Lines 2-7 read the essential geometric data of the problem. Line 9 generates the edges, and other necessary geometric relations. Lines 11-12 solve the problem by the BDM1 mixed finite element method.

4. TRIANGULATION AND GEOMETRIC DATA STRUCTURES

4.1. Geometric description and geometric relations. We follow [10] for the data representation of the set of all vertices, the regular triangulation $T$, the edges, and the boundaries.

The set of all vertices $\mathcal{N} = \{z_1, \cdots, z_N\}$ is represented by an $N \times 2$ matrix $node(1:N,1:2)$, where $N$ is the number of vertices, and the $i$-th row of node is the coordinates of the $i$-th vertex $z_i = (x_i, y_i)$, $node(i,:) = [x_i, y_i]$. Lines 2-3 of main.m give the node of our example. For example, the 1st vertex has coordinates $x_1 = -1$ and $y_1 = 1$. 

Figure 1. mesh of the example Figure 2. a triangle

The MATLAB program to solve this problem is given in main.m. Lines 2-7 read the essential geometric data of the problem. Line 9 generates the edges, and other necessary geometric relations. Lines 11-12 solve the problem by the BDM1 mixed finite element method.
Listing 2. main.m

```matlab
1  %% Geometry setting
2  node = [-1 1; 0 1; 1 1; -0.5 0.5; 0.5 0.5; -1 0; 0 0; 1 0;...
3       -0.5 -0.5; 0.5 -0.5; -1.0 -1.0; 0.0 -1.0; 1.0 -1.0];
4  elem = [4 2 1; 4 1 6; 4 6 7; 4 7 2; 5 3 2; 5 2 7; 5 7 8; 5 8 3; 9 7 6;...
5       9 6 11; 9 11 12; 10 8 7; 10 7 12; 10 12 13; 10 13 8];
6  bdEdge = [2 0 0; 1 0 0; 0 0 0; 0 0 0; 2 0 0; 0 0 0; 0 0 0; 1 0 0;...
7       0 0 1; 0 1 0; 0 0 0; 0 0 0; 0 0 0; 0 1 0; 0 1 0];
8  %% Geometric relations
9  [edge,elem2edge,signedge] = geomrelations(elem);
10  %%
11  [sigma,u] = diffusionbdm(node,elem,bdEdge,elem2edge,edge,...
12     signedge,@exactalpha,@f,@gD,@gN);
```

Listing 3. geomrelations.m

```matlab
1  function [edge,elem2edge,signedge] = geomrelations(elem)
2  NT = size(elem,1);
3  totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
4  [edge, useless, j] = unique(totalEdge,'rows');
5  elem2edge = reshape(j,NT,3);
6
7  signedge = ones(NT,3);
8  signedge(:,1) = signedge(:,1) - 2* (elem(:,2)>elem(:,3));
9  signedge(:,2) = signedge(:,2) - 2* (elem(:,3)>elem(:,1));
10  signedge(:,3) = signedge(:,3) - 2* (elem(:,1)>elem(:,2));
11  end
```

The triangulation \( T \) is represented by an \( NT \times 3 \) matrix \( \text{elem}(1:NT,1:3) \) with \( NT \) the number of elements. The \( i \)-th element \( K_i = \text{conv}\{z_i,z_j,z_k\} \) is stored as \( \text{elem}(i,:) = [i~j~k] \), where the vertices are given in the counterclockwise order. Lines 4-5 of `main.m` give the `elem` of our example. For example, the 1st element \( K_1 \) has three vertices in the order of \( z_4, z_2, z_1 \).

We call the the opposite edge of the \( i \)-th vertex, \( i = 1, 2, 3 \) of a triangle the \( i \)-th edge of the triangle.

The matrix `bdEdge(1:NT, 1:3)` indicates which edge of an element is on the boundary of the domain. For a non-boundary edge, the value is 0; the value is 1 or 2 for a Dirichlet or Neumann boundary edge respectively. Lines 4-5 of `main.m` give the `bdEdge` of our example. For example, the 1st element \( K_1 \) has three edges, the first edge is on the Neumann boundary, and the other two are not on the boundary.

The MATLAB code `geomrelations.m` generates `edge`, `elem2edge`, and `signedge`. The explanations of these codes can be found in [10].

The matrix `edge(1:NE, 1:2)` is defined with \( NE \) the total number of edges, and the \( k \)-th edge \( E_k \) with the starting vertex \( z_i \) and the terminal vertex \( z_j \) is stored as \( \text{edge}(k,:) = [i~j] \). We always ensure that \( \text{edge}(k,1) < \text{edge}(k,2) \). Lines 3-4 of `geomrelations.m`
Listing 4. gradlambda.m

```matlab
function [a,b,area] = gradlambda(node,elem)

n1 = elem(:,1); n2 = elem(:,2); n3 = elem(:,3);
NT = size(elem,1); a = zeros(NT,3); b = zeros(NT,3);
a(:,1) = node(n2,2)-node(n3,2); b(:,1) = node(n3,1)-node(n2,1);
a(:,2) = node(n3,2)-node(n1,2); b(:,2) = node(n1,1)-node(n3,1);
a(:,3) = node(n1,2)-node(n2,2); b(:,3) = node(n2,1)-node(n1,1);
area = (a(:,2).*b(:,3)-a(:,3).*b(:,2))/2.0;
end
```

generate edge. For our example, NE=28 and edge=[1 2; 1 4; 1 6; 2 3; 2 4; 2 ... 5; 2 7; 3 5; 3 8; 4 6; 4 7; 5 7; 5 8; 6 7; 6 9; 6 11; 7 8; 7 9; 7 10; ... 7 12; 8 10; 8 13; 9 11; 9 12; 10 12; 10 13; 11 12; 12 13]. For example, the 2nd edge’s starting vertex is z1 and terminal vertex is z4.

For an edge E_j with the starting vertex z_s and the terminal vertex z_t, its normal direction is defined as (y_t−y_s, x_s−x_t)/|E_j|.

The matrix elem2edge(1:NT,1:3) is the matrix whose k-th row represents the 3 global indices of the edges in the order of local edges. Line 5 of geomrelations.m generates elem2edge. For our example, elem2edge= [1 2 5; 3 10 2; 14 11 10; 7 ... 5 11; 4 6 8; 7 12 6; 17 13 12; 9 8 13; 14 15 18; 16 23 15; 27 24 23; ... 20 18 24; 17 19 21; 20 25 19; 28 26 25; 22 21 26]. For example, the 2nd element’s three edges are E_3, E_{10}, and E_2.

Each edge has its fixed global orientation, while on each element, it has a local orientation, localEdge=[2 3; 3 1; 1 2]. The matrix signedge is an NT×3 matrix with value 1 denoting the local and global orientations are the same, and −1 denoting that they are different. For our example, signedge=[-1 1 -1; 1 -1 -1; 1 -1 1; -1 1 -1; -1 1 -1; 1 -1 1; -1 1 ... -1; 1 -1 -1; 1 -1 1; -1 1 1].

4.2. Barycentric coordinate and its gradient. On an element K with counterclockwise vertices {z_1, z_2, z_3}, we define the barycentric coordinate \( \lambda_i = (a_i x + b_i y + c_i)/(2|K|) \), \( i = 1, 2, 3 \), such that \( \lambda_i(z_j) = \delta_{ij} \). We only need to compute the gradient (and curl, rot, div) operators in this paper, so we only need to compute \( a_i, b_i \), and the area \( |K| \). The formulas are

\[
\nabla \lambda_i = \frac{1}{2|K|} \begin{pmatrix}
  y_{i+1} - y_i \\
  x_{i+1} - x_i
\end{pmatrix} \quad \text{and} \quad 2|K| = \det \begin{pmatrix}
  x_2 - x_1 & x_3 - x_1 \\
  y_2 - y_1 & y_3 - y_1
\end{pmatrix}.
\]

The function gradlambda.m computes the coefficients a, b, and area. Here a and b are two NT×3 matrices, with each row stores the coefficients a and b for the three barycentric coordinates of corresponding 3 vertices.

4.3. Normal and tangential vectors. On a local element K with counterclockwise oriented vertices {z_1, z_2, z_3}, and \( E_1 = \{ z_2, z_3 \} \), \( E_2 = \{ z_3, z_1 \} \), and \( E_3 = \{ z_1, z_2 \} \) (Figure 3), we will discuss the \( \nabla \) and \( \nabla^\perp \) of \( \lambda_2 \) as examples:

\[
\nabla^\perp \lambda_2 = \frac{1}{2|K|} \begin{pmatrix}
  y_3 - y_1 \\
  x_1 - x_3
\end{pmatrix} \quad \text{and} \quad \nabla^\perp \lambda_2 = \frac{1}{2|K|} \begin{pmatrix}
  x_1 - x_3 \\
  y_1 - y_3
\end{pmatrix}.
\]
where \(|K|\) is the area of \(K\). On edge \(E_1 = \{z_2, z_3\}\), the unit tangential and normal vectors are

\[
t_{E_1} = \frac{1}{|E_1|} \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} \quad \text{and} \quad n_{E_1} = \frac{1}{|E_1|} \begin{pmatrix} y_3 - y_2 \\ x_2 - x_3 \end{pmatrix},
\]

where \(|E|\) is the length of \(E\). From Figure 3, we have \(\begin{pmatrix} x_1 - x_3 \\ y_1 - y_3 \end{pmatrix} \cdot n_{E_1} = \begin{pmatrix} y_3 - y_1 \\ x_1 - x_3 \end{pmatrix} \cdot n_{E_1}\). 

\(t_{E_1} = -H_1\), with \(H_1\) is the height of \(K_1\) on \(E_1\). Now, by the fact \(2|K| = |E_1| \cdot H_1\), we have

\[
\nabla^\bot \lambda_2 \cdot n_{E_1} = \nabla \lambda_2 \cdot t_{E_1} = -1/|E_1|.
\]

Similarly,

\[
\nabla^\bot \lambda_3 \cdot n_{E_1} = \nabla \lambda_3 \cdot t_{E_1} = 1/|E_1|.
\]

On the hand, since the tangential and normal vectors of an edge are orthogonal, we have

\[
(4.1) \quad \nabla^\bot \lambda_2 \cdot n_{E_2} = \nabla^\bot \lambda_3 \cdot n_{E_3} = 0.
\]

Globally, on \(E_\ell = \{z_s, z_t\}\) \((s < t)\), the unit tangential and normal vectors are

\[
t_{E_\ell} = \frac{1}{|E_\ell|} \begin{pmatrix} x_\ell - x_s \\ y_\ell - y_s \end{pmatrix} \quad \text{and} \quad n_{E_\ell} = \frac{1}{|E_\ell|} \begin{pmatrix} y_\ell - y_s \\ x_\ell - x_s \end{pmatrix}.
\]

We call the adjacent element whose out unit normal vector is the same as \(n_{E_\ell}\) as \(K_\ell^-\), and the element whose out unit normal vector is the opposite of \(n_{E_\ell}\) as \(K_\ell^+\). On both \(K_\ell^-\) and \(K_\ell^+\), we have

\[
(4.2) \quad \nabla^\bot \lambda_s \cdot n_{E_\ell} = \nabla \lambda_s \cdot t_{E_\ell} = -1/|E_\ell| \quad \text{and} \quad \nabla^\bot \lambda_\ell \cdot n_{E_\ell} = \nabla \lambda_\ell \cdot t_{E_\ell} = 1/|E_\ell|.
\]

5. Constructions of edge-based BDM basis functions

Let \(\lambda_i\) be the standard linear Lagrange finite element basis function of the vertex \(z_i\), i.e., it is piecewise linear on each element, globally continuous, 1 at \(z_i\) and 0 at other vertices.

For an edge \(E_\ell = \{z_s, z_t\}\), \(s < t\), \(1 \leq \ell \leq \text{NE}\) with the globally fixed starting vertex \(s\) and terminal vertex \(t\), its two BDM1 basis functions associated with the edge are

\[
(5.1) \quad \phi_{\ell,1} = \lambda_s \nabla^\bot \lambda_\ell \quad \text{and} \quad \phi_{\ell,2} = -\lambda_\ell \nabla^\bot \lambda_s.
\]

It’s clear that the basis functions are only non-zero in the two adjacent elements \(K_\ell^-\) and \(K_\ell^+\) (one element in the case of boundary) of \(E_\ell\).

**Lemma 5.1.** There hold

\[
(5.2) \quad \phi_{\ell,1} \cdot n_{E_k}|_{E_k} = \begin{cases} 0, & \text{if } k \neq \ell; \\
 \lambda_s/|E_\ell|, & \text{if } k = \ell; \end{cases} \quad \text{and} \quad \phi_{\ell,2} \cdot n_{E_k}|_{E_k} = \begin{cases} 0, & \text{if } k \neq \ell; \\
 \lambda_\ell/|E_\ell|, & \text{if } k = \ell, \end{cases}
\]

and

\[
(5.3) \quad \nabla \cdot \phi_{\ell,1} = \nabla \cdot \phi_{\ell,2} = \frac{1}{2|K^-|} \text{ on } K^- \quad \text{and} \quad \nabla \cdot \phi_{\ell,1} = \nabla \cdot \phi_{\ell,2} = -\frac{1}{2|K^+|} \text{ on } K^+.
\]

**Proof.** By the discussion in Section 4.3 we have

\[
\nabla^\bot \lambda_s \cdot n_{E_\ell} = -1/|E_\ell| \quad \text{and} \quad \nabla^\bot \lambda_\ell \cdot n_{E_\ell} = 1/|E_\ell|,
\]

and

\[
\phi_{\ell,1} \cdot n_{E_\ell}|_{E_k} = \lambda_s/|E_\ell| \quad \text{and} \quad \phi_{\ell,2} \cdot n_{E_\ell}|_{E_k} = \lambda_\ell/|E_\ell|.
\]

Assume \(K_\ell^-\)’s three counterclockwisely ordered vertices are \(\{z_{s-}, z_s, z_t\}\). We call edge with two endpoints \(z_t\) and \(z_{s-}\) to be \(E_{t,r-}\), and the edge with two endpoint \(z_{r-}\) and \(z_s\)
to be $E_{s,r-}$. Here the orientations of these two edges are not important. Then by (4.1),
$\nabla^2 \lambda_s \cdot n_{E_{s,r-}} = 0$, and the fact $\lambda_t$ is zero on $E_{s,r-}$, we get
$$\phi_{t,2} \cdot n_E = 0, \text{ with } E = E_{s,r-} \text{ or } E_{t,r-}.$$  

We can prove (5.2) is true for $\phi_{t,1}$ and $K^+_t$ similarly.

The divergence part of the theorem can be easily proved by recalling the definition of $\lambda_s$ and $\lambda_t$ on $K^-_t$ and $K^+_t$, and notice that the counterclockwise ordering of vertices on $K^-_t$ is $\{z_{r-}, z_s, z_t\}$, and that on $K^+_t$ is $\{z_{r+}, z_t, z_s\}$.

**Remark 5.2.** There are other definitions of the $BDM_1$ basis functions. One of the popular choice is the following:

$$(5.4) \quad \phi_{t,1} = \lambda_s \nabla^\perp \lambda_i - \lambda_t \nabla^\perp \lambda_s \quad \text{and} \quad \phi_{t,2} = \lambda_s \nabla^\perp \lambda_t + \lambda_t \nabla^\perp \lambda_s.$$  

The first basis function coincides with the $RT_0$ basis function with property $\phi_{t,1} \cdot n_{E_k} = |E_t| \delta_{t,k}$. This set of choices of basis functions is hierarchical. The reason we choose (5.1) is that it is symmetric. If the reader needs the basis to be hierarchical, one should choose (5.4).

**Remark 5.3.** The other versions of basis functions, for example, the RT basis function used in [2], choose $\phi_{t,1} = \nabla^\perp \delta_{t,k}$. The issue of this choice of basis functions is that the mass matrix has a condition number $Ch_{\max}/h_{\min}$, where $h_{\max}$ and $h_{\min}$ are the maximal and minimal diameters of the elements respectively. This will be a problem for an adaptively generated mesh. Though it can be easily fixed by preconditioning it with the inverse of the diagonal matrix of it, we avoid it by choosing basis in (5.1) or (5.4).

6. Constructing and solving the matrix problem

6.1. Assembling matrices.

6.1.1. Local matrices. For an element $K$, we want to compute the local contributions of this element to the matrices $B$ and $C$.

We use $\text{localEdge}=\{2,3;1,1,2\}$ to denote the local edge with respect to the local indices of vertices. For the $i$-th edge, we let $ii1 = \text{localEdge}(i,1)$ and $ii2 = \ldots$ $\text{localEdge}(i,2)$ to denote the local starting and terminal vertices of it. When the local orientation and the global orientation of the edge $E$ is different ($\text{signedge}$ of the edge is $-1$), we need to switch the local order to find the right global starting and terminal vertices of edge. For a given $i$-th local edge, by the line $i1 = (\text{signedge}(;i)>0) \cdot ii1 + \ldots (\text{signedge}(;i)<0) \cdot ii2$, we get $i1=ii1$ if the local orientation and the global orientation are the same, and $i1=ii2$ otherwise. $i2$ can be done similarly. Once we find the $i1$ (local starting vertex) and $i2$ (local terminal vertex) with right global orientation, we can get the corresponding coefficients $a_{i1}$ and $b_{i1}$ of $\lambda_{i1} = (a_{i1}x + b_{i1}y + c_{i1})/(2|K|)$, and the same things for $i2$. The function $\text{BDMrightorder.m}$ does the above job. With an input of $i = 1, 2,$ or $3$ be the local vertex index of an element, this function returns the values $i1$ and $i2$, which are the correct starting and terminal vertices of the corresponding edge with respect to the global fixed edge orientation, and $ai1, ai2, bi1, bi2$ are the corresponding coefficients of $\lambda_{i1}$ and $\lambda_{i2}$.

To compute the local contribution of an element $K$ to $B$, the local mass matrix contains three cases, $(\alpha^{-1}\phi_{i,1}, \phi_{j,1}), (\alpha^{-1}\phi_{i,1}, \phi_{j,2}),$ and $(\alpha^{-1}\phi_{i,2}, \phi_{j,2})$ with

$$\phi_{i,1} = \lambda_{i1} \nabla^\perp \lambda_{i2}, \quad \phi_{i,2} = -\lambda_{i2} \nabla^\perp \lambda_{i1}, \quad \phi_{j,1} = \lambda_{j1} \nabla^\perp \lambda_{j2}, \quad \text{and} \quad \phi_{j,2} = -\lambda_{j2} \nabla^\perp \lambda_{j1}.$$
For barycentric coordinates, \( \int_K \lambda_i \lambda_j dx = (1 + \delta_{i,j})|K|/12 \). An easy computation shows that

\[
(\alpha^{-1} \phi_{i,1}, \phi_{j,1})_K = \alpha^{-1}(1 + \delta_{i,j}) \frac{a_{i2} \cdot a_{j2} + b_{i2} \cdot b_{j2}}{48|K|};
\]

\[
(\alpha^{-1} \phi_{i,1}, \phi_{j,2})_K = -\alpha^{-1}(1 + \delta_{i,j}) \frac{a_{i1} \cdot a_{j1} + b_{i1} \cdot b_{j1}}{48|K|};
\]

\[
(\alpha^{-1} \phi_{i,2}, \phi_{j,2})_K = \alpha^{-1}(1 + \delta_{i,j}) \frac{a_{i1} \cdot a_{j1} + b_{i1} \cdot b_{j1}}{48|K|}.
\]

For \( k = 1, 2 \), \( \nabla \cdot \phi_{i,k} = 1/(2|K|) \) when the \( i \)-th edge has the same orientation as the global edge, and \(-1/(2|K|)\) otherwise. Thus, denote \( s(i)\) be the value of \( \text{signedge}(:,i) \), we have

\[-(\nabla \cdot \phi_{i,1}, 1)_K = -(\nabla \cdot \phi_{i,2}, 1)_K = -s(i)/2.\]

The local matrix (here we abuse the notations to use local \( \phi_{i,k}, i = 1 \cdots 3, k = 1, 2 \) to denote the local BDM basis functions):

\[-((\nabla \phi_{1,1}, 1)_K, (\nabla \phi_{2,1}, 1)_K, (\nabla \phi_{3,1}, 1)_K, (\nabla \phi_{1,2}, 1)_K, (\nabla \phi_{2,2}, 1)_K, (\nabla \phi_{3,2}, 1)_K)\]

is simply \(-s(1), s(2), s(3), s(1), s(2), s(3))/2\), or \([-\text{signedge}(:,1:3), \text{signedge}(:,1:3)]\) in MATLAB code.

6.1.2. Assembling global matrices. For a local index \( i \), its corresponding global edge index is \text{double}(elem2edge(:,i)). MATLAB function sparse is used to generate the global matrices from local contributions. This is one of the key step to ensure the vectorization of the MATLAB finite element code, see [10,11] for more detailed discussions on sparse.

Here are some comments of the MATLAB code assemblebdm.m.

- Line 1: The function is called by

  \[ A = \text{assemblebdm}(NT, NE, a, b, area, elem2edge, signedge, inva) \]

  where \( A \) is a \((2 \times NE+NT) \times (2 \times NE+NT)\) matrix, \( a, b, area \) are the coefficients of local barycentric coordinates, signedge is the \( NT \times 3 \) matrix about the local and global edge orientations, and inva is \( NT \times 1 \) vector of \( \alpha^{-1} \).

- Lines 4-15: We generates the matrix \( B = (\alpha^{-1} \mathbf{\sigma}_h, \mathbf{\tau}_h) \), \( \mathbf{\sigma}_h \) and \( \mathbf{\tau}_h \) in BDM1 by assembling local element-wise contributions.

- Lines 6-7: We generates the right starting and terminal indices of a global edge in the local element, and their corresponding coefficients of local barycentric coordinates.

- Lines 8-10: We compute \( E, H, \) and \( G \), which are \((\alpha^{-1} \phi_{i,1}, \phi_{j,1})_K, (\alpha^{-1} \phi_{i,1}, \phi_{j,2})_K, \) and \((\alpha^{-1} \phi_{i,2}, \phi_{j,2})_K\), respectively.

- Lines 11-13: \( B \) is generated by sparse.

\begin{verbatim}
function \[\text{elem2edge}(i, \text{signedge}, NE, a, b) = \text{BDMrightorder}(i, \text{signedge}, NT, a, b)\]
localEdge = [2 3; 3 1; 1 2];
i1 = \text{localEdge}(i,1);
i2 = \text{localEdge}(i,2);
ai1 = \text{a}((i1-1)*NT+(1:NT)');
ai2 = \text{a}((i2-1)*NT+(1:NT)');
b11 = \text{b}((i1-1)*NT+(1:NT)');
b12 = \text{b}((i2-1)*NT+(1:NT)');
\end{verbatim}
6.2. Assembling the force $f$ term. Since $\nabla \cdot \sigma_h \in P_0$, we only need the numerical integration of the $f$ term to be accurate as if $f$ is a constant on each element. Thus, the term related to $-(f, 1)_K$ can be computed by a one-point quadrature rule:

$$- \int_K f dx \approx -f(x_{\text{mid}}, y_{\text{mid}})|K|,$$

where $(x_{\text{mid}}, y_{\text{mid}})$ are the coordinates of the gravity center of the element $K$. 

6.3. Generating boundary data. On the boundary, we need sign_D and sign_N to denote the difference between orientations inherited from the local edge ordering of the element and the global edge orientations like edgesign. Since the unit out normal vector of an element on the boundary is the same as the unit out normal vector of the whole domain, sign_D and sign_N are actually the difference of normal directions of Dirichlet and Neumann edges and global out normal directions.

The MATLAB function boundary.m generates the Dirichlet and Neumann edges and their orientations with respect to the global edges.

- Lines 3-10: We generates un-sorted Dirichlet and Neumann edges and their signs.
- Lines 11-13: We generates sorted Dirichlet and Neumann edges and their signs.

We denote the set of indices of edges on the Dirichlet and Neumann boundary to be ind_D and ind_N, respectively.
To computer the term \(- (\mathbf{\tau} \cdot \mathbf{n}_D) |_{\Gamma_D}\), we need to be careful about two things. One is the difference between unit out normal vector of the domain and that of the edge, the other one is the numerical quadrature formula. In order to guarantee the convergence order of BDM\(_1\) element, we should use two-point numerical quadrature on an edge. The 2-point Gauss-Legendre quadrature of a function \(f(x)\) on interval \([a, b]\) is:

\[
\int_a^b f(x) dx \approx \frac{b - a}{2} \left( f \left( \frac{a + b}{2} \right) - \frac{b - a}{2\sqrt{3}} \right).
\]

Note that on a Dirichlet edge \(E_j = \{z_s, z_t\}, s < t\), \(\phi_{j,1} \cdot \mathbf{n}_{E_j} = \lambda_1 / |E_j|\) and \(\phi_{j,2} \cdot \mathbf{n}_{E_j} = \lambda_2 / |E_j|\). Thus, to compute \(- \int_{E_j} (\phi_{j,1} \cdot \mathbf{n}_D) g_D dx\) by formula (6.1), we need the value of \(g_D\) at quadrature points \(p_1 = \frac{n_1 + n_2}{2} - \frac{n_2 - n_1}{2\sqrt{3}}\) and \(p_2 = \frac{n_1 + n_2}{2} + \frac{n_2 - n_1}{2\sqrt{3}}\), where \(n_1\) and \(n_2\) are the coordinates of the \(z_s\) and \(z_t\), respectively. We also need to know the value of \(\lambda_s\) and \(\lambda_t\) at \(p_1\) and \(p_2\), which are

\[
\lambda_s(p_1) = \lambda_t(p_2) = \frac{1}{2} + \frac{1}{2\sqrt{3}} \quad \text{and} \quad \lambda_s(p_2) = \lambda_t(p_1) = \frac{1}{2} - \frac{1}{2\sqrt{3}}.
\]

Thus, by letting \(s(j) = \text{sign}_D(j)\), \(s(j) = 1\) or \(-1\) be the number denoting the difference between the local and global edge orientation on \(\Gamma_D\), we have

\[
- \int_{E_j} (\phi_{j,1} \cdot \mathbf{n}_D) g_D dx \approx - s(j) \left( g_D(p_1) \left( \frac{1}{4} + \frac{1}{4\sqrt{3}} \right) + g_D(p_2) \left( \frac{1}{4} - \frac{1}{4\sqrt{3}} \right) \right),
\]

\[
- \int_{E_j} (\phi_{j,2} \cdot \mathbf{n}_D) g_D dx \approx - s(j) \left( g_D(p_1) \left( \frac{1}{4} - \frac{1}{4\sqrt{3}} \right) + g_D(p_2) \left( \frac{1}{4} + \frac{1}{4\sqrt{3}} \right) \right).
\]

To handle the Neumann boundary condition, on each \(E_j = \{z_s, z_t\} \in \mathcal{E}_N\), we want to compute \(\sigma_N |_{E_j} = c_{j,1} \phi_{j,1} + c_{j,2} \phi_{j,2}\). Then \(\sigma_N \cdot \mathbf{n}_D |_{E_j} = c_{j,1} \phi_{j,1} \cdot \mathbf{n}_D + c_{j,2} \phi_{j,2} \cdot \mathbf{n}_D\). Let \(s(j) = 1\) or \(-1\) be the number denoting the difference between the local and global edge orientation on \(E_j\), we should have

\[
s(j)(c_{j,1} \lambda_s + c_{j,2} \lambda_t) / |E_j| \approx g_N.
\]
function [A,b,sol,freeDof]=rhside(node,elem,edge,bdEdge,area,A,sol,f,gD,gN)
NT = size(elem,1); NE = size(edge,1);
[Dirichlet, Neumann,sign_D,sign_N] = boundary(elem,bdEdge);
%% Assemble right hand side.
mid = (node(elem(:,1),:)+node(elem(:,2),:)+node(elem(:,3),:))/3;
b2 = accumarray((1:NT)',-f(mid).*area,[NT 1]);
%% Drichelet BC
n1= node(Dirichlet(:,1),:); n2= node(Dirichlet(:,2),:);
p1=(n1-n2)/2*sqrt(1/3)+(n2+n1)/2; p2=(n2-n1)/2 *sqrt(1/3)+(n2+n1)/2;
intgDphi1n = (gD(p1)*(1+sqrt(1/3))+gD(p2)*(1-sqrt(1/3)))/4;
intgDphi2n = (gD(p2)*(1+sqrt(1/3))+gD(p1)*(1-sqrt(1/3)))/4;
[useless, ind_D] = intersect(edge, Dirichlet, 'rows');
bb1 = accumarray(ind_D,-sign_D.*intgDphi1n,[NE 1]);
bb2 = accumarray(ind_D,-sign_D.*intgDphi2n,[NE 1]);
b1 = [bb1;bb2];
%% Neumann BC
n1= node(Neumann(:,1),:); n2= node(Neumann(:,2),:);
p1=(n1-n2)/2*sqrt(1/3)+(n2+n1)/2; p2=(n2-n1)/2 *sqrt(1/3)+(n2+n1)/2;
edgeLengthN = sqrt(sum((n1-n2).ˆ2,2));
intgNlams = edgeLengthN.*(gN(p1)*(1+sqrt(1/3))+gN(p2)*(1-sqrt(1/3)))/4;
intgNlamt = edgeLengthN.*(gN(p2)*(1+sqrt(1/3))+gN(p1)*(1-sqrt(1/3)))/4;
[useless, ind_N] = intersect(edge, Neumann, 'rows');
sol(ind_N) = 2*sign_N.*(2*intgNlams-intgNlamt);
sol(ind_N+NE) = 2*sign_N.*(2*intgNlamt-intgNlams);
%% modify right hand side
b = [b1;b2]; b = b - A *sol;
%%freeDof
isBdDof = false(2*NE+NT,1); isBdDof(ind_N)=true; isBdDof(ind_N+NE)=true;
freeDof = find(~isBdDof);
end

Let the $L^2$-projection of $g_N$ on to $P_1(E_j) = \text{span}\{\lambda_s, \lambda_t\}$ to be $g_{N,A} = d_s \lambda_s + d_t \lambda_t$:
\[
\begin{pmatrix}
(\lambda_s, \lambda_s)_{E_j} & (\lambda_s, \lambda_t)_{E_j} \\
(\lambda_s, \lambda_t)_{E_j} & (\lambda_t, \lambda_t)_{E_j}
\end{pmatrix}
\begin{pmatrix}
d_s \\
d_t
\end{pmatrix}
= 
\begin{pmatrix}
(g_N, \lambda_s)_{E_j} \\
(g_N, \lambda_t)_{E_j}
\end{pmatrix}
\]

Since $\int_E \lambda_i \lambda_j ds = |E|(1 + \delta_{ij})/6$, replace $(g_N, \lambda_s)_{E_j}$ and $(g_N, \lambda_t)_{E_j}$, by $I_{ij,s}$ and $I_{ij,t}$ using the two-point numerical quadrature \([6.1]\) as we did for $g_D$, we get
\[
d_s = (4I_{ij,s} - 2I_{j,t})/|E_j| \quad \text{and} \quad d_t = (4I_{ij,t} - 2I_{j,s})/|E_j|.
\]
So $\sigma_N|_{E_j} = c_{j,1}\phi_{j,1} + c_{j,2}\phi_{j,2}$ with $c_{j,1} = s(j)(4I_{j,s} - 2I_{j,t})$ and $c_{j,2} = s(j)(4I_{j,t} - 2I_{j,s})$, or, in the notation of \([2.9]\),
\[
x_j = s(j)(4I_{j,s} - 2I_{j,t}) \quad \text{and} \quad x_{NE+j} = s(j)(4I_{j,t} - 2I_{j,s}).
\]

With this know $\sigma_N$, then the right-hand side of the discrete problem can be easily handled as in \([1,10]\).

The followings are some comments about the function rhside.m of handling the $f$ term and boundary conditions.

- Lines 5-6: We generate terms related to $-(f,v)$.  

Listing 9. diffusionbdm.m

```matlab
function [sigma,u] = diffusionbdm(node,elem,bdEdge,elem2edge,edge,...
    signedge,exactalpha,f,gD,gN)
NT = size(elem,1); NE = size(edge,1);
sol = zeros(2*NE+NT,1);
inva = 1./exactalpha((node(elem(:,1))+node(elem(:,2))+node(elem(:,3)))/3);
[a,b,area] = gradlambda(node,elem);
A = assemblebdm(NT,NE,a,b,area,elem2edge,signedge,inva);
[A,b,sol,freeDof] = rhside(node,elem,edge,bdEdge,area,A,sol,f,gD,gN);
sol(freeDof) = A(freeDof,freeDof) \ b(freeDof);
sigma = sol(1:2*NE); u = sol(2*NE+1:end);
end
```

- Lines 8-15: We generate terms related to $-(\tau \cdot n, g_D)_{\Gamma_D}$.
- Lines 17-24: We construct the $\sigma_N$.
- Line 26: We handle the right-hand side of the matrix problem.
- Lines 28-29: We find the degrees of freedom of the matrix problem. (Those terms related to $\sigma_N$ are known, thus not free.)

### 6.4. Solving the BDM mixed problem.

The MATLAB function `diffusionbdm.m` is the principle function to solve the BDM mixed problem. With all the building blocks given before, it is relatively easy. Line 5 is used to get $\alpha^{-1}$ in each element. All the rest lines are self-explanatory.

### 7. Checking errors

The final step of a numerical test is often the convergence test by computing some norms of the error between the exact and numerical solutions obtained for different mesh sizes. In our case, we need to compute $\|\alpha^{-1/2}(\sigma - \sigma_h)\|_0$ and $\|u - u_h\|_0$.

For the both terms $\|\alpha^{-1/2}(\sigma - \sigma_h)\|_0$ and $\|u - u_h\|_0$, the direct way to compute them is summing up the errors on each element by direct computations. To lower the numerical quadrature order, the first step should be

$$
\|\alpha^{-1/2}(\sigma - \sigma_h)\|_0^2 = (\alpha^{-1}(\sigma - \sigma_h), \sigma - \sigma_h) = (\alpha^{-1}(\sigma, \sigma) - 2(\alpha^{-1}\sigma, \sigma_h) + (\alpha^{-1}\sigma_h, \sigma_h),
$$

$$
\|u - u_h\|_0^2 = (u - u_h, u - u_h) = (u, u) - 2(u, u_h) + (u_h, u_h).
$$

The terms $\alpha^{-1}(\sigma, \sigma)$ and $(u, u)$ can be computed exactly by softwares like Mathematica and Maple. In our example, $(\alpha^{-1}(\sigma, \sigma) = 1993/75 \approx 26.5733$ and $(u, u) = 18131/7500 \approx 2.41747$. The terms $(\alpha^{-1}\sigma_h, \sigma_h)$ and $(u_h, u_h)$ can also be computed exactly. For the terms $(\alpha^{-1}\sigma, \sigma_h)$ and $(u_h, u_h)$, numerical quadratures must be used.

Since this part of codes is less important, we only give brief comments about the functions we used. The MATLAB function `exactsigma.m` returns the exact value of $\sigma$. The function `sigmahBDM.m` returns the values of $\sigma_h$ at those numerical quadrature points. The function `errorBDM.m` is used to compute the errors. Here, we use a 6-point quadrature to compute them. Matrix `xw` is a $6 \times 3$ matrix, with `xw(:,i,2)` are the coordinates such that the quadrature point is $p = p_1 * (1-xw(i,1)-xw(i,2)) + p_2 * xw(i,1) + p_3 * xw(i,2)$, where...
Listing 10. exactsigma.m

```matlab
function [sigma1,sigma2] = exactsigma(p)
x = p(:,1); y = p(:,2);
sigma1=-2*x.*y.*y-1;
sigma2=(x<0).*(-2*x.*x.*y-10)+(x>=0).*(-2*x.*x.*y-1);
end
```

Listing 11. sigmahBDM.m

```matlab
function [sigmah1,sigmah2] = ... sigmahBDM(elem,node,NE,elem2edge,signedge,w1,w2,sigma)
[a,b,area] = gradlambda(node,elem); NT = size(elem,1);
lambda = [1-w1-w2,w1,w2];
sigmah1 = zeros(NT,1); sigmah2 = zeros(NT,1);
for i = 1:3
    [i1,i2,ai1,bi1,ai2,bi2] = BDMrightorder(i,signedge,NT,a,b);
i1 = double(elem2edge(:,i));
sigmah1 = sigmah1+sigma(i1).*lambda.*bi2./area/2 - sigma(i1+NE).*lambda.*bi1./area/2;
sigmah2 = sigmah2-sigma(i1).*lambda.*ai2./area/2 + sigma(i1+NE).*lambda.*ai1./area/2;
end
end
```

$p_1, p_2, \text{and } p_3$ are the coordinates of three vertices; $xw(:,3)$ is the corresponding weight for the point.

We use the code listed in Listing 13 to compute the error.

The original mesh given has mesh size $h = 1$. We refine the mesh uniformly several times, for example, using `[node,elem,bdEdge]=uniformbisect(node,elem,bdEdge),` where `uniformbisect` is a uniform refinement MATLAB function given in the package iFEM [10]. We compute the errors for different mesh sizes, and have the Table 1. We get

$$\frac{\|\alpha^{-1}(\sigma - \sigma_h)\|_0}{\|\alpha^{-1}(\sigma - \sigma_{h/2})\|_0} \approx 4 \quad \text{and} \quad \frac{\|u - u_h\|_0}{\|u - u_{h/2}\|_0} \approx 2.$$  

This result is in agreement with the a priori error estimates (2.6).

8. Related finite elements

Once we know how to programming $BDM_1$ methods, we can actually write codes for similar finite elements, with two things in mind: Writing basis functions in barycentric coordinates and correcting the local edge orientation. For more higher order elements, we may use Legendre polynomials instead of Lagrange polynomials, but the idea is similar.

8.1. Other $H(\text{div})$-conforming edge-based basis in 2D. For $RT_0$ element, as mentioned in Remark 5.2, its basis function on an edge can be written as

$$\phi^e = \lambda_s \nabla \perp \lambda_t - \lambda_t \nabla \perp \lambda_s$$
Listing 12. errorBDM.m

```matlab
function [error_sigma, error_u] = errorBDM(node, elem, NE, area, elem2edge, ...
    signedge, inva, sigma, u, ss, uu)

n1 = elem(:,1); n2 = elem(:,2); n3 = elem(:,3);
p1 = node(n1,:); p2 = node(n2,:); p3 = node(n3,:);
xw=[0.44594849091597 0.44594849091597 0.22338158967801;...
    0.44594849091597 0.10810301816807 0.22338158967801;...
    0.10810301816807 0.44594849091597 0.22338158967801;...
    0.09157621350977 0.09157621350977 0.10995174365532;...
    0.09157621350977 0.81684757298046 0.10995174365532;...
    0.81684757298046 0.09157621350977 0.10995174365532];
NP =size(xw,1); NT = size(elem,1);

uuhlocal = zeros(NT,1); % (u, uh)_K
sshlocal = zeros(NT,1); % (\alpha^{-1}\sigma, \sigma)_K
shshlocal = zeros(NT,1); % (\alpha^{-1}\sigma, \sigma)_K

for i=1:NP
    p=p1*(1-xw(i,1)-xw(i,2))+p2*xw(i,1)+p3*xw(i,2);
    uuhlocal = uuhlocal + exactu(p)*xw(i,3);
    [sigma1,sigma2] = exactsigma(p);
    [sigmah1,sigmah2] = sigmahBDM(elem,node,NE,elem2edge,xw(i,1),xw(i,2),sigma);
    sshlocal = sshlocal + (sigma1.*sigmah1+sigma2.*sigmah2)*xw(i,3);
    shshlocal = shshlocal + (sigmah1.*sigmah1+sigmah2.*sigmah2)*xw(i,3);
end

uuhlocal = u.*uuhlocal.*area; uhuhlocal = u.*u.*area;
sshlocal = inva.*sshlocal.*area; shshlocal = inva.*shshlocal.*area;
error_u = sqrt(abs(uu-2*sum(uuhlocal)+sum(uhuhlocal)));
error_sigma = sqrt(abs(ss-2*sum(sshlocal)+sum(shshlocal)));
end
```

Listing 13. checking errors

```matlab
[uu=18131/7500; ss=1993/75;
[error_sigma, error_u] = errorBDM(node, elem, NE, area, elem2edge, ...
    signedge, inva, sigma, u, ss, uu);

Table 1. Errors of σ and u on different mesh sizes

| h   | \|\alpha^{-1}(\sigma - \sigma_h)\|_0 | \|\alpha^{-1}(\sigma - \sigma_h/2)\|_0 | \|u - u_h\|_0 | \|u - u_h/2\|_0 |
|-----|-----------------|-----------------|---------|---------|
| 1   | 1.6968e-01      | 4.9712e-01      |         |         |
| 1/2 | 4.2091e-02      | 4.0314          |         | 2.4400e-01 | 2.0374  |
| 1/4 | 1.0600e-02      | 3.9707          | 1.2118e-01 | 2.0135  |
| 1/8 | 2.6630e-03      | 3.9805          | 6.0481e-02 | 2.0037  |
| 1/16| 6.6739e-04      | 3.9901          | 3.0226e-03 | 2.0009  |
| 1/32| 1.6705e-04      | 3.9952          | 1.5111e-03 | 2.0002  |
| 1/64| 4.1788e-05      | 3.9975          | 7.5555e-04 | 2.0001  |
```
Actually, it is insensitive to the staring and terminal vertices:
\[
\phi^r_t = \lambda_s \nabla^\perp \lambda_t - \lambda_t \nabla^\perp \lambda_s = \lambda_t \nabla^\perp \lambda_s - \lambda_s \nabla^\perp \lambda_t,
\]
and on adjacent elements \( \mathcal{K}^- = \{ z_{r-}, z_s, z_t \} \) and \( \mathcal{K}^+ = \{ z_{r+}, z_t, z_s \} \),
\[
\phi^r_t |_{\mathcal{K}^-} = \frac{1}{2|\mathcal{K}^-|} \left( x - x_{r-} \right) \quad \text{and} \quad \phi^r_t |_{\mathcal{K}^+} = \frac{1}{2|\mathcal{K}^+|} \left( x - x_{r+} \right).
\]
Thus lines 4-5 of \textit{BDMrightorder} is unnecessary for \( RT_0 \) elements.

For all \( RT_k \) and \( BDM_k \) spaces, basis functions are divided into two categories, edge based functions and element-based functions. The element based functions are usually easy to construct since they are only non-zero in one element, see \([4]\). So we will only discuss edge based basis functions. Both \( RT_k \) and \( BDM_k \) spaces have \( k+1 \) basis functions on each edge. We only need to functions to span \( P_k(E) \) on \( E_t \). For \( BDM_2 \) and \( RT_2 \), three edge basis functions on \( E_t \) are
\[
\phi^1_{t,1} = \lambda_s^2 \nabla^\perp \lambda_t, \quad \phi^1_{t,2} = \lambda_s \lambda_t (\nabla^\perp \lambda_t - \nabla^\perp \lambda_s), \quad \text{and} \quad \phi^1_{t,3} = -\lambda_s^2 \nabla^\perp \lambda_s.
\]
That is, we use \( \lambda_s^2, \lambda_t^2, \) and \( \lambda_s \lambda_t \) to span \( P_2 \) on \( E_t \). We can use other choices, for example, Legendre polynomials for high order spaces. With this observation, the code developed in this paper can be easily adapted to these spaces.

8.2. \textbf{Nédélec spaces in 2D}. For Nédélec spaces, the 1st and 2nd types Nédélec spaces correspond to their \( H(\text{div}) \) counterparts are \( RT \) and \( BDM \) spaces, respectively. We also only need to discuss the construction of basis functions on edges. By \([4,2]\), the zero-moment \( H(\text{curl}) \) edge basis function is
\[
\psi^\text{ned}_{t} = \lambda_s \nabla \lambda_t - \lambda_t \nabla \lambda_s.
\]
The linear \( H(\text{curl}) \) edge basis functions are
\[
\psi^\text{ned}_{t,1} = \lambda_s \nabla \lambda_t \quad \text{and} \quad \psi^\text{ned}_{t,2} = -\lambda_t \nabla \lambda_s.
\]

8.3. \textbf{\( H(\text{div}) \) and \( H(\text{curl}) \) and basis functions in 3D}. For a tetrahedral mesh, \( H(\text{div}) \) basis functions are defined on faces. Like the edge structure in this paper, we should define a face matrix. If \( F_t = \{ z_r, z_s, z_t \} \), we need ensure \( r < s < t \), and choose the normal direction such that the area of \( \{ z_r, z_s, z_t \} \) is positive. Once this is done, we can do similar things as in 2D. The \( BDM_1 \) basis functions in 3D on a face \( F_t = \{ z_r, z_s, z_t \} \) are
\[
\lambda_r \nabla \lambda_s \times \nabla \lambda_t, \quad \lambda_s \nabla \lambda_t \times \nabla \lambda_r, \quad \text{and} \quad \lambda_t \nabla \lambda_r \times \nabla \lambda_s.
\]
Its \( RT_0 \) basis function on \( F \) is
\[
\lambda_r \nabla \lambda_s \times \nabla \lambda_t + \lambda_s \nabla \lambda_t \times \nabla \lambda_r + \lambda_t \nabla \lambda_r \times \nabla \lambda_s.
\]
For three dimensionalal \( H(\text{curl}) \) space, basis functions in barycentric coordinates can be found in \([12]\).

\begin{thebibliography}{9}
\bibitem{1} J. Alibert, C. Carstensen, and S.A. Funken, \textit{Remarks around 50 lines of Matlab: short finite element implementation}, Numer. Algoritm. 20, 117-137 (1999) \[1 \[6,3\]
\bibitem{2} C. Bahriawati and C. Carstensen, \textit{Three Matlab implementations of the lowest-order Raviart-Thomas mfem with a posteriori error control}, Comput. Methods Appl. Math., Vol.5(2005), No.4, pp.333-361. \[1 \[5,3\]
\end{thebibliography}
BDM MIXED FEM IN MATLAB

[3] F. Brezzi; J. Douglas, and L. D Marini, Two families of mixed finite elements for second order elliptic problems, Numer. Math., vol 47. no. 2., 1985. 217–235.

[4] D. Boffi, F. Brezzi, and M. Fortin, Mixed Finite Element Methods and Applications, Springer Series in Computational Mathematics, 44, Springer, 2013.

[5] Z. Cai and S. Zhang, Recovery-based error estimator for interface problems: conforming linear elements, SIAM J. Numer. Anal., Vol. 47, No. 3, pp. 2132–2156, 2009.

[6] Z. Cai and S. Zhang, Recovery-based error estimator for interface problems: mixed and nonconforming elements, SIAM J. Numer. Anal., Vol. 48, No. 1, pp. 30–52, 2010.

[7] Z. Cai and S. Zhang, Flux recovery and a posteriori error estimators: Conforming elements for scalar elliptic equations, SIAM J. Numer. Anal., Vol. 50, No. 1, pp. 151-170, 2012.

[8] Z. Cai and S. Zhang, Robust equilibrated residual error estimator for diffusion problems: Conforming elements, SIAM J. Numer. Anal., Vol. 50, No. 1, pp. 578–602, 2010.

[9] L. Chen, iFEM: an integrated finite element methods package in MATLAB, Technical Report, University of California at Irvine. 2009.

[10] S. Funken, D. Praetorius, and P. Wissgott, Efficient implementation of adaptive P1-FEM in MATLAB, Comput. Methods Appl. Math. 11, 460-490 (2011).

[11] J. Gopalakrishnan, L. F. Demkowicz and L. E. Garcia-Castillo, Nédélec spaces in affine coordinates, Computers and Mathematics with Applications, Volume 49, Issue 7-8, pp. 1285-1294, 2005.

[12] P. A. Raviart and I. M. Thomas, A mixed finite element method for second order elliptic problems, Lect. Notes Math. 606, Springer-Verlag, Berlin and New York (1977), 292-315.

[13] P. Solin, K. Segeth, and I. Dolezel, Higher-Order Finite Element Methods, CRC Press, 2003.

[14] L. N. Trefethen, Spectral Methods in MATLAB, SIAM, Philadelphia, 2000.