Introduction

In this project I focused on implementation of mixed Poisson problem with $\mathcal{RT}_0-\mathcal{P}_0$ elements. My work is based on the paper "Three MATLAB implementations of the lowest-order Raviart-Thomas MFEM with a posteriori error control" by C. Bahriawati and C. Carstensen(`http://www2.mathematik.hu-berlin.de/~cc/cc_homepage/download/2005-BC_CC-Three_MATLAB_Implementations_Lowest-Order_Raviart-Thomas_MFEM.pdf`).
Since my goal was to understand the way one can build $\mathcal{RT}_0$ elements, I tried to make the code as simple to read as possible, ignoring performance issues (e.g., there are a lot of computations in loops and all matrices are fully stored). Also, I don't think that such code can be used in real world, since a lot of computations are performed by hand (similar to 50 lines Matlab implementation of regular Poisson problem), but still it was really useful in understanding of how H_{div} elements work.

Mixed Poisson problem with RT0-P0 elements

1. **Description of problem**
   Let Ω be a bounded domain in the plane with polygonal boundary $\Gamma = \Gamma_D \cup \Gamma_N$ split into two regions: Dirichlet and Neumann boundary respectively. Given $f \in L^2(\Omega), g \in L^2(\Gamma_N)$ and $U_D \in H^1(\Omega \cap C(\bar{\Omega}))$ such that

$$-\Delta u = f \qquad \text{in } \Omega \tag{1}$$
$$u = u_D \qquad \text{on } \Gamma_D \tag{2}$$
$$\nabla u \cdot \nu = g \qquad \text{on } \Gamma_N \tag{3}$$

Split equation (1) into two parts by:

$$-\operatorname{div} p = f \text{ and } p = \nabla u, \qquad \text{in } \Omega$$

for unknown $u \in H_1(\Omega)$ and $p \in L^2(\Omega)$ with $\operatorname{div} p \in L^2(\Omega)$.
Now the weak formulation reads:

$$\int_\Omega p \cdot q dx + \int_\Omega u \operatorname{div} q dx = \int_{\Gamma_D} u_D q \cdot \nu \qquad \forall q \in H_{0,N}(\operatorname{div}, \Omega) \tag{4}$$
$$\int_\Omega v \operatorname{div} p dx = -\int_\Omega v f dx \qquad \forall v \in L^2(\Omega) \tag{5}$$

where

$$H_{0,N}(\operatorname{div}, \Omega) := \{q \in H(\operatorname{div}, \Omega) : q \cdot \nu = 0 \text{ on } \Gamma_N\}$$
$$H_{g,N}(\operatorname{div}, \Omega) := \{q \in H(\operatorname{div}, \Omega) : q \cdot \nu = g \text{ on } \Gamma_N\}$$

For the discretization we choose $\mathcal{RT}_0$ and $\mathcal{P}_0$ spaces for flux and displacement respectively:

$$\begin{aligned}\mathcal{RT}_0(\mathcal{T}) := \{q \in L^2(T) : \forall T \in \mathcal{T}\ \exists a \in \mathbb{R}^2\ \exists b \in \mathbb{R}\ \forall x \in T, q(x) = a + bx \\ \text{and } \forall E \in \Sigma_\Omega, \lceil q \rceil_E \cdot \nu_E = 0\} \\ \mathcal{P}_0(\mathcal{T}) := \{v \in L^2(\Omega) : \forall T \in \mathcal{T}\ v|_T \in \mathcal{P}_0(T)\}\end{aligned}$$

where $\mathcal{T}$ is a regular triangulation, Σ is the set of all interior edges and $\lceil q \rceil_E$ denotes the jump of q across the edge E shared by the two neighboring elements.
With the Σ_N -piecewise constant approximation g_h of g, $g_h|_E = \int_E g ds/|E|$ for each $E \in \Sigma_N$ of length $|E|$, the discrete spaces read

$$\begin{aligned}M_{h,g} := \{q_h \in \mathcal{RT}_0(\mathcal{T}) : q_h \cdot \nu = g_h \text{ on } \Gamma_N\} \\ M_{h,0} := \mathcal{RT}_0(\mathcal{T}) \cap H_{0,N}(\operatorname{div}, \Omega) \\ L_h := \mathcal{P}_0(\mathcal{T})\end{aligned}$$

The discrete problem reads: seek $(u_h, p_h) \in L_h \times M_{h,g}$ with

$$\int_\Omega p_h \cdot q_h dx + \int_\Omega u_h \operatorname{div} q_h dx = \int_{\Gamma_D} u_D q_h \cdot \nu \qquad \forall q_h \in M_{h,0} \tag{6}$$
$$\int_\Omega v_h \operatorname{div} p_h dx = -\int_\Omega v_h f dx \qquad \forall v_h \in L_h \tag{7}$$

Let $M_{h,0} = span\{\psi_1, ..., \psi_M\}$, with respect to this basis (possibly in a different order of the indices), we have the components $x_\psi = (x_1, ..., x_N)$ of $p_h = \sum_{k=1}^{M} x_k \psi_k \in M_{h,g}$ and the components $x_u = (x_{M+1}, ..., x_{M+L}) of u_h|_{T_l} = x_{M+l}$ for $l = 1, ..., L$ and for an enumeration $\mathcal{T} = \mathcal{T}_1, ..., \mathcal{T}_L$ of the $L = card(\mathcal{T})$ elements. Then the problem transforms into a linear system for the unknown $(x_1, ..., x_M$ and $(x_{M+1}, ..., x_{M+L}$:

$$\sum_{k=1}^{M} \int_\Omega \psi_j \cdot \psi_k dx + \sum_{l=1}^{L} x_{N+l} \int_\Omega \operatorname{div} \psi_j dx = \int_{\Gamma_D} u_D \psi_j \cdot \nu - \sum_{m=M+1}^{N} g_h|_{E_m} \int_\Omega \psi_j \cdot \psi_m dx \tag{8}$$

$$\sum_{k=1}^{M} x_k \int_\Omega \operatorname{div} \psi_k dx = -\int_\Omega f dx - \sum_{m=M+1}^{N} g_h|_{E_m} \int_{T_l} \operatorname{div} \psi_k dx \tag{9}$$

or, equivalently,

$$\begin{pmatrix} B & C \\ C^T & 0 \end{pmatrix} \begin{pmatrix} x_\psi \\ x_u \end{pmatrix} = \begin{pmatrix} b_D \\ b_f \end{pmatrix}$$

2. **Domain and mesh**
We consider polygon domain, covered by regular triangulation $\mathcal{T}$, i.e. set of closed triangles $T = conv(a, b, c)$, with vertices a, b, c. Each edge in the set Σ is $E = conv(a, b)$.The boundary of the domain is given by $\Sigma = \Sigma_N \cup \Sigma_D$. In the code we use nodes $z_1, ..., z_n$ with coordinates in $\mathbb{R}^2$, that are stored in `coordinate.dat`. The element $T_m = conv(z_i, z_j, z_k)$ (with counterclockwise enumeration) is described by the global labels i, j, k, stored in row m of `element.dat`.

3. **Additional information about edges**
The degrees of freedom of flux variable are edge-oriented, therefore we need to perform edge enumeration and connect all edges with geometric information of the triangulation.
In order to do this we will need three additional matrices:

- **matrix `nodes2element`** is a quadratic matrix of dimension n (=number of nodes), s.t.:

$$node2element(k, l) = \begin{cases} j, & \text{if } (k, l) \text{ describe an edge of element number } j \\ 0, & \text{otherwise} \end{cases}$$

  Here the orientation of an edge is important. Suppose $E = conv(x_k, x_l) \in T_i$ and $E = conv(x_l, x_k) \in T_j$, then `nodes2element(k,l)=i` and `nodes2element(l,k)=j`.
  `Matlab` implementation:

```
nodes2element=zeros(length(coordinates),length(coordinates));
% nodes2element is a matrix that describes elements in terms of edges
% node2element(i,j)=number of element, if (i,j)-edge belongs to this
% element
% node2element(i,j)=0 otherwise

% each row in 'element' consists of its vertices,
% thus for each i-th element, its edges are :
% {element(i,1),element(i,2)}
% {element(i,2),element(i,3)}
% {element(i,3),element(i,1)}
% note that {i,j} and {j,i} describe the same edge up to orientation
% and therefore these edges belong to adjacent triangles

for k=1:length(element)
        nodes2element(element(k,1),element(k,2))=k;
        nodes2element(element(k,2),element(k,3))=k;
        nodes2element(element(k,3),element(k,1))=k;
end
```

- **matrix `nodes2edge`** is a quadratic matrix of dimension n (=number of nodes), s.t.:

$$node2edge(k, l) = \begin{cases} j, & \text{if nodes } x_k, x_l \text{ describe an edge number } j \\ 0, & \text{otherwise} \end{cases}$$

  In this case, we ignore orientation of edges, thus this matrix is symmetric. Also here we introduce variable `noedges`, which returns total number of edges.
  `Matlab` implementation:

```
nodes2edge=zeros(length(coordinates),length(coordinates));
% nodes2edge is a matrix that describes edges in terms of nodes
% node2eledge(i,j)=number of edge, if conv(i,j) = edge
% node2eledge(i,j)=0 otherwise

% we don't consider orientation of the edges, thus
% nodes2edge is symmetric matrix
% node2element(i,j)=k and node2element(j,i)=l for adjacent
% triangles number k and l, so we need only upper triangular
% or lower triangular part of this matrix

B=nodes2element+nodes2element';
[I,J]=find(triu(B));

for k=1:length(I)
        nodes2edge(I(k,1),J(k,1))=k;
end
nodes2edge=nodes2edge+nodes2edge';
noedges=size(I,1);
```

- **matrix `nodes2element`** is a matrix of size `noedges`×4 , s.t.its jth row consists of k, l, m, n, where k and l are initial and end nodes of edge j and m, n are labels of triangles that share this edge (with positive and negative orientations respectively), so

$$edge2element(j,[3,4]) = \begin{cases} (p,q), & \text{if elements } p \text{ and } q \text{ share edge } j \\ (p,0), & \text{if edge } j \text{ is a boundary edge of element } p \end{cases}$$

  `Matlab` implementation:

```
edge2element=zeros(noedges,4);
% nodes2element is a matrix that describes edges in terms of nodes and
% triangles
% each row of the matrix corresponds to one edge and
% first entry of this row is the initial node of edge
% second entry is the end node of the edge
% third entry is number the triangle that has this edge wrt positive
% orientation
% fourth entry is number the triangle that has this edge wrt negative
% orientation


for m=1:size(element)
        for k=1:3
                initial_node=element(m,k);
                        if k<3
                                end_node=element(m,k+1);
                        else
                                end_node=element(m,1);
                        end
                p=nodes2edge(element(m,k),element(m,rem(k,3)+1));
                if edge2element(p,1)==0
                        edge2element(p,:)=[initial_node end_node ...
                        nodes2element(initial_node,end_node) ...
                        nodes2element(end_node,initial_node)];
                end
        end
end
```

These three matrices are gathered by function `edge.m`:

```
function [nodes2element,nodes2edge,noedges,edge2element]=...
edge(element,coordinate);
```

4. **Stiffness matrix**
In order to assemble the matrix, we need to know the basis functions of Raviart-Thomas space. Given an edge $E \in \Sigma$, there are either two elements T_+ and T_- in $\mathcal{T}$ with the joint edge $E = \partial T_+ \cap \partial T_-$ or there is exactly one element T_+ with $E \in \partial T_+$. Then if $T_\pm = conv(E \cup \{P_\pm\})$ for the vertex $P_\pm$ opposite to E of $T_\pm$ define the **edge basis function**

$$\psi_E(x) = \begin{cases} \pm\frac{|E|}{2|T_\pm|}(x - P_\pm), \text{for } x \in T_\pm \\ 0, \text{ elsewhere} \end{cases}$$

here $|T|= \dfrac{1}{2}\det(P_2 - P_1, P_3 - P1)$ for $T = conv(P_1, P_2, P_3)$ and $|E|$ is length of E.
With such construction, one can prove the Lemma below.
Lemma. There hold

(a)
$$\psi_E \cdot \nu_E = \begin{cases} 0, \text{ along } (\cup\Sigma) \setminus E \\ 1, \text{ along } E \end{cases}$$

(b) $\psi_E \in H(\text{div}, \Omega)$

(c) $\{\psi_E\}|_{E\in\Sigma}$ is a basis of $\mathcal{RT}_0(\mathcal{T})$

(d)
$$\text{div}\,\psi_E = \begin{cases} \pm\frac{|E|}{|T_\pm|}, \text{ on } T_\pm \\ 0, \text{ elsewhere} \end{cases}$$

Also, local definition of the basis function is: given E_1, E_2, E_3 - the edges of a triangle T opposite to its vertices P_1, P_2, P_3, respectively, let ν_{E_j} denote the unit normal vector of E_j chosen with a global fixed orientation while ν_j denotes the outer unit normal of T along E_j. Define

$$\psi_{E_j}(x) = \sigma_j \frac{|E_j|}{2|T|}(x - P_j), \text{ for } j = 1, 2, 3 \text{ and } x \in T$$

where $\sigma_j = \nu_j \cdot \nu_{E_j}$ is $+1$ if ν_{E_j} points outward and otherwise -1.

- **local matrices**
On each triangle local matrices are $B_T(3 \times 3)$ and $C_T(3 \times 3)$ with

$$(B_T)_{j,k} := \int_T \psi_{E_j} \cdot \psi_{E_k}, \text{ for } j, k = 1, 2, 3$$
$$C_T = diag\{\int_T \text{div}\,\psi_{E_1} dx, \int_T \text{div}\,\psi_{E_2} dx, \int_T \text{div}\,\psi_{E_3} dx\}$$

Then, calculating all elements and using the Lemma, we get that on each triangle (without orientation of the edges)

$$B_T := \frac{1}{48|T|} C_T^T N^T M N C_T$$
$$C_T^T = C_T = diag\{|E_1|, |E_2|, |E_3|\}$$

where

$$M = \begin{pmatrix} 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{R}^{6\times 6}, \text{ and} N = \begin{pmatrix} 0 & P_1 - P_2 & P_1 - P_3 \\ P_2 - P_1 & 0 & P_2 - P_3 \\ P_3 - P_1 & P_3 - P_2 & 0 \end{pmatrix} \in \mathbb{R}^{6\times 3}$$

`Matlab` implementation:

```
function [stB,stC]=stimaB_my(coord);
% local stiffness matrix on one triangle

N=coord(:)*ones(1,3)-ones(3,1)*coord(:)';
```

```
stC=diag([norm(N([5,6],2)),norm(N([1,2],3)),norm(N([1,2],2))]);
M=diag(ones(6,1));
for k=1:4
        M(k,k+2)=1;
end
for k=1:2
        M(k,k+4)=1;
end
M=M+M';
absT=0.5*det([1,1,1;coord]);
stB=stC*N'*M*N*C/(48*absT);
```

- **global matrix**
  Global stiffness matrix A consists of matrices B, C, C^T, that can be constructed using local $B(T), C(T)$ with correct global signs of edges.
  So we need to take all entries $(B(T))_{j,k}, (C(T))_{l,m}$, multiply them by signs of involved basis functions (= sign of edges) and put into correct place in B. Consider $T_k \in \mathcal{T}$ with edges $E_{k_1}, E_{k_2}, E_{k_3}$ and let $\sigma_{k_1}, \sigma_{k_2}, \sigma_{k_3}$ be the corresponding global signs. Then we can define a part of global matrix B as:

$$B_{gl}(T_k) = diag\{\sigma_{k_1}, \sigma_{k_2}, \sigma_{k_3}\} B_{loc}(T_k) diag\{\sigma_{k_1}, \sigma_{k_2}, \sigma_{k_3}\}$$

  And similar formula holds for matrix C.
  For each triangle, we can get the global numbers of edges using `nodes2edge`, then for each edge we can get labels of elements sharing this edge using `edge2element`. Suppose we consider element T_k, then the label of its first edge E_{K_1} is `j=nodes2edge(element(k,1),element(k,2))`, and it has negative sign if `k=edge2element(j,4)` and positive if `k=edge2element(j,4)`.
  The following code performs assembling of global stiffness matrix:

```
% Assemble matrices B and C  from local matrices stB and stC

B=zeros(noedges,noedges);
C=zeros(noedges,size(element,1));
for j=1:size(element,1)
         coord=coordinate(element(j,:),:)';
         % edges of each element:
         I(1,1)=nodes2edge(element(j,2),element(j,3));
         I(2,1)=nodes2edge(element(j,3),element(j,1));
         I(3,1)=nodes2edge(element(j,1),element(j,2));
         % signs of edges:
         signum=ones(1,3);
         signum(find(j==edge2element(I,4)))=-1;
         % assembling global matrices:
         [stB,stC]=stimaB(coord);
         B(I,I)=B(I,I)+diag(signum)*stB*diag(signum);
         C(I,j)=diag(signum)*diag(stC);
end

% Global stiffness matrix A
A=zeros(noedges+size(element,1),noedges+size(element,1));
A=[B,C;C',zeros(size(C,2),size(C,2))];
```

5. **Right-hand side vector**
   For simplicity, assume that we have only homogeneous Dirichlet boundary conditions, then the RHS vector is:

$$b = \begin{pmatrix} b_D \\ b_f \end{pmatrix}$$

   where b_D and b_f correspond to the Dirichlet BCs and external force respectively.

- **computing b_D**
  Our goal is to compute the entries of the form $\int_{\Gamma_D} u_D \psi_j \cdot \nu ds$. On each edge we have just one basis

function with normal component $\psi \cdot \nu = 1$ (the rest are equal zero), thus

$$\int_{\Gamma_D} u_D \psi_j \cdot \nu ds = \int_E u_D ds \simeq u_D(x_M, y_M)|E| = (b_D)_i$$

where (x_M, y_M) is the midpoint of the edge E
This part is implemented in the following way:

```
% Boundary conditions
% each row of matrix exterior contains initial and end nodes
% of boundary edges
exterior=edge2element(find (edge2element(:,4)==0),[1 2]);
for k = 1:size(exterior ,1)
        for k = 1:size(exterior ,1)
                b(nodes2edge(exterior(k,1),exterior(k,2)))=...
                norm(coordinate(exterior(k,1),:)-...
                coordinate(exterior(k,2),:))*u_D(sum(coordinate(exterior(k,:),:)
        end
end
```

- **computing** b_f
  Entries of b_f contain $-\int_{T_l} f dx$, which can be approximated by one point quadrature rule at center of gravity of each element z_{T_l}:

  $$(b_f)_i = -|T| f(z_{T_l})$$

  which is given in the code as:

```
% Volume force
b = zeros(noedges+size(element ,1),1);
for j = 1:size(element ,1)
        absT=0.5*det([1,1,1; coordinate(element(j,:),:)']);
        b(noedges+j)=-absT*f(sum(coordinate(element(j,:),:))/3);
end
```

Finally, we solve the system by `x=A/b;`

6. **Results**
Consider solving $-\Delta u = f$ in Ω with homogeneous boundary conditions $u_D = 0$ on $\partial\Omega$, where Ω is a unit square split into eight triangles:

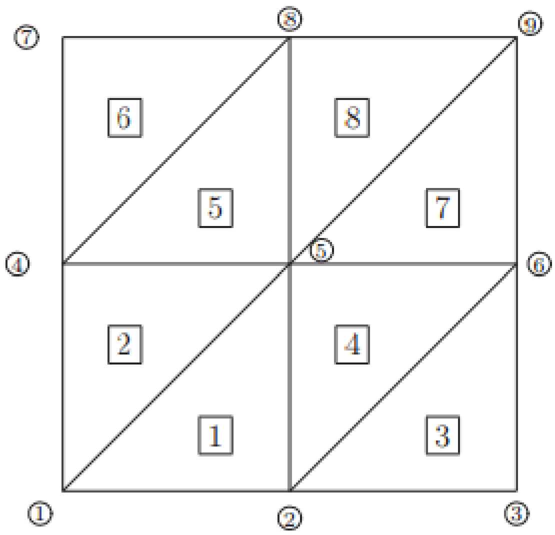
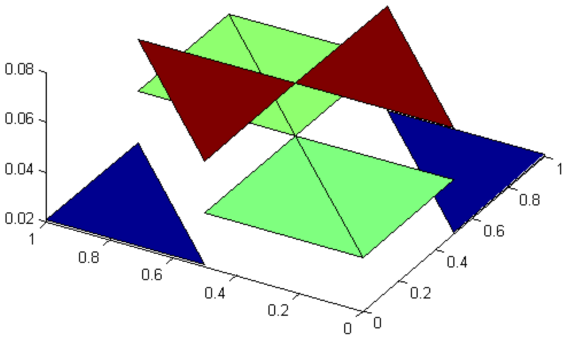


Figure 1: triangulation of the domain

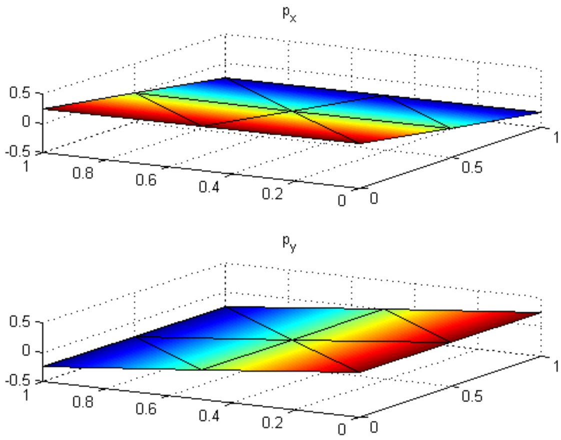
Implementation of functions f, u_D:

```
%f.m
function volumeforce=f(x);
volumeforce=ones(size(x,1),1);
%u_D.m
function dir=u_D(x);
dir=zeros(size(x,1),1);
```

The discrete solution is given below:



(a) discrete displacement



(b) x and y components of discrete flux