MATLAB Programming for Virtual Element Methods

1 二维网格剖分与预处理

1.1 网格的数据结构及图示

1.1.1 基本数据结构

我们采用 Chen Long 有限元工具箱 iFEM 中给出的数据结构, 用 node 表示节点坐标, elem 表示单元的连通性, 即单元顶点编号. 例如考虑下图中 L 形区域的一个简单剖分 (对一般的多角形剖分类似). 可参考网页说明:

https://www.math.uci.edu/~chenlong/ifemdoc/mesh/meshbasicdoc.html

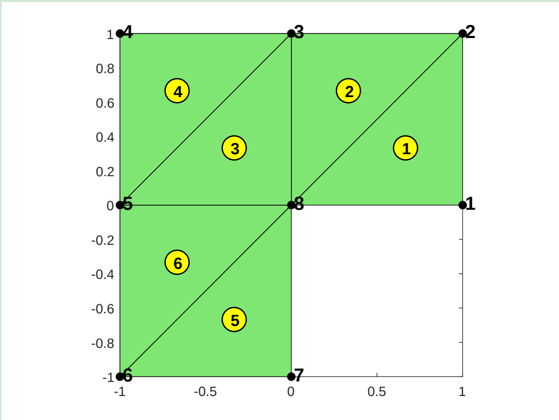


图 1. L 形区域的剖分

1. 数组 node: 节点坐标

   在编程中我们需要每个节点的坐标, 用 node 记录, 它是两列的一个矩阵, 第一列表示各节点的横坐标, 第二列表示各节点的纵坐标, 行的索引对应节点标号. 图中给出的顶点坐标信息如下

   8x2 double

	1	2
1	1	0
2	1	1
3	0	1
4	-1	1
5	-1	0
6	-1	-1
7	0	-1
8	0	0

   这里左侧的序号对应节点的整体编号.

2. 数组 elem: 连通性 (局部整体对应)

   数组 elem 给出每个三角形的顶点编号, 它给出的是单元的连通性信息, 每行对应一个单元.

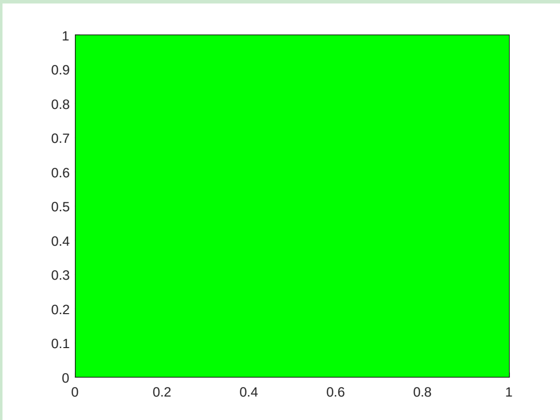
6x3 double

	1	2	3
1	1	2	8
2	3	8	2
3	8	3	5
4	4	5	3
5	7	8	6
6	5	6	8

图中第一列表示所有三角形的第一个点的编号, 第二列表示第二个点的编号, 依此类推. 注意三角形顶点的顺序符合逆时针定向. elem 是有限元编程装配过程中的局部整体对应。

1.1.2 补片函数 patch

我们要画出每个单元, 对三角形单元 MATLAB 有专门的命令, 对多边形我们需要采用补片函数 patch. 实际上三角剖分采用的也是 patch, 为此以下只考虑 patch. 以下只考虑二维区域剖分的图示, 命名为 showmesh.m. 一个简单的例子如下图



可如下编程

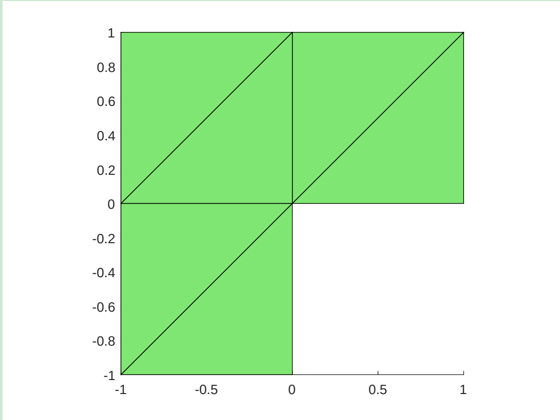
```
node = [0 0; 1 0; 1 1; 0 1];
elem = [1 2 3 4];
patch('Faces',node,'Vertices',elem,'FaceColor','g')
```

对多个相同类型的单元, 如下

```
function showmesh(node,elem)
h = patch('Faces',elem, 'Vertices', node);
set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
axis equal; axis tight;
```

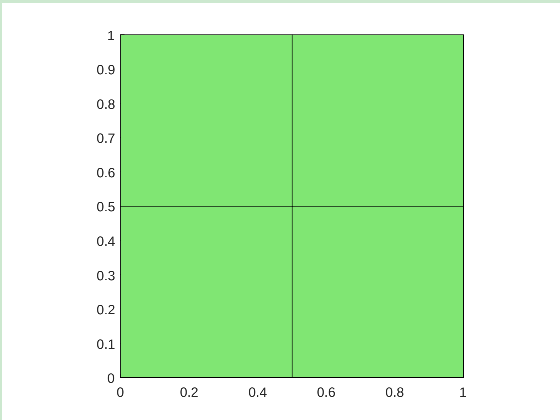
例 1.1 (三角剖分) 对前面的梯形区域可如下调用 showmesh 函数

```
node = [1,0; 1,1; 0,1; -1,1; -1,0; -1,-1; 0,-1; 0,0];
elem = [1,2,8; 3,8,2; 8,3,5; 4,5,3; 7,8,6; 5,6,8];
showmesh(node,elem);
```



例 1.2 (四边形剖分)　对矩形区域的四边形剖分可如下调用 showmesh 函数

```
[X,Y] = ndgrid(0:0.5:1,0:0.5:1);
node = [X(:), Y(:)];
elem = [1 2 5 4; 2 3 6 5; 4 5 8 7; 5 6 9 8];
showmesh(node,elem)
```



1.1.3　操作 cell 数组的 cellfun 函数

对含有不同多角形剖分的区域, 因每个单元顶点数不同, elem 一般以 cell 数组存储. 为了使用 patch 画图 (不用循环语句逐个), 我们需要将 elem 的每个 cell 填充成相同维数的向量, 填充的值为 NaN, 它不会起作用. 我们先介绍 MATLAB 中操作 cell 数组的函数 cellfun. 例如, 考虑下面的例子

例 1.3　计算 cell 数组中元素的平均值和维数

```
C = {1:10, [2; 4; 6], []};
averages = cellfun(@mean, C)
[nrows, ncols] = cellfun(@size, C)

% 结果为 averages =  5.5000    4.0000        NaN
%         nrows =  1  3  0,    ncols = 10  1  0
```

cellfun 的直接输出规定为数值数组, 如果希望输出的可以是其他类型的元素, 那么需要指定 UniformOutput 为 false, 例如

例 1.4　对字符进行缩写

```
days = {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'};
abbrev = cellfun(@(x) x(1:3), days, 'UniformOutput', false)
```

正因为此时输出类型可以任意, MATLAB 默认仍保存为 cell 类型. 上面的结果为

```
abbrev =
  1×5 cell array
    {'Mon'}    {'Tue'}    {'Wed'}    {'Thu'}    {'Fri'}
```

1.1.4 showmesh 函数的建立

现在考虑下图所示的剖分

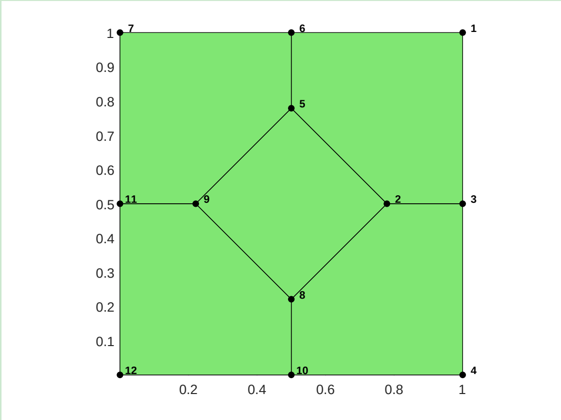


图 2. 多角形网格

相关的网格数据保持在 meshex1.mat 中. 程序如下

```
load('meshex1.mat'); % node, elem

max_n_vertices = max(cellfun(@length, elem));
% function to pad the vacancies ( 横向拼接 )
padding_func = @(vertex_ind) [vertex_ind,...
      NaN(1,max_n_vertices-length(vertex_ind))];
tpad = cellfun(padding_func, elem, 'UniformOutput', false);
tpad = vertcat(tpad{:});
h = patch('Faces', tpad,'Vertices', node);
set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
axis equal; axis tight;
```

最终给出的 showmesh 函数如下

CODE 1. showmesh.m (2D 网格画图)

```
function showmesh(node,elem)
%Showmesh displays a mesh in 2-D.

if ¬iscell(elem)
    h = patch('Faces', elem, 'Vertices', node);

```

```
else
    max_n_vertices = max(cellfun(@length, elem));
    padding_func = @(vertex_ind) [vertex_ind,...
        NaN(1,max_n_vertices-length(vertex_ind))];  % function to pad the ...
            vacancies
    tpad = cellfun(padding_func, elem, 'UniformOutput', false);
    tpad = vertcat(tpad{:});
    h = patch('Faces', tpad, 'Vertices', node);
end

set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
axis equal; axis tight;
```

1.1.5 showsolution 函数的建立

程序如下

```
function showsolution(node,elem,u)
%Showsolution displays the solution corresponding to a mesh given by ...
    [node,elem] in 2-D.

data = [node,u];
patch('Faces', elem,...
    'Vertices', data,...
    'FaceColor', 'interp',...
    'CData', u / max(abs(u)) );
axis('square');
sh = 0.05;
xlim([min(node(:,1)) - sh, max(node(:,1)) + sh])
ylim([min(node(:,2)) - sh, max(node(:,2)) + sh])
zlim([min(u) - sh, max(u) + sh])
xlabel('x'); ylabel('y'); zlabel('u');

view(3); grid on; % view(150,30);
```

我们来说明一下.

- patch 也可以画空间中的直面, 此时只要把 'Vertices' 处的数据换为三维的顶点坐标.
- 对解 u, 显然 `data = [node,u]` 就是我们画图需要的三维点坐标.
- patch 后的

  ```
  'FaceColor', 'interp', 'CData', u / max(abs(u))
  ```

  是三维图形的颜色, 它根据 'CData' 数据进行插值获得 (不对颜色进行设置, 默认为黑色). 也可以改为二维的

  ```
  set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
  ```

  此时显示的只是一种颜色, 对解通常希望有颜色的变化.

- 需要注意的是, 即便是三维数据, 若不加最后的

  ```
  view(3); grid on; %view(150,30);
  ```

  给出的也是二维图 (投影, 即二维剖分图).

当然上面的程序也可改为适合多角形剖分的, 如下

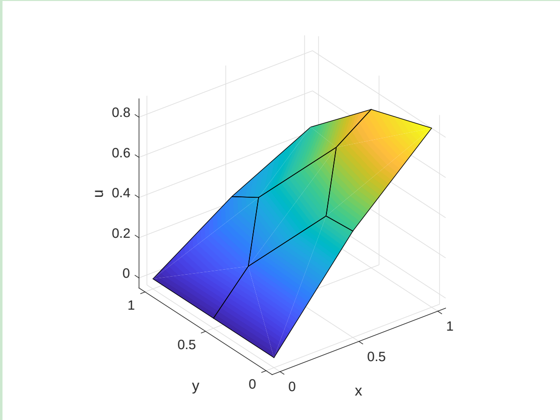
CODE 2. showsolution.m

```
function showsolution(node,elem,u)
%Showsolution displays the solution corresponding to a mesh given by ...
    [node,elem] in 2-D.

data = [node,u];
if ¬iscell(elem)
    patch('Faces', elem,...
        'Vertices', data,...
        'FaceColor', 'interp',...
        'CData', u / max(abs(u)) );
else
    max_n_vertices = max(cellfun(@length, elem));
    padding_func = @(vertex_ind) [vertex_ind,...
        NaN(1,max_n_vertices-length(vertex_ind))]; % function to pad the ...
            vacancies
    tpad = cellfun(padding_func, elem, 'UniformOutput', false);
    tpad = vertcat(tpad{:});
    patch('Faces', tpad,...
        'Vertices', data,...
        'FaceColor', 'interp',...
        'CData', u / max(abs(u)) );
end
axis('square');
sh = 0.05;
xlim([min(node(:,1)) - sh, max(node(:,1)) + sh])
ylim([min(node(:,2)) - sh, max(node(:,2)) + sh])
zlim([min(u) - sh, max(u) + sh])
xlabel('x'); ylabel('y'); zlabel('u');

view(3); grid on; % view(150,30);
```

例 1.5 例如, 可如下画 $u(x,y)=\sin x\cos y$ 的图像

```
load('meshex1.mat');
x = node(:,1); y = node(:,2); u = sin(x).*cos(y);
showsolution(node,elem,u);
```

结果如下



注 1.1 可以看到, showmesh 与 showsolution 唯一不同的地方就是添加

```
view(3); grid on; %view(150,30);
```

三维网格的单元是多面体, 我们一般也是逐个面画图, 此时可在 showmesh 下添加上面的语句. 修改后的 showmesh 如下 (画图时三维的 elem 要存储为面)

CODE 3. showmesh.m

```
function showmesh(node,elem)
%Showmesh displays a mesh in 2-D and 3-D.

if ¬iscell(elem)
    h = patch('Faces', elem, 'Vertices', node);
else
    max_n_vertices = max(cellfun(@length, elem));
    padding_func = @(vertex_ind) [vertex_ind,...
        NaN(1,max_n_vertices-length(vertex_ind))];  % function to pad the ... vacancies
    tpad = cellfun(padding_func, elem, 'UniformOutput', false);
    tpad = vertcat(tpad{:});
    h = patch('Faces', tpad, 'Vertices', node);
end

dim = size(node,2);
if dim==3
    view(3); set(h,'FaceAlpha',0.4); % 透明度
end

set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
axis equal; axis tight;
```

显然, 用该函数也可画解的图像

```
load('meshex1.mat');
x = node(:,1); y = node(:,2); u = sin(x).*cos(y);
% show the solution by using showmesh
data = [node,u];
showmesh(data,elem);
```

结果一致, 只不过图像的颜色是单一的罢了. 为了方便, 我们单独建立了 showsolution 函数.

1.2 Generation of the polygonal meshes

The MATLAB tool, PolyMesher, is applied to generate the polygonal meshes, and to obtain the basic data structure: `node` and `elem`. All M-files with slight changes are placed in the following folder directory: ~/progam/PolyMesherModified. One only needs to run the mesh generation function meshfun.m in the tool folder, which is displayed as follows:

CODE 4. meshfun.m

```
%Meshfun: genearte basic data structure (node, elem) representing meshes

% ----------- Useage -------------
% Preserved as meshdata.mat
% load('meshdata.mat') to load basic data structure: node, elem
% ----------------------------------

clc;clear;close all
addpath(genpath(pwd)); % Include all folders under the current path

% -------- Choices of the domain --------
% Options: Rectangle_Domain, Circle_Domain, Upper_Circle_Domain,
% Upper_Circle_Circle_Domain, Rectangle_Circle_Domain, Circle_Circle_Domain
Domain = @Rectangle_Domain;
NElem = 5; MaxIter = 300;
[node,elem]= PolyMesher(Domain,NElem,MaxIter);

% -------- Plot the mesh --------
showmesh(node,elem);
findnode(node);

% -------- Save the mesh data --------
save meshdata node elem
```

1.3 网格的标记

1.3.1 节点标记

可如下给出图 2 中的节点编号

```
load('meshex1.mat');
showmesh(node,elem);
findnode(node);
```

函数文件如下

CODE 5. findnode.m

```
function findnode(node,range)
%Findnode highlights nodes in certain range.

hold on
dotColor = 'k.';
```

```
if nargin==1
    range = (1:size(node,1))';
end
plot(node(range,1),node(range,2),dotColor, 'MarkerSize', 15);
shift = [0.015 0.015];
text(node(range,1)+shift(1),node(range,2)+shift(2),int2str(range), ...
     'FontSize',8,'FontWeight','bold'); % show index number
hold off
```

注 1.2 当然也可改为适用于三维情形, 这里略, 见 GitHub 上传程序 (tool 文件夹内).

1.3.2 单元标记

现在标记单元. 我们需要给出单元的重心, 从而标记序号. 重心的计算后面说明.

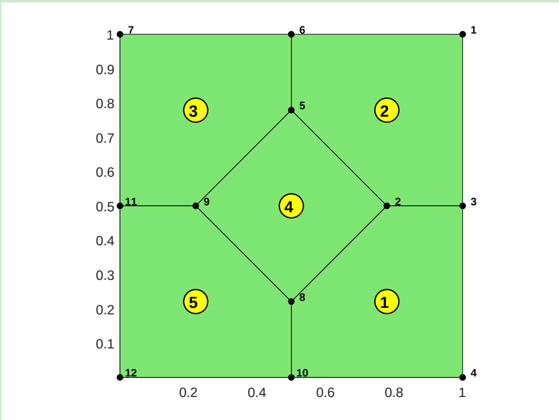


图 3. Polygonal mesh

主程序如下

```
load('meshex1.mat');
showmesh(node,elem);
findnode(node);
findelem(node,elem);
```

标记单元的函数如下

CODE 6. findelem.m

```
function findelem(node,elem,range)
%Findelem highlights some elements

hold on

if nargin==2
    range = (1:size(elem,1))';
end

center = zeros(length(range),2);
s = 1;
for iel = range(1):range(end)
```

```
    if iscell(elem)
        index = elem{iel};
    else
        index = elem(iel,:);
    end
    verts = node(index, :); verts1 = verts([2:end,1],:);
    area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
    area = 0.5*abs(sum(area_components));
    center(s,:) = sum((verts+verts1).*repmat(area_components,1,2))/(6*area);
    s = s+1;
end

plot(center(:,1),center(:,2),'o','LineWidth',1,'MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',18);
text(center(:,1)-0.02,center(:,2),int2str(range),'FontSize',12,...
    'FontWeight','bold','Color','k');

hold off
```

注 1.3 这里用圆圈标记单元, 对不同的剖分, 圆圈内的数字不一定在合适的位置, 需要手动调整. 为了方便, 可直接用红色数字标记单元序号.

1.3.3 边的标记

Chen L 在如下网页

https://www.math.uci.edu/~chenlong/ifemdoc/mesh/auxstructuredoc.html

中给出了一些辅助网格数据结构 (三角剖分), 其中的 `edge` 就是记录每条边的顶点编号 (去除重复边). 以下设 NT 表示三角形单元的个数, NE 表示边的个数 (不重复). 我们简单说明一下那里的思路.

- 只要给出每条边两端的节点编号. 内部边在 `elem` 中会出现两次, 边界边只会出现一次, 我们可用 2 标记内部边, 1 标记边界边.

- 内部边在 `elem` 中会出现两次, 但它们是同一条边. 为了给定一致的标记, 我们规定每条边起点的顶点编号小于终点的顶点编号, 即 `edge(k,1)<edge(k,2)`.

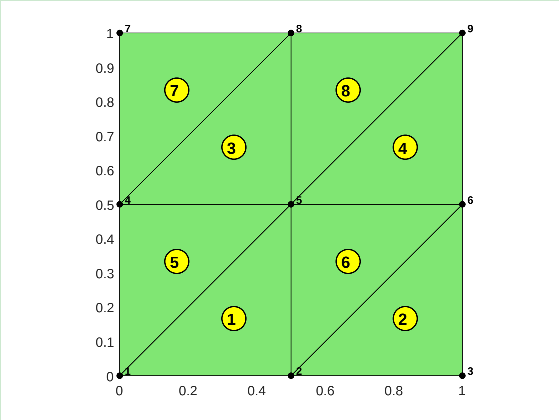
- 规定三角形的第 i 条边对应第 i 个顶点 (不是必须的, 这个规定有利于网格二分程序的实现), 例如, 设第 1 个三角形顶点顺序为 [1,4,5], 那么边的顺序应是 4-5, 5-1, 1-4. 在 MATLAB 中, 有如下对应

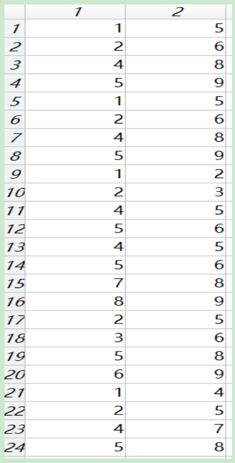
```
所有单元的第 1 条边: elem(:,[2,3]);  % NT * 2
所有单元的第 2 条边: elem(:,[3,1]);  % NT * 2
所有单元的第 3 条边: elem(:,[1,2]);  % NT * 2
```

为了满足 `edge(k,1)<edge(k,2)`, 可对以上每个矩阵按行进行排列 (每行的两个元素进行比较). 在 MATLAB 中用 `sort(A,2)` 实现. 把这些边逐行排在一起, 则所有的边 (包含重复) 为

```
totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
```

它是 `3NT * 2` 的矩阵. `totalEdge` 见下面的右图.





	1	2
1	1	5
2	2	6
3	4	8
4	5	9
5	1	5
6	2	6
7	4	8
8	5	9
9	1	2
10	2	3
11	4	5
12	5	6
13	4	5
14	5	6
15	7	8
16	8	9
17	2	5
18	3	6
19	5	8
20	6	9
21	1	4
22	2	5
23	4	7
24	5	8

- 在 MATLAB 中, `sparse` 有一个特殊的性质 (summation property), 当某个位置指标出现两次, 则相应的值会相加. 这样, 使用如下命令 (`sparse(i,j,s)`, 若 s 为固定常数, 直接写常数即可)

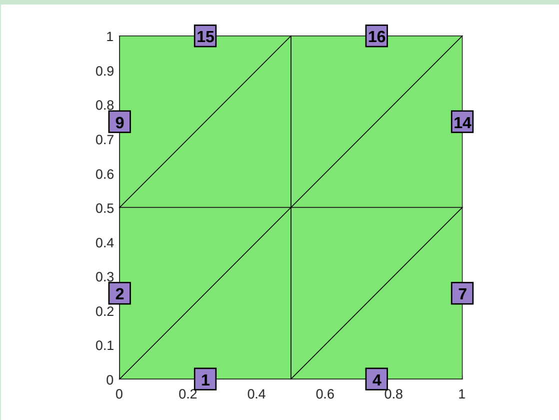
  ```
  sparse(totalEdge(:,1),totalEdge(:,2),1)
  ```

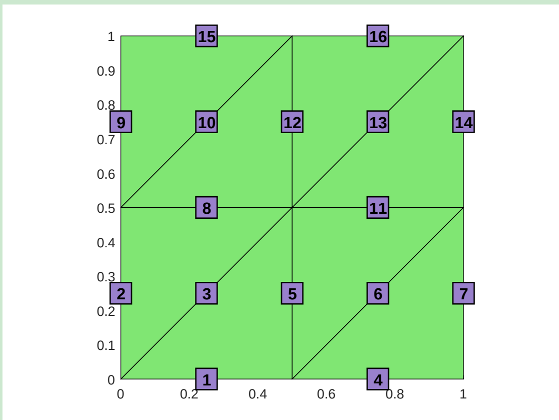
  1-5 边对应的位置 (1,5) 的值就是 2, 而 1-2 对应的位置 (1,2) 为 1, 即重复边的都是 2, 不重复的为 1. 用 `find` 可找到所有非零元素的位置及相应的值 (非零元素只有 1 和 2, 对应边界边和内部边). 显然, `sparse` 命令产生的矩阵, 第一行对应起点 1 的边, 第二行对应起点 2 的边, 等等.

- 我们希望按下列方式排列边: 先找到所有起点为 1 的边, 再找所有起点为 2 的边, 等等. 由于 `find` 是按列找非零元素, 因此我们要把上一步的过程如下修改 (转置)

  ```
  sparse(totalEdge(:,2),totalEdge(:,1),1)
  ```

  这样, 第一列对应的是起点为 1 的边, 第二列对应的是起点为 2 的边.





综上, 我们可如下标记边界边或所有的边

```
% -------------- edge ----------
[node,elem] = squaremesh([0 1 0 1],0.5);
figure, % boundary edges
showmesh(node,elem);
bdInd = 1;
```

```
findedgeTr(node,elem,bdInd);
figure, % all edges
showmesh(node,elem);
findedgeTr(node,elem);
```

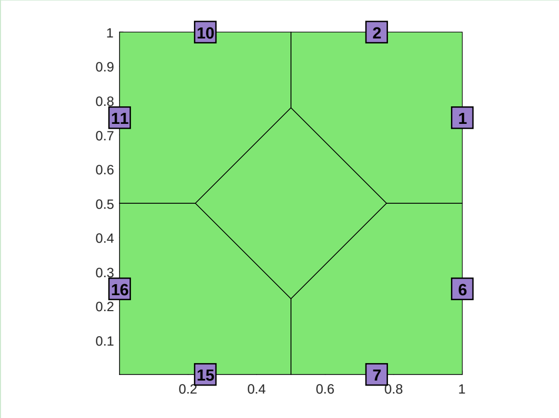
函数文件如下

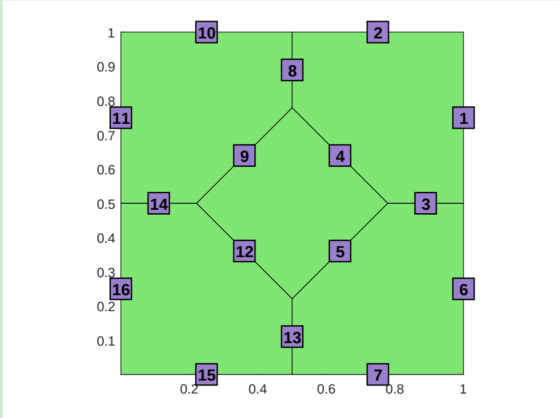
```
function findedgeTr(node,elem,bdInd)
%FindedgeTr highlights edges for triangulation
% bdEdge = 1; % boundary edge;
% other cases: all edges

hold on
% -------- edge matrix -------
totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
% bdEdge = edge(s==1,:);

% ------- range ---------
if nargin==2 || bdInd≠1
    range = (1:size(edge,1))'; % all edges
else
    range = find(s==1); % boundary edges
end

% ------ edge index ------
midEdge = (node(edge(range,1),:)+node(edge(range,2),:))/2;
plot(midEdge(:,1),midEdge(:,2),'s','LineWidth',1,'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.6 0.5 0.8],'MarkerSize',20);
text(midEdge(:,1)-0.025,midEdge(:,2),int2str(range), ...
    'FontSize',12,'FontWeight','bold','Color','k');
```

注意因为前面进行了转置, `edge = [j,i]`.

上面的思想也可用于多角形剖分, 只不过因边数不同, 要逐个单元存储每条边. 图 3 中第 1 个单元的顶点顺序为 [8,10,4,3,2], 我们按顺序 8-10, 10-4, 4-3, 3-2, 2-8 给出单元的边的标记. 所有边的起点就是 `elem` 中元素按列拉直给出的结果, 而终点就是对 [10,4,3,2,8] 这种循环的结果拉直. 可如下实现

```
% the starting points of edges
v0 = horzcat(elem{:})';

% the ending points of edges
shiftfun = @(verts) [verts(2:end),verts(1)];
T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
v1 = horzcat(elem{:})';
```





其他过程与三角剖分一致. 如下运行

```
% -------------- edge (polygonal meshes) ----------
load('meshex1.mat');
figure, % boundary edges
showmesh(node,elem);
bdInd = 1;
findedge(node,elem,bdInd)
figure, % all edges
showmesh(node,elem);
findedge(node,elem)
```

函数文件如下

CODE 7. findedge.m

```
function findedge(node,elem,bdInd)
%Findedge highlights edges
% bdEdge = 1; % boundary edge;
% other cases: all edges

hold on
% -------- edge matrix -------
if iscell(elem)
    shiftfun = @(verts) [verts(2:end),verts(1)];
    T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
    v0 = horzcat(elem{:})'; % the starting points of edges
    v1 = horzcat(T1{:})'; % the ending points of edges
    totalEdge = sort([v0,v1],2);
else
    totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
end
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
% bdEdge = edge(s==1,:);

% ------- range ---------
if nargin==2 || bdInd≠1
    range = (1:size(edge,1))'; % all edges
```

```
else
    range = find(s==1); % boundary edges
end

% ------ edge index ------
midEdge = (node(edge(range,1),:)+node(edge(range,2),:))/2;
plot(midEdge(:,1),midEdge(:,2),'s','LineWidth',1,'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.6 0.5 0.8],'MarkerSize',20);
text(midEdge(:,1)-0.025,midEdge(:,2),int2str(range), ...
    'FontSize',12,'FontWeight','bold','Color','k');
```

注 1.4 如果只是单纯生成 edge 矩阵, 那么也可如下

```
edge = unique(totalEdge,'rows');
```

unique 的速度要比前面给出的方式慢, 但后者方式可以用来生成对应单元的边界, 命名为 elem2edge, 它在计算中更重要.

1.4 VEM 空间的辅助数据结构与几何量

网格中有许多数据在计算中很有用, 例如边的标记、单元的直径、面积等. 参考 iFEM 的相关内容, 见网页

https://www.math.uci.edu/~chenlong/ifemdoc/mesh/auxstructuredoc.html

本节针对一般的多角形剖分给出需要的数据结构与几何量.

我们的数据结构包括

表 1. 数据结构

node, elem	基本数据结构
elem2edge	边的自然序号 (单元存储)
edge	一维边的端点标记
bdEdge	边界边的端点标记
edge2elem	边的左右单元
neighbor	目标单元边的相邻单元

1.4.1 elem2edge 的生成

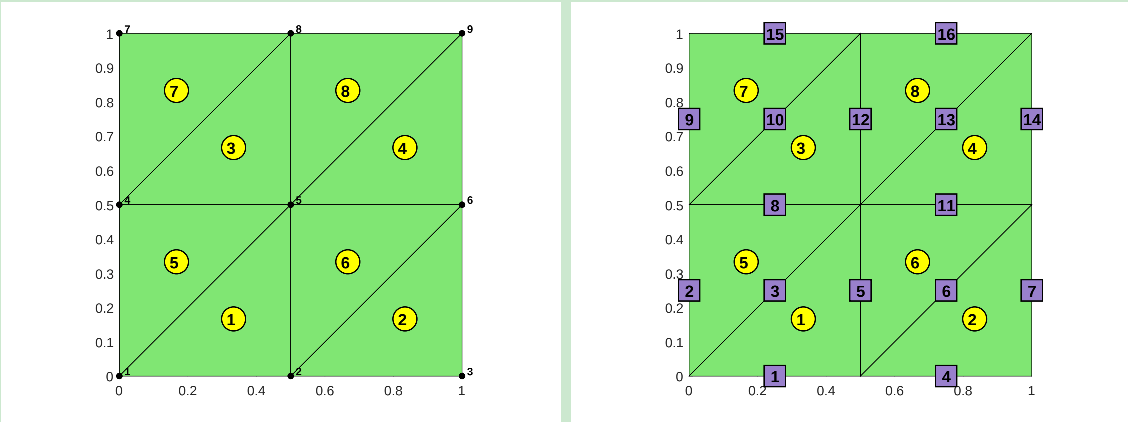
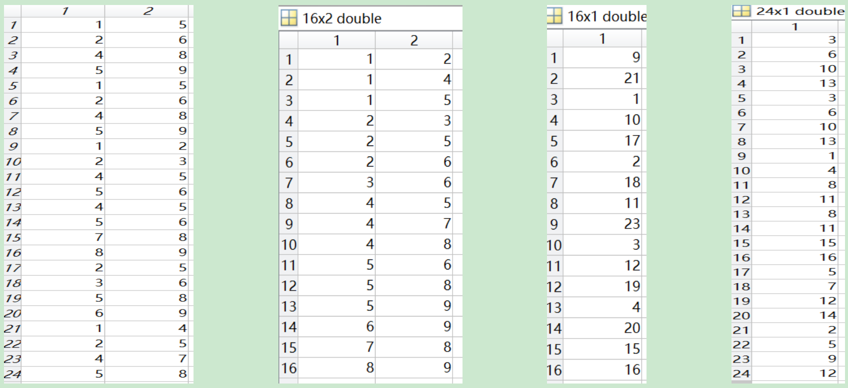


图 4. 三角剖分边的自然序号

考虑图 4 中给出的三角剖分, 我们说明一下 iFEM 中的思路.

- 根据前面的说明, 我们可给出含重复边的数组 `totalEdge` 见图 5(a).



	1	2
1	1	5
2	2	6
3	4	8
4	5	9
5	1	5
6	2	6
7	4	8
8	5	9
9	1	2
10	2	3
11	4	5
12	5	6
13	4	5
14	5	6
15	7	8
16	8	9
17	2	5
18	3	6
19	5	8
20	6	9
21	1	4
22	2	5
23	4	7
24	5	8

(a) totalEdge

16x2 double

	1	2
1	1	2
2	1	4
3	1	5
4	2	3
5	2	5
6	2	6
7	3	6
8	4	5
9	4	7
10	4	8
11	5	6
12	5	8
13	5	9
14	6	9
15	7	8
16	8	9

(b) edge

16x1 double

	1
1	9
2	21
3	1
4	10
5	17
6	2
7	18
8	11
9	23
10	3
11	12
12	19
13	4
14	20
15	15
16	16

(c) i1

24x1 double

	1
1	3
2	6
3	10
4	13
5	3
6	6
7	10
8	13
9	1
10	4
11	8
12	11
13	8
14	11
15	15
16	16
17	5
18	7
19	12
20	14
21	2
22	5
23	9
24	12

(d) totalJ

图 5. elem2edge 图示

- 如下可去除重复的行, 即重复的边 (重复边一致化才能使用)

```
[edge, i1, totalJ] = unique(totalEdge,'rows');
```

这里, `edge` 是 `NE * 2` 的矩阵, 对应边的集合, 注意 `unique` 会按第一列从小到大给出边 (相应地第二列也进行了排序), 见图 5(b).

`i1` 是 `NE * 1` 的数组, 它记录 `edge` 中的每条边在原来的 `totalEdge` 的位置 (重复的按第一次出现记录). 比如, 上面的 1-5 边, 第一次出现的序号是 1, 则 `i1` 第一个元素就是 1.

`totalJ` 记录的是 `totalEdge` 的每条边在 `edge` 中的自然序号. 比如, 1-5 在 `edge` 中是第 3 个, 则 `totalEdge` 的所有 1-5 边的序号为 3.

- 只要把 `totalJ` 恢复成三列即得所有三角形单元边的自然序号, 这是因为 `totalEdge` 排列的规则是: 前 `NT` 行对应所有单元的第 1 条边, 中间 `NT` 行对应第 2 条边, 最后 `NT` 行对应第 3 条边. 综上, 可如下获取 `elem2edge`.

```
% -------- elem2edge (triangulation) --------
[node,elem] = squaremesh([0 1 0 1],0.5);
totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
[edge, i1, totalJ] = unique(totalEdge,'rows');
NT = size(elem,1);
elem2edge = reshape(totalJ,NT,3);
```

结果如下

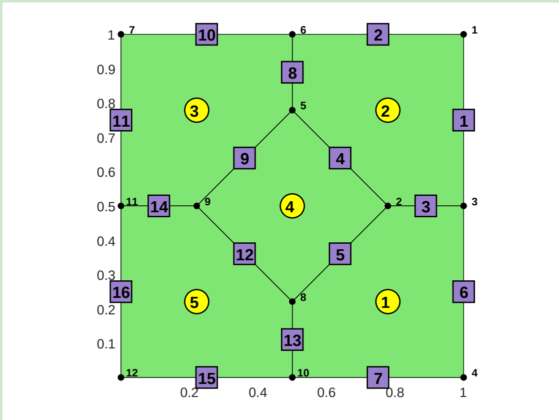
8x3 double

	1	2	3
1	3	1	5
2	6	4	7
3	10	8	12
4	13	11	14
5	3	8	2
6	6	11	5
7	10	15	9
8	13	16	12

上面的思路适用于多角形剖分, 只不过此时因边数不同, 要逐个单元存储每条边, 以保证对应 (三角形按局部边存储可快速恢复). 当我们获得 `totalJ` 后, 它与 `totalEdge` 的行对应, 从而可对应 `elem` 获得 `elem2edge`. 在 MATLAB 中可用 `mat2cell` 实现, 请参考相关说明. 我们把前面获取 `edge`, `bdEdge` 以及这里的 `elem2edge` 的过程放在一个 M 文件中

```
% % -------- elem2edge (triangulation) --------
% [node,elem] = squaremesh([0 1 0 1],0.5);
% showmesh(node,elem); findelem(node,elem); findnode(node);
% findedge(node,elem);

% -------- elem2edge (polygonal meshes) --------
load('meshex1.mat');
showmesh(node,elem);
findnode(node); findelem(node,elem);
findedge(node,elem);

if iscell(elem)
    % totalEdge
    shiftfun = @(verts) [verts(2:end),verts(1)];  % or shiftfun = @(verts) ...
        circshift(verts,-1);
    T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
    v0 = horzcat(elem{:})'; % the starting points of edges
    v1 = horzcat(T1{:})'; % the ending points of edges
    totalEdge = sort([v0,v1],2);

    % elem2edge
    [¬, ¬, totalJ] = unique(totalEdge,'rows');
    elemLen = cellfun('length',elem); % length of each element
    elem2edge = mat2cell(totalJ,elemLen,1);
```

```
    elem2edge = cellfun(@transpose, elem2edge, 'UniformOutput', false);

else % Triangulation
    totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
    [¬, ¬, totalJ] = unique(totalEdge,'rows');
    NT = size(elem,1);
    elem2edge = reshape(totalJ,NT,3);
end

% -------- edge, bdEdge --------
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
bdEdge = edge(s==1,:);
```



结果如下

5x1 cell

	1
1	[13,7,6,3,5]
2	[3,1,2,8,4]
3	[14,9,8,10,11]
4	[12,5,4,9]
5	[15,13,12,14,16]

(a) elem2edge

16x2 double

	1	2
1	1	3
2	1	6
3	2	3
4	2	5
5	2	8
6	3	4
7	4	10
8	5	6
9	5	9
10	6	7
11	7	11
12	8	9
13	8	10
14	9	11
15	10	12
16	11	12

(b) edge

8x2 double

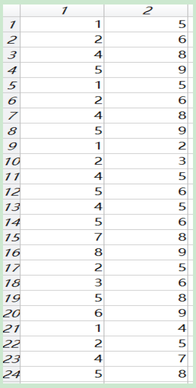
	1	2
1	1	3
2	1	6
3	3	4
4	4	10
5	6	7
6	7	11
7	10	12
8	11	12

(c) bdEdge

图 **6.** Auxiliary mesh data structure

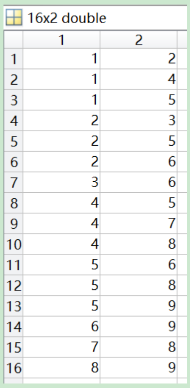
1.4.2 edge2elem 的生成

对给定的一条边 e, 有时候希望知道包含它的单元有哪些. 对内部边, 就是哪两个单元以 e 为公共边. 为此, 我们定义矩阵 `edge2elem`, 维数为 `NE*2`, 其中 `NE` 是一维边的个数. 它的前两列分别存储相邻的三角形编号. 为了方便, 称第一列为左单元编号, 第二列为右单元编号. 注意, 对边界边我们规定两个编号一致.



	1	2
1	1	5
2	2	6
3	4	8
4	5	9
5	1	5
6	2	6
7	4	8
8	5	9
9	1	2
10	2	3
11	4	5
12	5	6
13	4	5
14	5	6
15	7	8
16	8	9
17	2	5
18	3	6
19	5	8
20	6	9
21	1	4
22	2	5
23	4	7
24	5	8

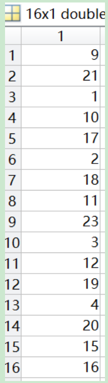
(a) totalEdge



16x2 double

	1	2
1	1	2
2	1	4
3	1	5
4	2	3
5	2	5
6	2	6
7	3	6
8	4	5
9	4	7
10	4	8
11	5	6
12	5	8
13	5	9
14	6	9
15	7	8
16	8	9

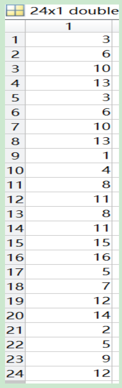
(b) edge



16x1 double

	1
1	9
2	21
3	1
4	10
5	17
6	2
7	18
8	11
9	23
10	3
11	12
12	19
13	4
14	20
15	15
16	16

(c) i1



24x1 double

	1
1	3
2	6
3	10
4	13
5	3
6	6
7	10
8	13
9	1
10	4
11	8
12	11
13	8
14	11
15	15
16	16
17	5
18	7
19	12
20	14
21	2
22	5
23	9
24	12

(d) totalJ

- `totalEdge` 记录了所有的重复边, 称第一次出现的重复边为左单元边, 第二次出现的重复边为右单元边. 根据前面的说明,

  ```
  [~, i1, totalJ] = unique(totalEdge,'rows');
  ```

  执行上面语句给出的 `i1` 记录了左单元边.

- 类似地, 对 `totalEdge` 的逆序使用 `unique`:

  ```
  [~, i2] = unique(totalEdge(end:-1:1,:),'rows');
  ```

  或

  ```
  [~, i2] = unique(totalJ(end:-1:1),'rows');
  ```

  给出的 `i2` 记录了左单元边, 但现在的序号与原先的有差别. 以图中的例子为例, 此时 1 相当于原来的 24, 2 相当于 23, 依此类推. 它们的和总是 25, 即 `length(totalEdge)+1` (三角形为 `3*NT+1`). 这样, 还原后的为

  ```
  i2 = length(totalEdge)+1-i2;
  ```

- `totalJ` 或 `totalEdge` 并不是逐个单元存储的. 对三角剖分, 它是如下存储的

  ```
  所有单元的第 1 条边: elem(:,[2,3]);  % NT * 2
  所有单元的第 2 条边: elem(:,[3,1]);  % NT * 2
  所有单元的第 3 条边: elem(:,[1,2]);  % NT * 2
  ```

  即, 前 `NT` 行与所有单元的第 1 条边对应, 中间的 `NT` 行与所有单元的第 2 条边对应, 最后的 `NT` 行与所有单元的第 3 条边对应. 设 `totalJ` 行的单元序号为 `totalJelem`, 则

  ```
  totalJelem = repmat((1:NT)',3,1);
  ```

- 综上, 对三角形剖分, 有

  ```
  edge2elem = totalJelem([i1,i2]);
  ```

- 对多角形剖分, 只要修改 `totalJelem` 即可. 对多角形情形, `totalEdge` 是按单元排列的, 只要按单元边数进行编号即可. 例如, 前面给出的例子, 每个单元的边数为

5x1 double

	1
1	5
2	5
3	5
4	4
5	5

这里, 第 1 个单元有 5 条边, 第 2 个单元有 5 条边, 等等. 为此, totalJelem 的前 5 行都为 1 (对应单元 1), 接着的 5 行为 2, 等等. 如下给出

```
Num = num2cell((1:NT)');
Len = num2cell(cellLen);
totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, ...
    'UniformOutput', false);
totalJelem = vertcat(totalJelem{:});
```

综上, 可如下实现 edge2elem

```
% ------- edge2elem --------
if iscell(elem)
    Num = num2cell((1:NT)');  Len = num2cell(cellLen);
    totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', ...
        false);
    totalJelem = vertcat(totalJelem{:});
else
    totalJelem = repmat((1:NT)',3,1);
end
[¬, i2] = unique(totalJ(end:-1:1),'rows');
i2 = length(totalEdge)+1-i2;
edge2elem = totalJelem([i1,i2]);
```

多角形剖分结果如下

16x2 double

	1	2
1	2	2
2	2	2
3	1	2
4	2	4
5	1	4
6	1	1
7	1	1
8	2	3
9	3	4
10	3	3
11	3	3
12	4	5
13	1	5
14	3	5
15	5	5
16	5	5

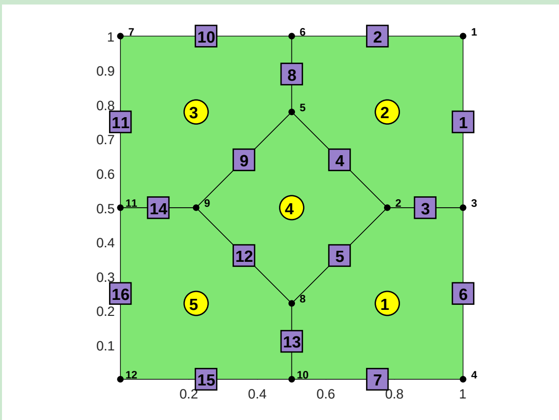


图 7. edge2elem

例如, 序号 12 的边连接的两个单元编号为 4 和 5.

注 1.5 totalEdge 是按单元顺序排列的, i1 对应 e 第一次出现的单元, i2 对应第二次出现的单元, 自然 edge2elem 的第一列单元序号小于或等于第二列单元序号.

1.4.3 neighbor 的生成

`neighbor` 是 `NT*NE` 的矩阵, 它的结构如下

`neighbor` 的结构

i	j	neighbor(i,j)
Ki	ej	相邻三角形

`edge2elem(:,1)` 对应左单元 K_i, 行索引对应边 e_j, 相邻三角形是 `edge2elem(:,2)`.
`edge2elem(:,2)` 对应右单元 K_i, 行索引对应边 e_j, 相邻三角形是 `edge2elem(:,1)`.
可用 sparse 实现 `neighbor`.

```
% --------- neighbor ---------
NE = size(edge,1);
ii1 = edge2elem(:,1); jj1 = (1:NE)'; ss1 = edge2elem(:,2);
ii2 = edge2elem(:,2); jj2 = (1:NE)'; ss2 = edge2elem(:,1);
label = (ii2≠ss2);
ii2 = ii2(label); jj2 = jj2(label); ss2 = ss2(label);
ii = [ii1;ii2]; jj = [jj1;jj2]; ss = [ss1;ss2];
neighbor = sparse(ii,jj,ss,NT,NE);
```

注 1.6 本文给出的 `neighbor` 与 iFEM 不同, 那里是按单元给出每个顶点相对的单元序号, 这是由三角形的特殊性决定的.

1.5 网格相关的几何量

几何量包括

表 2. 几何量

elemCentroid	单元重心坐标
area	单元面积
diameter	单元直径

- 单元的重心如下计算

$$x_K = \frac{1}{6|K|}\sum_{i=0}^{N_v-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i),$$

$$y_K = \frac{1}{6|K|}\sum_{i=0}^{N_v-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i),$$

  这里 N_v 是单元顶点个数.

- 单元面积

$$|K| = \frac{1}{2}\left|\sum_{i=0}^{N_v-1} x_i y_{i+1} - x_{i+1} y_i\right|.$$

- 单元直径就是所有顶点之间最长的距离, MATLAB 提供了 `pdist` 函数, 它计算各对行向量的相互距离.

以上几何量可如下获得

```
% ------ elemCentroid, area, diameter -------
elemCentroid = zeros(NT,2); area = zeros(NT,1); diameter = zeros(NT,1);
s = 1;
for iel = 1:NT
    if iscell(elem)
        index = elem{iel};
    else
        index = elem(iel,:);
    end
    verts = node(index, :); verts1 = verts([2:end,1],:);
    area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
    ar = 0.5*abs(sum(area_components));
    area(iel) = ar;
    elemCentroid(s,:) = ...
        sum((verts+verts1).*repmat(area_components,1,2))/(6*ar);
    diameter(s) = max(pdist(verts));
    s = s+1;
end
```

1.6 auxstructure 与 auxgeometry 函数

为了输出方便, 我们把所有的数据结构或几何量保存在结构体 `aux` 中. 考虑到数据结构在编程中不一定使用 (处理网格时用), 我们把数据结构与几何量分别用函数生成, 命名为 auxstructure.m 和 auxgeometry.m. 为了方便使用, 程序中把三角剖分按单元存储的数据转化为元胞数组. auxstructure.m 函数如下

CODE 8. auxstructure.m

```
function aux = auxstructure(node,elem)

NT = size(elem,1);
if iscell(elem)
    % totalEdge
    shiftfun = @(verts) [verts(2:end),verts(1)];  % or shiftfun = @(verts) ...
        circshift(verts,-1);
    T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
    v0 = horzcat(elem{:})'; % the starting points of edges
    v1 = horzcat(T1{:})'; % the ending points of edges
    totalEdge = sort([v0,v1],2);

    % -------- elem2edge: elementwise edges -------
    [¬, i1, totalJ] = unique(totalEdge,'rows');
    elemLen = cellfun('length',elem); % length of each elem
    elem2edge = mat2cell(totalJ,elemLen,1);
    elem2edge = cellfun(@transpose, elem2edge, 'UniformOutput', false);
```

```
else % Triangulation
    totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
    [¬, i1, totalJ] = unique(totalEdge,'rows');
    elem2edge = reshape(totalJ,NT,3);
end

% -------- edge, bdEdge --------
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
bdEdge = edge(s==1,:);

% ------- edge2elem --------
if iscell(elem)
    Num = num2cell((1:NT)');    Len = num2cell(elemLen);
    totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', ...
        false);
    totalJelem = vertcat(totalJelem{:});
else
    totalJelem = repmat((1:NT)',3,1);
end
[¬, i2] = unique(totalJ(end:-1:1),'rows');
i2 = length(totalEdge)+1-i2;
edge2elem = totalJelem([i1,i2]);

% --------- neighbor ---------
NE = size(edge,1);
ii1 = edge2elem(:,1); jj1 = (1:NE)'; ss1 = edge2elem(:,2);
ii2 = edge2elem(:,2); jj2 = (1:NE)'; ss2 = edge2elem(:,1);
label = (ii2≠ss2);
ii2 = ii2(label); jj2 = jj2(label); ss2 = ss2(label);
ii = [ii1;ii2]; jj = [jj1;jj2]; ss = [ss1;ss2];
neighbor = sparse(ii,jj,ss,NT,NE);


if ¬iscell(elem) % transform to cell
    elem = mat2cell(elem,ones(NT,1),3);
    elem2edge = mat2cell(elem2edge,ones(NT,1),3);
end

aux.node = node; aux.elem = elem;
aux.elem2edge = elem2edge;
aux.edge = edge; aux.bdEdge = bdEdge;
aux.edge2elem = edge2elem;
aux.neighbor = neighbor;
```

auxgeometry.m 函数如下

CODE 9. auxgeometry.m

```
function aux = auxgeometry(node,elem)
```

```

% ------ elemCentroid, area, diameter -------
NT = size(elem,1);
elemCentroid = zeros(NT,2); area = zeros(NT,1); diameter = zeros(NT,1);
s = 1;
for iel = 1:NT
    if iscell(elem)
        index = elem{iel};
    else
        index = elem(iel,:);
    end
    verts = node(index, :); verts1 = verts([2:end,1],:);
    area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
    ar = 0.5*abs(sum(area_components));
    area(iel) = ar;
    elemCentroid(s,:) = ...
        sum((verts+verts1).*repmat(area_components,1,2))/(6*ar);
    diameter(s) = max(pdist(verts));
    s = s+1;
end

if ¬iscell(elem) % transform to cell
    elem = mat2cell(elem,ones(NT,1),3);
end

aux.node = node; aux.elem = elem;
aux.elemCentroid = elemCentroid;
aux.area = area;
aux.diameter = diameter;
```

1.7 边界设置

假设网格的边界只有 Dirichlet 与 Neumann 两种类型, 前者用 `eD` 存储 Dirichlet 节点的编号, 后者用 `elemN` 存储 Neumann 边界的起点和终点编号 (即一维问题的连通性信息).

1.7.1 边界边的定向

辅助数据结构中曾给出了边界边 `bdEdge`, 但它的定向不再是逆时针, 因为我们规定 `edge(k,1)<edge(k,2)`. Neumann 边界条件中会遇到 $\partial_n u$, 这就需要我们恢复边界边的定向以确定外法向量 (边的旋转获得).

给定 `totalEdge`, 即所有单元的边 (含重复且无定向), 我们有两种方式获得边 (第一种可获得边界边, 第二种只获得所有边):

- 一是累计重复的次数 (1 是边界, 2 是内部)

```
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
bdEdge = edge(s==1,:);
```

- 二是直接去掉重复的边

```
[edge, i1, ¬] = unique(totalEdge,'rows');
```

这里, i1 记录的是 edge 在重复边 totalEdge 中的位置.

显然, i1(s==1) 给出的是边界边 bdEge 在 totalEdge 中的位置. totalEdge 的原来定向是知道的, 由此就可确定边界边的定向, 程序如下

```
[node,elem] = squaremesh([0 1 0 1],0.5);
NT = size(elem,1);
% totalEdge
if iscell(elem)
    shiftfun = @(verts) [verts(2:end),verts(1)];
    T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
    v0 = horzcat(elem{:})'; % the starting points of edges
    v1 = horzcat(T1{:})'; % the ending points of edges
    allEdge = [v0,v1];
    totalEdge = sort(allEdge,2);
else
    allEdge = [elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])];
    totalEdge = sort(allEdge,2);
end
[¬,¬,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
[edge, i1, ¬] = unique(totalEdge,'rows');
bdEdge = allEdge(i1(s==1),:); % counterclockwise
```

1.7.2 边界的设置

下面说明如何实现 eD 和 elemN. 以下图为例

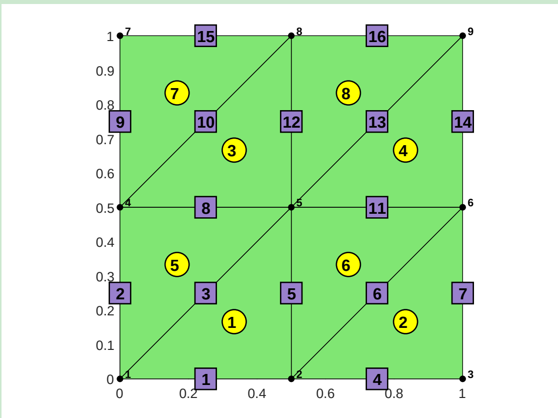


图 8. 边的自然序号

- 边界边的序号顺序为 1, 2, 4, 7, 9, 14, 15, 16. 定向的 bdEdge 给出的是这些边的起点与终点编号, 只要按索引对应即可.

- 边界我们用函数确定, 例如矩形 $[0,1]^2$ 的右边界为满足 $x=1$ 的线段组成. 只需要判断 bdEdge 对应的边的中点在不在该线段上. 如下

```
bdFun = 'x==1';
nodebdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
x = nodebdEdge(:,1); y = nodebdEdge(:,2);
id = eval(bdFun);
```

这里, `eval` 将字符串视为语句并运行. 现在给定了若干个 `x`, 执行 `eval(bdFun)` 就会判断哪些 `x` 满足条件, 返回的是逻辑数组 `id = [0 0 0 1 0 1 0 0]'`, 即索引中的第 4,6 条边在右边界上.

- 这样, 我们就可抽取出需要的边 `bdEdge(id,:)`. 需要注意的是, `node` 在边界上不一定精确为 1, 通常将上面的 `bdFun` 修改为

  ```
  bdFun = 'abs(x-1)<1e-4';
  ```

- Neumann 边界通常比 Dirichlet 边界少, 为此在建函数的时候, 输入的字符串默认为是 Neumann 边界的, 其他的都是 Dirichlet. 另外, 当没有边界字符串的时候, 规定所有边都是 Dirichlet 边.

根据上面的讨论, 我们可以给出函数 setboundary.m

CODE 10. setboundary.m

```
function bdStruct= setboundary(node,elem,varargin)
% varargin: string for Neumann boundary

% ------- totalEdge ---------
if iscell(elem)
    shiftfun = @(verts) [verts(2:end),verts(1)];  % or shiftfun = @(verts) ...
        circshift(verts,-1);
    T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
    v0 = horzcat(elem{:})'; % the starting points of edges
    v1 = horzcat(T1{:})'; % the ending points of edges
    allEdge = [v0,v1];
else % Triangulation
    allEdge = [elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])];
end
totalEdge = sort(allEdge,2);

% --------  counterclockwise bdEdge --------
[¬,¬,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
[¬, i1, ¬] = unique(totalEdge,'rows');
bdEdge = allEdge(i1(s==1),:);

% --------- set boundary --------
nE = size(bdEdge,1);
% initial as Dirichlet (true for Dirichlet, false for Neumann)
bdFlag = true(nE,1);
nodebdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
x = nodebdEdge(:,1); y = nodebdEdge(:,2);
nvar = length(varargin); % 1 * size(varargin,2)
% note that length(varargin) = 1 for bdNeumann = [] or ''
```

```
if (nargin==2) || (¬isempty(varargin{1}))
    for i = 1:nvar
        bdNeumann = varargin{i};
        id = eval(bdNeumann);
        bdFlag(id) = false;
    end
end
bdStruct.eD = unique(bdEdge(bdFlag,:));
bdStruct.elemN = bdEdge(¬bdFlag,:);
```

这里, `ed` 和 `elemN` 保存在结构体 `bdStruct` 中. 例如,

1. 以下命令给出的边界全是 Dirichlet 边界:

   `bdStruct = setboundary(node,elem);`

   `bdStruct = setboundary(node,elem, []);`

   `bdStruct = setboundary(node,elem, '');`

2. `bdStruct = setboundary(node,elem,'x==1')` 将右边界设为 Neumann 边, 其他为 Dirichlet 边.

3. `bdStruct = setboundary(node,elem,'(x==1)|(y==1)')` 将右边界与上边界设为 Neumann 边.

注 1.7 以下两种写法等价

```
bdStruct = setboundary(node,elem,'(x==1)|(y==1)');
bdStruct = setboundary(node,elem,'x==1', 'y==1');
```

2 Poisson 方程的一阶虚拟元方法

考虑 Poisson 方程的 Dirichlet 问题

$$\begin{cases} -\Delta u = f & \text{in} \quad \Omega, \\ u = g & \text{on} \quad \partial\Omega. \end{cases} \tag{2.1}$$

为了方便, 以下设 $g=0$.

2.1 变分问题

VEM 的局部形函数空间为

$$V_k(K) := \left\{ v \in H^1(K): \quad \Delta v \in \mathbb{P}_{k-2}(K), \quad v|_{\partial K} \in \mathbb{B}_k(\partial K) \right\},$$

式中,

$$\mathbb{B}_k(\partial K) = \left\{ v \in C^k(\partial K): \quad v|_e \in \mathbb{P}_k(e), \quad e \subset \partial K \right\}$$

称为边界元空间. 给定区域的剖分 $\mathcal{T}_h$, 对模型问题 (2.1), 相应的有限元空间取为协调有限元空间

$$V_h = \left\{ v \in H_0^1(\Omega) \cap C(\overline{\Omega}): \quad v|_K \in V_k(K), \quad K \in \mathcal{T}_h \right\}.$$

变分问题为: 求 $u_h \in V_h$, 使得

$$a_h(u_h, v) = F_h(v) \quad \forall v \in V_h,$$

式中

$$a_h(u,v) = \sum_{K \in \mathcal{T}_h} a_K^h(u,v),$$

$$a_K^h(u,v) = \int_K \nabla(\Pi_k^\nabla u) \cdot \nabla(\Pi_k^\nabla v) \mathrm{d}x + S_K(u - \Pi_k^\nabla u, v - \Pi_k^\nabla v),$$

$$F_h(v) = \sum_{K \in \mathcal{T}_h} F_K^h(v), \qquad F_K^h(v) = \int_K \Pi_{k-2}^0 f v \mathrm{d}x.$$

规定, $\mathbb{P}_{-1} = \{0\}$, $\Pi_{-1}^0 = \Pi_0^0$ (即常数投影).

局部 H^1 投影定义为

$$\Pi_k^\nabla: \quad H^1(K) \to \mathbb{P}_k(K), \quad v \mapsto \Pi_k^\nabla v,$$

满足

$$\begin{cases} a_K(\Pi_k^\nabla v, p) = a_K(v,p), \quad p \in \mathbb{P}_k(K), \\ P_0(\Pi_k^\nabla v) = P_0(v) \end{cases}$$

或

$$\begin{cases} \left(\nabla(\Pi_k^\nabla v), \nabla p\right)_K = (\nabla v, \nabla p)_K, \quad p \in \mathbb{P}_k(K), \\ P_0(\Pi_k^\nabla v) = P_0(v). \end{cases} \tag{2.2}$$

一次情形的稳定项取为

$$\begin{aligned} S_K(v - \Pi_1^\nabla v, w - \Pi_1^\nabla w) &= \sum_{p_i \in \partial K} (v - \Pi_1^\nabla v)(p_i)(w - \Pi_1^\nabla w)(p_i) \\ &=: \sum_{i=1}^{N_v} \chi_i(v - \Pi_1^\nabla v)\chi_i(w - \Pi_1^\nabla w). \end{aligned}$$

2.2 过渡矩阵 D

为了验证程序的正确性, 后面每一步都会参考文献 [1] 中例子的结果, 那里的数据结构为

```
node = [0 0; 3 0; 3 2; 3/2 4; 0 4];
elem = {1:5};
aux = auxstructure(node,elem);
```

查看结构体 aux 可以看到, 重心、单元直径和单元面积与那里的一致.

类似线性代数, 我们把虚拟元空间 $V_k(K)$ 的基函数排成行向量, 记为

$$\phi^T = (\phi_1, \phi_2, \cdots, \phi_{N_k}),$$

其中, N_k 是基函数的个数. 设 $\mathbb{P}_k(K)$ 的基为

$$m^T = (m_1, m_2, \cdots, m_{N_p}),$$

因 $\mathbb{P}_k(K) \subset V_k(K)$, 故可设

$$(m_1, m_2, \cdots, m_{N_p}) = (\phi_1, \phi_2, \cdots, \phi_{N_k})D, \quad \text{或} \quad m^T = \phi^T D, \tag{2.3}$$

称 D 是从 $V_k(K)$ (的基) 到 $\mathbb{P}_k(K)$ (的基) 的过渡矩阵. 以下约定, 用拉丁字母 i,j 等作为基函数 ϕ^T 的索引, 而用希腊字母 α,β 作为 m^T 的索引. 根据自由度的定义,

$$m_\alpha = \sum_{i=1}^{N_k} \phi_i D_{i\alpha}, \quad D_{i\alpha} = \chi_i(m_\alpha),$$

且 D 是 $N_k \times N_p$ 的矩阵.

过渡矩阵是可计算的. 对一次元, 即 $k=1$ 情形, 有

$$\chi_i(v) = v(p_i), \quad i = 1, \cdots, N_k = N_v,$$

$$m^T = \left\{ m_1(x,y) = 1, \;\; m_2(x,y) = \frac{x - x_K}{h_K}, \;\; m_3(x,y) = \frac{y - y_K}{h_K} \right\},$$

从而

$$D = (D_{i\alpha})_{N_v \times 3}, \quad D_{i,\alpha} = \chi_i(m_\alpha) = m_\alpha(p_i) = \begin{cases} 1, & \alpha = 1 \\ \dfrac{x_i - x_K}{h_K}, & \alpha = 2 \\ \dfrac{y_i - y_K}{h_K}, & \alpha = 3 \end{cases}.$$

显然第一列全为 1. 我们将一次性把所有单元的过渡矩阵 D 计算出来, 并用元胞数组存储. $k=1$ 的程序如下

```
node = [0 0; 3 0; 3 2; 3/2 4; 0 4]; elem = {1:5};
aux = auxstructure(node,elem);

node = aux.node; elem = aux.elem;
nodeCentroid = aux.nodeCentroid;
diameter = aux.diameter;

NT = size(elem,1); D = cell(NT,1); B = cell(NT,1);
```

```
for iel = 1:NT
    index = elem{iel};      Nv = length(index);
    xK = nodeCentroid(iel,1); yK = nodeCentroid(iel,2); hK = diameter(iel);
    x = node(index,1); y = node(index,2);
    m = @(x,y) [(x-xK)./hK,(y-yK)./hK];  % m2,m3

    D1 = zeros(Nv,3); D1(:,1) = 1;
    D1(:,2:3) = m(x,y);
    D{iel} = D1;
end
```

注意这里多个匿名函数的使用. 所得结果与文献 [1] 的一致.

2.3 椭圆投影在基下的矩阵

对 VEM 中的函数, 当它的自由度已知时, 椭圆投影就是可计算的. 而 $\chi_i(\phi_j)=\delta_{ij}$, 故节点基函数的投影可计算. 设椭圆投影算子 Π_k^∇ 在节点基下的矩阵记为 $\mathbf{\Pi}_k^\nabla$, 即

$$\Pi_k^\nabla(\phi_1,\phi_2,\cdots,\phi_{N_k})=(\phi_1,\phi_2,\cdots,\phi_{N_k})\mathbf{\Pi}_k^\nabla \quad 或 \quad \Pi_k^\nabla\phi^T=\phi^T\mathbf{\Pi}_k^\nabla.$$

根据自由度的定义, $\mathbf{\Pi}_k^\nabla$ 的第 j 列就是 $\Pi_k^\nabla\phi_j$ 的自由度向量, 即

$$\mathbf{\Pi}_k^\nabla=\big(\chi_i(\Pi_k^\nabla\phi_j)\big). \tag{2.4}$$

2.3.1 椭圆投影定义的向量形式

定义 2.2 等价于

$$\begin{cases} a_K(\Pi_k^\nabla\phi_i,m_\alpha)=a_K(\phi_i,m_\alpha) \\ P_0(\Pi_k^\nabla\phi_i)=P_0(\phi_i) \end{cases},\quad i=1,\cdots,N_k,\ \ \alpha=1,\cdots,N_p$$

或

$$\begin{cases} (\nabla\Pi_k^\nabla\phi_i,m_\alpha)_K=(\nabla\phi_i,m_\alpha)_K \\ P_0(\Pi_k^\nabla\phi_i)=P_0(\phi_i) \end{cases},\quad i=1,\cdots,N_k,\ \ \alpha=1,\cdots,N_p,$$

写成向量形式为

$$\begin{cases} a_K(m,\Pi_k^\nabla\phi^T)=a_K(m,\nabla\phi^T), \\ P_0(\Pi_k^\nabla\phi^T)=P_0(\phi^T). \end{cases}$$

或

$$\begin{cases} \displaystyle\int_K\nabla m\cdot\nabla\Pi_k^\nabla\phi^T\mathrm{d}x=\int_K\nabla m\cdot\nabla\phi^T\mathrm{d}x, \\ P_0(\Pi_k^\nabla\phi^T)=P_0(\phi^T). \end{cases} \tag{2.5}$$

设投影向量 $\Pi_k^\nabla\phi^T$ 在多项式基 m^T 下的矩阵为 $\mathbf{\Pi}_{k*}^\nabla$, 即

$$\Pi_k^\nabla\phi^T=m^T\mathbf{\Pi}_{k*}^\nabla, \tag{2.6}$$

称 $\mathbf{\Pi}_{k*}^\nabla$ 为椭圆投影关于多项式基的 Riesz 表示. 由过渡矩阵的关系知, 两组不同基下的矩阵满足

$$\mathbf{\Pi}_k^\nabla=D\mathbf{\Pi}_{k*}^\nabla.$$

令

$$G = a_K(m, m^T) = \int_K \nabla m \cdot \nabla m^T \mathrm{d}x,$$

$$B = a_K(m, \phi^T) = \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x,$$

则由 (2.5) 知

$$\begin{cases} G\mathbf{\Pi}_{k*}^{\nabla} = B, \\ P_0(m^T)\mathbf{\Pi}_{k*}^{\nabla} = P_0(\phi^T), \end{cases}$$

其中, G 是 $N_p \times N_p$ 的方阵, B 是 $N_k \times N_p$ 的矩阵, 而第二个方程实际上是一个行向量.

注意, G 不可逆, 因为 ∇m 的第一个分量为零, 从而 G 的第一行全为零. 这是因为 (2.5) 不考虑限制条件时, 投影不唯一. 这样, 我们只要把第一行恒为零的方程用投影的限制条件替换, 即

$$\tilde{G}\mathbf{\Pi}_{k*}^{\nabla} = \tilde{B},$$

式中,

$$\tilde{G} = G + \begin{bmatrix} P_0(m^T) \\ O \end{bmatrix}, \quad \tilde{B} = B + \begin{bmatrix} P_0(\phi^T) \\ O \end{bmatrix}.$$

则 $\tilde{G}$ 可逆, 且

$$\mathbf{\Pi}_{k*}^{\nabla} = \tilde{G}^{-1}\tilde{B}, \quad \mathbf{\Pi}_{k}^{\nabla} = D\mathbf{\Pi}_{k*}^{\nabla}.$$

2.3.2 可计算性

我们要把 (2.5) 的右端用自由度表示, 即计算 B, 它的第一行自然为零. 分部积分有

$$B = \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x = -\int_K \Delta m \cdot \phi^T \mathrm{d}x + \sum_{e\subset\partial K} \int_e (\nabla m \cdot \boldsymbol{n}_e)\phi^T \mathrm{d}s, \tag{2.7}$$

右端第一项转化为 ϕ^T 的自由度计算, 第二项限制在边界上是多项式, 归结为普通的有限元问题.

注意, $\tilde{G}$ 或 G 不需要直接计算, 这是因为我们有如下的一致性关系 (consistency relation).

引理 2.1

$$G = BD, \quad \tilde{G} = \tilde{B}D.$$

证明 对第一式, 直接验证有

$$G = \int_K \nabla m \cdot \nabla m^T \mathrm{d}x = \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x D = BD.$$

对第二式,

$$\begin{aligned} \tilde{G} = G + \begin{bmatrix} P_0(m^T) \\ O \end{bmatrix} &= \int_K \nabla m \cdot \nabla m^T \mathrm{d}x + \begin{bmatrix} P_0(m^T) \\ O \end{bmatrix} \\ &= \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x D + \begin{bmatrix} P_0(\phi^T) \\ O \end{bmatrix} D = \tilde{B}D. \end{aligned}$$

□

因为 D 不可逆, 所以不能通过直接计算 G, 然后反推出 B (B 比较难算). 这体现了 VEM 中可计算问题的思考过程不能省略.

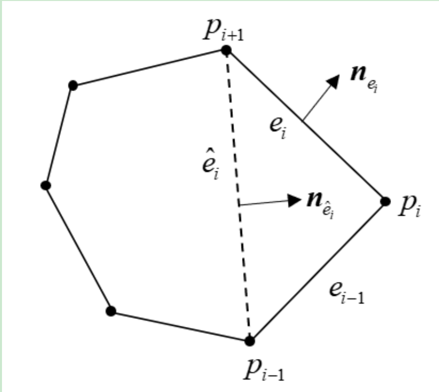


图 9. $\boldsymbol{n}_{\hat{e}_i}$ 的示意图

考虑 $k=1$ 的情形. 尺度单项式的次数不超过 1, 故 $\Delta \underline{m} = \mathbf{0}$, 只需要考虑边界项. 此时, $\nabla \underline{m}$ 的每个分量都是常数, 且外法向量在每个边界单元上也是常数, 都可直接提出积分号. 每个 ϕ_i 在边界上是一次多项式, 且顶点值满足 Kronecker 性质 (第 i 个基对应第 i 个点). 设 p_{i-1} 表示 p_i 的上一个点, p_{i+1} 表示下一个点, 则有

$$\begin{aligned}
&\sum_{e\subset\partial K}\int_e (\nabla m_\alpha \cdot \boldsymbol{n}_e)\phi_i \mathrm{d}s \\
&= \sum_{e\subset\partial K}(\nabla m_\alpha \cdot \boldsymbol{n}_e)\int_e \phi_i \mathrm{d}s = \sum_{j=1}^{N_v}(\nabla m_\alpha \cdot \boldsymbol{n}_{e_j})\frac{\phi_i(p_j)+\phi_i(p_{j+1})}{2}|e_j| \\
&= \nabla m_\alpha \cdot \frac{|e_{i-1}|\,\boldsymbol{n}_{e_{i-1}} + |e_i|\,\boldsymbol{n}_{e_i}}{2}
\end{aligned} \tag{2.8}$$

考察三角形 $\Delta p_{i-1}p_ip_{i+1}$, 我们有

$$\overrightarrow{p_{i+1}p_{i-1}} + \overrightarrow{p_{i-1}p_i} + \overrightarrow{p_ip_{i+1}} = \mathbf{0},$$

三角形逆时针旋转 $90°$ 后结果不变. 显然每个边旋转后都与对应边的内法向量共线, 且长度不变, 故

$$|\hat{e}_i|\,\boldsymbol{n}_{\hat{e}_i} + (-\,|e_{i-1}|\,\boldsymbol{n}_{e_{i-1}}) + (-\,|e_i|\,n_{e_i}) = \mathbf{0}$$

或

$$|\hat{e}_i|\,\boldsymbol{n}_{\hat{e}_i} = |e_{i-1}|\,\boldsymbol{n}_{e_{i-1}} + |e_i|\,\boldsymbol{n}_{e_i}.$$

综上,

$$\sum_{e\subset\partial K}\int_e (\nabla m_\alpha \cdot \boldsymbol{n}_e)\phi_i \mathrm{d}s = \nabla m_\alpha \cdot \frac{|\hat{e}_i|\,\boldsymbol{n}_{\hat{e}_i}}{2},$$

这里 $|\hat{e}_i|\,\boldsymbol{n}_{\hat{e}_i}$ 就是定向边 $\overrightarrow{p_{i+1}p_{i-1}}$ 逆时针旋转 $90°$ 后得到的向量. 注意, 向量 $\boldsymbol{\alpha} = (a,b)$ 旋转 $90°$ 后坐标变为 $(-b,a)$.

为了计算 B, 我们只要计算出

$$\begin{bmatrix} \nabla m_1 \\ \vdots \\ \nabla m_{N_p} \end{bmatrix}, \qquad \frac{1}{2}\left[|\hat{e}_1|\,\boldsymbol{n}_{\hat{e}_1}, \cdots, |\hat{e}_{N_k}|\,\boldsymbol{n}_{\hat{e}_{N_k}}\right]$$

然后进行相乘即可, 这里 ∇m_α 的坐标按行排, $|\hat{e}_i|\,\boldsymbol{n}_{\hat{e}_i}$ 的坐标按列排.

```
node = [0 0; 3 0; 3 2; 3/2 4; 0 4]; elem = {1:5};
aux = auxstructure(node,elem);
diameter = aux.diameter;

NT = size(elem,1); B = cell(NT,1);
for iel = 1:NT
    index = elem{iel};     Nv = length(index);
    hK = diameter(iel);
    x = node(index,1); y = node(index,2);
    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
    Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a ...
        rotation of edge vector
    B{iel} = Gradm*normVec;
end
```

2.3.3 投影限制条件

对 $k=1$, 计算中 P_0 通常采用如下离散形式

$$P_0 v = \frac{1}{N_v}\sum_{i=1}^{N_v} v(p_i), \tag{2.9}$$

从而

$$P_0(\phi^T) = \frac{1}{N_v}[1,\cdots,1].$$

我们把前面所有的矩阵用同一个函数生成.

```
function [D,Bs,G,Gs] = VEM_MAT_Poisson(aux)

% aux = auxgeometry(node,elem);

node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid;
diameter = aux.diameter;
NT = size(elem,1);

D = cell(NT,1);
% B = cell(NT,1);  % it is not used in the computation
Bs = cell(NT,1);
G = cell(NT,1);  Gs = cell(NT,1);
for iel = 1:NT
    index = elem{iel};     Nv = length(index);
    xK = elemCentroid(iel,1); yK = elemCentroid(iel,2); hK = diameter(iel);

    m = @(x,y) [(x-xK)./hK,(y-yK)./hK];  % m2,m3
    x = node(index,1); y = node(index,2);

    % D
    D1 = zeros(Nv,3); D1(:,1) = 1;
```

```
    D1(:,2:3) = m(x,y);   D{iel} = D1;

    % B, Bs, G, Gs
    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
    Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a ...
        rotation of edge vector
    B1 = Gradm*normVec; B1s = B1; B1s(1,:) = 1/Nv;
    % B{iel} = B1;
    Bs{iel} = B1s;
    G{iel} = B1*D1;     Gs{iel} = B1s*D1;
end
```

对 $k=1$ 情形, 文献 [1] 中 $B(2,1)$ 元素应该少了一个负号, 其他结果都一致.

2.4 L^2 投影在基下的矩阵

2.4.1 矩阵表示

L^2 projection is defined by

$$\begin{cases} \Pi_k^0: \ V_k(K) \to \mathbb{P}_k(K), \quad v \mapsto \Pi_k^0 v, \\ (\Pi_k^0 v, q)_K = (v, q)_K, \end{cases}$$

which is equivalent to the following formula

$$(\Pi_k^0 \phi_i, m_\alpha)_K = (\phi_i, m_\alpha)_K, \quad i = 1, \cdots, N_k, \ \alpha = 1, \cdots, N_p \tag{2.10}$$

or

$$\int_K m \Pi_k^0 \phi^T \mathrm{d}x = \int_K m \phi^T \mathrm{d}x. \tag{2.11}$$

设 Π_k^0 在基 ϕ^T 下的矩阵为 $\boldsymbol{\Pi}_k^0$, 即

$$\Pi_k^0 \phi^T = \phi^T \boldsymbol{\Pi}_k^0,$$

而 $\Pi_k^0 \phi^T$ 在多项式基 m^T 下的矩阵为 $\boldsymbol{\Pi}_{k*}^0$, 即

$$\Pi_k^0 \phi^T = m^T \boldsymbol{\Pi}_{k*}^0. \tag{2.12}$$

由 (2.3) 知,

$$\boldsymbol{\Pi}_k^0 = D \boldsymbol{\Pi}_{k*}^0.$$

把这些关系代入 (2.11), 我们有 L^2 投影的矩阵表示

$$H \boldsymbol{\Pi}_{k*}^0 = C,$$

式中,

$$H = \int_K m m^T \mathrm{d}x, \quad C = \int_K m \phi^T \mathrm{d}x.$$

注 2.1 H 是可逆的 (Gram 矩阵), 但 C 不可计算, 因为它涉及到 K 上 $\geq k-1$ 的矩量.

2.4.2 可计算性

为了 L^2 投影的可计算性, 多出的自由度通常用 Π_k^∇ 代替, 即定义

$$(\Pi_k^0 w_h, q)_K = (\Pi_k^\nabla w_h, q)_K, \quad q \in \mathcal{M}_k \backslash \mathcal{M}_{k-2},$$

也就是说, 我们在空间

$$W_k(K) = \left\{ w_h \in \widetilde{V}_k(K) : \ (w_h - \Pi_k^\nabla w_h, q)_K = 0, \ \ q \in \mathcal{M}_k \backslash \mathcal{M}_{k-2} \right\}$$

中考虑问题, 其中

$$\widetilde{V}_k(K) := V_{k,k}(K) = \left\{ v \in H^1(K) : \ \ v|_{\partial K} \in \mathbb{B}_k(\partial K), \ \Delta v \in \mathbb{P}_k(K) \right\}.$$

为此, 定义 $\tilde{C}$ 为

$$\tilde{C}(\alpha, :) = \begin{cases} \displaystyle\int_K m_\alpha \phi^T \mathrm{d}x, & \deg(m_\alpha) \le k-2, \\ \displaystyle\int_K m_\alpha \Pi_k^\nabla \phi^T \mathrm{d}x, & \deg(m_\alpha) = k-1, \ k, \end{cases}$$

从而

$$H \boldsymbol{\Pi}_{k*}^0 = \tilde{C}.$$

注 2.2 上面的 ϕ^T 是 $W_k(K)$ 的基函数. 根据定义, 在 $W_k(K)$ 中显然有 $\tilde{C} = C$. 而由 (2.6), 即

$$\Pi_k^\nabla \phi^T = m^T \boldsymbol{\Pi}_{k*}^\nabla,$$

我们有

$$\int_K m_\alpha \Pi_k^\nabla \phi^T \mathrm{d}x = \int_K m_\alpha m^T \mathrm{d}x \boldsymbol{\Pi}_{k*}^\nabla = H(\alpha, :) \boldsymbol{\Pi}_{k*}^\nabla. \tag{2.13}$$

这表明 $\tilde{C}$ 确实可计算.

注 2.3 当 $k=1$ 时, 有

$$\tilde{C}(\alpha, :) = \int_K m_\alpha \Pi_k^\nabla \phi^T \mathrm{d}x, \quad \alpha = 1, 2, 3.$$

可以证明, 当 $k = 1, 2$ 时,

$$\Pi_k^\nabla w_h = \Pi_k^0 w_h, \quad w_h \in W_k(K),$$

这里对 $k=2$, Π_2^∇ 的唯一性限制条件改为

$$\int_K v_h \mathrm{d}x = \int_K \Pi_2^\nabla v_h \mathrm{d}x, \tag{2.14}$$

而不是用边界积分. 这样看来, L^2 投影似乎不需要考虑计算问题, 因为已经知道

$$\boldsymbol{\Pi}_k^0 = \boldsymbol{\Pi}_k^\nabla, \quad \boldsymbol{\Pi}_{k*}^0 = \boldsymbol{\Pi}_{k*}^\nabla.$$

实则不然, 在双线性形式的表示中仍要计算矩阵 H. 类似椭圆投影, 我们有如下的一致性关系

引理 2.2 在 $W_k(K)$ 中, 有 $C = \tilde{C}$, 从而

$$H = CD = \tilde{C}D.$$

证明 直接计算有

$$H = \int_K mm^T \mathrm{d}x = \int_K m\phi^T D \mathrm{d}x = \int_K m\phi^T \mathrm{d}x D = CD.$$

□

注 2.4 由 (2.13) 可知, $\tilde{C}$ 的计算实际上需要用到 H 的若干行. 因此, L^2 投影的一致性关系用处不大.

下面考虑如何计算单元上的积分. 设单元的重心为 z_0, 顶点分别为 $z_1, \cdots, z_{N_v}$, 连接重心与顶点, 则给出单元 K 的一个三角剖分, 共有 N_v 个小三角形. 这样, K 上的积分转化为这些小三角形上的积分之和. iFEM 中提供了三角形上的 Gauss 求积节点与权重, 即 `quadpts.m`. 我们简单说明一下其用法. 那里的权重和节点实际上是针对面积坐标下的参考三角形进行的, 积分公式为

$$\iint_{\hat{T}} f(\lambda_1, \lambda_2, \lambda_3) \mathrm{d}\hat{\sigma} \approx |\hat{T}| \sum_{i=1}^{n_g} w_i f(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i}), \quad |\hat{T}| = \frac{1}{2},$$

其中,

$$\begin{cases} x = x_1\lambda_1 + x_2\lambda_2 + x_3\lambda_3 \\ y = y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3 \end{cases}, \qquad \lambda_1 + \lambda_2 + \lambda_3 = 1, \tag{2.15}$$

注意到

$$\det\left(\frac{\partial(x, y)}{\partial(\lambda_1, \lambda_2)}\right) = 2S,$$

这里 S 是三角形 T 的代数面积, 我们有

$$\int_T F(x, y)\mathrm{d}\sigma = 2\,|T| \int_T f(\lambda_1, \lambda_2, \lambda_3)\mathrm{d}\hat{\sigma} = |T| \sum_{i=1}^{n_g} w_i f(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i}).$$

因变换前后点出的值不变, 故

$$\int_T F(x, y)\mathrm{d}\sigma = |T| \sum_{i=1}^{n_g} w_i F(x_i, y_i).$$

利用 (2.15) 可把参考元上的 Gauss 点 $(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i})$ 转化为 T 上的点. 对 VEM 的单元 K, 可给出其三角剖分的基本数据结构, 记为 `nodeT, elemT`, 这样可如下进行 K 上的积分.

```
function Int = integralTri(fun,n,nodeT,elemT)
%% integralTri: approximate integrals in a polygonal domain
% with trianguation (nodeT,elemT).
% n: n-th order quadrature rule
% fun: one or more anonymous functions, e.g. fun = @(x,y) [f1(x,y), f2(x,y)]

[lambda,weight] = quadpts(n);
NT = size(elemT,1); Int = 0;
for iel = 1:NT
    vT = nodeT(elemT(iel,:),:);
    area = 0.5*abs(det([[1;1;1],vT]));
    xy = lambda*vT;
    f = fun(xy(:,1),xy(:,2));
    Int = Int + area*weight*f;
end
```

注意 fun 容许是多个按行排列的匿名函数.

我们把 $k=1$ 时的 H 添加到之前的 VEM_MAT_Poisson.m 中, 如下

CODE 11. VEM_MAT_Poisson.m

```
function [D,Bs,G,Gs,H] = VEM_MAT_Poisson(aux)

% aux = auxgeometry(node,elem);

node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid;
diameter = aux.diameter;
NT = size(elem,1);

D = cell(NT,1);
% B = cell(NT,1);  % it is not used in the computation
Bs = cell(NT,1);
G = cell(NT,1);  Gs = cell(NT,1); H = cell(NT,1);
for iel = 1:NT
    index = elem{iel};     Nv = length(index);
    xK = elemCentroid(iel,1); yK = elemCentroid(iel,2); hK = diameter(iel);

    m1 = @(x,y) 1 * ones(size(x));
    m2 = @(x,y) (x-xK)./hK;
    m3 = @(x,y) (y-yK)./hK;

    m = @(x,y) [m2(x,y), m3(x,y)];  % m2,m3
    x = node(index,1); y = node(index,2);

    % D
    D1 = zeros(Nv,3); D1(:,1) = 1;
    D1(:,2:3) = m(x,y);   D{iel} = D1;

    % B, Bs, G, Gs
    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
    Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of edge vector
    B1 = Gradm*normVec; B1s = B1; B1s(1,:) = 1/Nv;
    % B{iel} = B1;
    Bs{iel} = B1s;
    G{iel} = B1*D1;     Gs{iel} = B1s*D1;

    % H
    nodeT = [node(index,:);elemCentroid(iel,:)];
    elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
    m = @(x,y) [m1(x,y).*m1(x,y), m1(x,y).*m2(x,y), m1(x,y).*m3(x,y), ...
                m2(x,y).*m1(x,y), m2(x,y).*m2(x,y), m2(x,y).*m3(x,y), ...
                m3(x,y).*m1(x,y), m3(x,y).*m2(x,y), m3(x,y).*m3(x,y)];
    H1 = zeros(3,3);
    H1(:) = integralTri(m,2,nodeT,elemT); % n = 2
```

```
    H{iel} = H1;
end
```

注 2.5 当 m_α 较多时, H 可直接用循环求, 如

```
    m1 = @(x,y) 1 * ones(size(x));
    m2 = @(x,y) (x-xK)./hK;
    m3 = @(x,y) (y-yK)./hK;
    m = {m1,m2,m3};

    H1 = zeros(3,3);
    for i = 1:3
        for j = 1:3
            fun = @(x,y) m{i}(x,y).*m{j}(x,y);
            H1(i,j) = integralTri(fun,2,nodeT,elemT);
        end
    end
    H{iel} = H1;
```

2.5 Poisson 方程一阶 VEM 程序

2.5.1 问题描述

考虑更一般的问题

$$\begin{cases} -\Delta u + cu = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma_D, \\ \partial_n u = g_N & \text{on } \Gamma_N, \end{cases}$$

虚拟元方法为: 找 $u_h \in V_h^g$, 使得

$$a_h(u_h, v) + b_h(u_h, v) = \ell_h(v), \quad v \in V_h,$$

其中,

- 空间为

$$V_h = \left\{v \in H_0^1(\Omega) \cap C(\overline{\Omega}): \quad v|_K \in V_k(K), \quad K \in \mathcal{T}_h\right\},$$

$$V_h^g = \left\{v \in H^1(\Omega) \cap C(\bar{\Omega}): \quad v|_K \in V_k(K), \quad K \in \mathcal{T}_h, \quad v|_{\Gamma_D} = g_D\right\}.$$

- 双线性形式为 $a_K^h(u, v)$ 同前, 而

$$b_h(u, v) = \sum_{K \in \mathcal{T}_h} b_K^h(u, v), \quad b_K^h(u, v) = c \int_K \Pi_k^0 u \cdot \Pi_k^0 v \mathrm{d}x.$$

- 右端为

$$\ell_h(v) = F_h(v) + \int_{\Gamma_N} g_N v \mathrm{d}s.$$

注意, 对 $k = 1$, 自由度 $\chi_i(u_h)$ 是节点值. 这与常规情形没有什么区别, Dirichlet 条件可以直接处理; 而 v 在边界上是一次多项式, Neumann 边界条件可按一维有限元问题进行装配.

2.5.2 单元刚度矩阵的计算

为了方便, 令

$$A_K^1(i,j) = a_K(\Pi_k^\nabla \phi_j, \Pi_k^\nabla \phi_i) = \int_K \nabla(\Pi_k^\nabla \phi_j)\cdot\nabla(\Pi_k^\nabla \phi_i)\mathrm{d}x,$$

$$A_K^2(i,j) = S_K(\phi_j - \Pi_k^\nabla \phi_j, \phi_i - \Pi_k^\nabla \phi_i) = \sum_{r=1}^{N_k} \chi_r(\phi_j - \Pi_k^\nabla \phi_j)\chi_r(\phi_i - \Pi_k^\nabla \phi_i).$$

第一式写成矩阵形式为

$$A_K^1 = \int_K \nabla\Pi_k^\nabla \phi\cdot\nabla\Pi_k^\nabla \phi^T \mathrm{d}x = (\mathbf{\Pi}_{k*}^\nabla)^T \int_K \nabla m\cdot\nabla m^T \mathrm{d}x \mathbf{\Pi}_{k*}^\nabla = (\mathbf{\Pi}_{k*}^\nabla)^T G \mathbf{\Pi}_{k*}^\nabla,$$

至于第二式, 由 (2.4) 知, $\chi_r(\Pi_k^\nabla \phi_i) = (\mathbf{\Pi}_k^\nabla)_{ri}$, 于是

$$A_K^2(i,j) = \sum_{r=1}^{N_v} (\boldsymbol{I} - \mathbf{\Pi}_k^\nabla)_{ri}(\boldsymbol{I} - \mathbf{\Pi}_k^\nabla)_{rj} = \Big[(\boldsymbol{I} - \mathbf{\Pi}_k^\nabla)^T(\boldsymbol{I} - \mathbf{\Pi}_k^\nabla)\Big]_{ij}$$

或

$$A_K^2 = (\boldsymbol{I} - \mathbf{\Pi}_k^\nabla)^T(\boldsymbol{I} - \mathbf{\Pi}_k^\nabla).$$

类似有,

$$B_K = \int_K \Pi_k^0 \phi\cdot\Pi_k^0 \phi^T \mathrm{d}x = (\mathbf{\Pi}_{k*}^0)^T \int_K m m^T \mathrm{d}x \mathbf{\Pi}_{k*}^0 = (\mathbf{\Pi}_{k*}^0)^T H \mathbf{\Pi}_{k*}^0,$$

而

$$\mathbf{\Pi}_{k*}^0 = \mathbf{\Pi}_{k*}^\nabla, \quad k = 1.$$

如下给出单元刚度矩阵

```
% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    Pis = Gs{iel}\Bs{iel};    Pi  = D{iel}*Pis;    I = eye(size(Pi));

    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
    BK = pde.c*Pis'*H{iel}*Pis;  % Pis = Pi0s;
    ABelem{iel} = reshape(AK+BK,1,[]); % straighten as row vector for easy ...
        assembly
end
```

注 2.6 这里对单元刚度矩阵进行了行拉直, 见后面装配的说明.

2.5.3 单元载荷向量的计算

方法 1: 右端的

$$F_K^h(v) = \int_K \Pi_{k-2}^0 f v \mathrm{d}x,$$

当 $k=1$ 时, 规定 $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (2.9) 定义, 即单元顶点值的平均. 注意到

$$F_K^h(v) = \int_K \Pi_0^0 f v \mathrm{d}x = \int_K \Pi_0^0 f \Pi_k^0 v \mathrm{d}x,$$

我们有单元载荷向量为

$$F_K = \int_K \Pi_0^0 f \Pi_0^0 \phi \mathrm{d}x = f(x_K, y_K)\,|K|\,\frac{1}{N_v}[1, \cdots, 1]_{N_k}^T,$$

这里为了方便, $\Pi_0^0 f$ 用中点值近似.

```
% Load vector
% % method 1
Nv = length(elem{iel}); Nk = Nv;
fK = pde.f(elemCentroid(iel,:))*area(iel)/Nv*ones(Nk,1);
```

方法 2: 在 $W_k(K)$ 中, L^2 投影 Π_k^0 是可计算的, 为此右端的近似也可改为

$$F_K^h(v) = \int_K \Pi_k^0 f v \mathrm{d}x.$$

注意到

$$F_K^h(v) = \int_K \Pi_k^0 f v \mathrm{d}x = \int_K \Pi_k^0 f \Pi_k^0 v \mathrm{d}x = \int_K f \Pi_k^0 v \mathrm{d}x,$$

此时单元载荷向量

$$F_K = \int_K f \Pi_k^0 \phi \mathrm{d}x = (\boldsymbol{\Pi}_{k*}^0)^T \int_K f m \mathrm{d}x, \tag{2.16}$$

这里用到 (2.12). 右端的积分若采用中点型积分公式近似, 则有

$$F_K = \int_K f \Pi_k^0 \phi \mathrm{d}x = (\boldsymbol{\Pi}_{k*}^0)^T \int_K f m \mathrm{d}x = (\boldsymbol{\Pi}_{k*}^0)^T \begin{bmatrix} f(x_K, y_K)\,|K| \\ 0 \\ 0 \end{bmatrix}.$$

```
% Load vector
% % method 2
fK = Pis'*[pde.f(elemCentroid(iel,:))*area(iel);0;0];
```

方法 3: 当然, (2.16) 的右端也可用 Gauss 积分近似.

```
% Load vector
% % method 3
index = elem{iel};      Nv = length(index);
xK = elemCentroid(iel,1); yK = elemCentroid(iel,2); hK = diameter(iel);
m1 = @(x,y) 1*ones(size(x)); m2 = @(x,y) (x-xK)./hK; m3 = @(x,y) ...
    (y-yK)./hK;
nodeT = [node(index,:);elemCentroid(iel,:)];
elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
m = @(x,y) pde.f([x,y]).*[m1(x,y), m2(x,y), m3(x,y)]; % f(p) = f([x,y])
rhs = integralTri(m,2,nodeT,elemT); rhs = rhs';
fK = Pis'*rhs;
```

注 2.7 这里对载荷向量进行了行拉直, 见后面装配的说明. 三种方法误差量级一致, 本文采用第二种. 注意, 对某些问题, 若没有定义 L^2 投影 Π_k^0, 则使用第一种; 若定义了 Π_k^0, 且是高阶方法, 则用第三种.

最终的单元刚度矩阵和载荷向量如下计算

```
% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    % Projection
    Pis = Gs{iel}\Bs{iel};   Pi  = D{iel}*Pis;   I = eye(size(Pi));
    % Stiffness matrix
    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
    BK = pde.c*Pis'*H{iel}*Pis;  % Pis = Pi0s;
    ABelem{iel} = reshape(AK+BK,1,[]); % straighten as row vector for easy ...
        assembly
    % Load vector
    fK = Pis'*[pde.f(elemCentroid(iel,:))*area(iel);0;0];
    belem{iel} = fK'; % straighten as row vector for easy assembly
end
```

2.5.4 刚度矩阵与载荷向量的装配

先考虑三角剖分. $\Delta z_1 z_2 z_3$ 的局部整体对应如下

$$\{1,2,3\}\,(\text{local}) \qquad \to \qquad \{z_1,z_2,z_3\}\,(\text{global}),$$

这里 z_1,z_2,z_3 是三角形顶点的整体编号, 所有三角形的可如下实现

```
z1 = elem(:,1); % column vector
z2 = elem(:,2);
z3 = elem(:,3);
```

- 对固定单元, 设单元刚度矩阵为

$$[K^e] \;=\; \begin{array}{c|ccc} & u_i & u_j & u_l \\ \hline u_i & k_{11} & k_{12} & k_{13} \\ u_j & k_{21} & k_{22} & k_{23} \\ u_l & k_{31} & k_{32} & k_{33} \end{array}\,,$$

这里左侧表示整体刚度矩阵的行指标, 上侧表示列指标. 对 k_{12}, 我们要把它放在 (i,j) 位置, 即在 (z_1,z_2) 处赋值 k_{12}, 这可用稀疏矩阵函数 `sparse` 完成. 它的用法如下

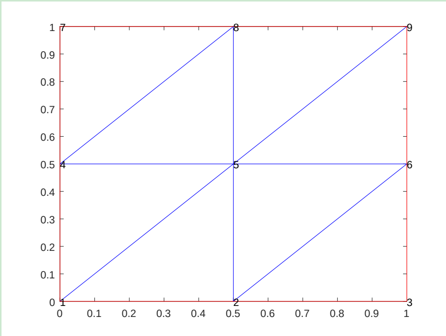
```
S = sparse(i,j,s,m,n,nzmax);
```

它分配了一个 $m\times n$ 的稀疏矩阵, 且最多有 nzmax 个非零分量, 其中 `i,j` 是向量, 对应分量指出非零值的位置, `s` 则是非零值的向量 (nzmax 可以不给出). 因此, 若想在 (z_1,z_2) 处赋值 k_{12}, 可以如下操作

```
N = size(node,1); A = sparse(z1,z2,k12,N,N);
```

该命令把所有单元刚度矩阵的 (1,2) 处的值放到了整体刚度矩阵的对应位置中 (`k12` 是列向量, 存储所有单元的).

- 我们自然会有这样的疑问: 如果有两个单元刚度矩阵 (1,2) 处的值对应相同的整体指标 (i,j), 那么它们应该相加, 而不是简单的赋值. 事实上, 前面已经提到, 在 MATLAB 中, `sparse` 中出现相同指标规定为相加, 因而如果的确出现, 则本身已经解决. 我们要说的是, 对非对角线元素, 上面提到的情形是不可能出现的.



只需要考虑共享一条边的三角形, 因为出现相同整体指标 (i,j), 意味着单元三角形有共同的边 $i-j$. 不失一般性, 以图中的 1-5 边为例. 根据逆时针规则, 对三角形 $\Delta z_5 z_1 z_2$ 中, 顺序如下

$$
[K^e] \quad = \quad \begin{array}{ccc|c} u_5 & u_1 & u_2 & \\ \hline k_{11} & k_{12} & k_{13} & u_5 \\ k_{21} & k_{22} & k_{23} & u_1 \\ k_{31} & k_{32} & k_{33} & u_2 \end{array} \;,
$$

即 k_{12} 对应整体指标 $(5,1)$. 而对 $\Delta z_1 z_5 z_4$, 一定不会出现 k_{12} 对应 $(5,1)$, 只可能出现 $(1,5)$, $(5,4)$, $(4,1)$. 这表明, 的确不会出现上面考虑的相同整体指标 (i,j) 的情形. 但不管怎么样, 对角线还是会出现相同位置的, 这也许是 MATLAB 中规定 `sparse` 相同指标函数值相加的原因.

- 上面的分析表明, 我们只要把单元刚度矩阵相同局部位置的三元组向量 `(i,j,s)` 获得, 用 `sparse(i,j,s,N,N)` 就处理好了该局部位置. 其他位置同样处理. 一个快速的装配方式是把所有局部位置的 `(i,j,s)` 拼接在一起实现, 即

$$
\begin{bmatrix} \mathbf{i}_{11} & \mathbf{j}_{11} & \mathbf{s}_{11} \\ \mathbf{i}_{12} & \mathbf{j}_{12} & \mathbf{s}_{12} \\ \vdots & \vdots & \vdots \\ \mathbf{i}_{33} & \mathbf{j}_{33} & \mathbf{s}_{33} \end{bmatrix}
$$

这里按行优先的顺序排列. 注意每个 $\mathbf{i}_{ij}$ 都对应 `NT` 个单元, 从而 $\mathbf{i}$ 共有 $NT \times N_{dof}^2$ 个元素, 其中, $N_{dof} = 3$ 表示单元的自由度个数. 核心代码如下

CODE 12. sparse 装配指标

```
NT = size(elem,1); Ndof = 3; nnz = NT*Ndof^2;
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
id = 0;
for i = 1:Ndof
    for j = 1:Ndof
        ii(id+1:id+NT) = elem(:,i);   % zi
```

```
        jj(id+1:id+NT) = elem(:,j);   % zj
        ss(id+1:id+NT) = K(:,i,j);    % kij
        id = id + NT;
    end
end
```

这里假设 (i,j) 位置的单元刚度矩阵值用三维数组存储.

注 2.8 可用向量法给出 ii, jj, 考虑到只是简单的赋值, 这里不这样处理, 以体现直观. 也可事先将每个单元刚度矩阵按行拉成一行, 然后逐个单元堆成一个矩阵, 则 ss 可由该矩阵按列拉直得到. 后面将采用这种处理, 为此在生成单元刚度矩阵时, 我们进行行拉直.

- 用稀疏矩阵, 右端可如下实现

```
b = sparse(z1,1,F1,N,1);
b = b+sparse(z2,1,F2,N,1);
b = b+sparse(z3,1,F3,N,1);
```

这里, F1 是所有单元载荷第 1 个位置的元素组成的向量. 当然也可用

```
ii = elem(:); jj = 1; ss = [F1;F2;F3]; % elem(:) = [z1;z2;z3]
b = sparse(ii,jj,ss,N,1);
```

b 一般不是稀疏的, 存储稀疏的形式访问起来会麻烦. 我们可用下面的语句代替

```
b = accumarray(elem(:),[F1;F2;F3],[N 1]);
```

同理, 右端向量也按行拉直存储.

上面的思路只适合相同类型的单元剖分. 对一般的多角形剖分, 我们可对不同构型进行, 即把相同单元的放在一起, 逐一考虑.

- 首先可找到所有的构型

  `elemLen = cellfun('length',elem); vertNum = unique(elemLen);`

  假设现在只有 4,5,6,7 边形, 且分别有 N4, N5, N6, N7 个.

- 对 4 边形, 用 `Nv = 4; idNv = find(elemLen==Nv);` 可找到所有 4 边形的序号. 比如, 第 67, 82 这两个单元是 4 边形. 对这两个相同构型的单元, 我们就可按照之前的思路进行装配, 找到相应的三元组, 记为 $(\mathbf{i}_4,\mathbf{j}_4,\mathbf{s}_4)$. 类似可找到其他构型的, 分别记为 $(\mathbf{i}_5,\mathbf{j}_5,\mathbf{s}_5)$, $(\mathbf{i}_6,\mathbf{j}_6,\mathbf{s}_6)$, $(\mathbf{i}_7,\mathbf{j}_7,\mathbf{s}_7)$. 由此就可生成最终的三元组

$$(\mathbf{i},\mathbf{j},\mathbf{s}) := \begin{bmatrix} \mathbf{i}_4 & \mathbf{j}_4 & \mathbf{s}_4 \\ \mathbf{i}_5 & \mathbf{j}_5 & \mathbf{s}_5 \\ \mathbf{i}_6 & \mathbf{j}_6 & \mathbf{s}_6 \\ \mathbf{i}_7 & \mathbf{j}_7 & \mathbf{s}_7 \end{bmatrix}.$$

  注意, nnz 是所有单元刚度矩阵的元素个数之和, 即 `nnz = sum(elemLen.^2)`.

装配程序如下

CODE 13. Poisson 方程 VEM 的 sparse 装配指标

```
% --------- Assemble the stiffness matrix and load vector ---------
elemLen = cellfun('length',elem); vertNum = unique(elemLen);
nnz = sum(elemLen.^2);
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);

id = 0; ff = zeros(N,1);
for Nv = vertNum(:)' % only valid for row vector

    % assemble the matrix
    idNv = find(elemLen == Nv); % find polygons with Nv vertices
    NTv = length(idNv); % number of elements with Nv vertices

    elemNv = cell2mat(elem(idNv)); % elem
    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
    s = 1; Ndof = Nv;
    for i = 1:Ndof
        for j = 1:Ndof
            ii(id+1:id+NTv) = elemNv(:,i); % zi
            jj(id+1:id+NTv) = elemNv(:,j); % zj
            ss(id+1:id+NTv) = K(:,s);
            id = id + NTv; s = s+1;
        end
    end

    % assemble the vector
    ff = ff +  accumarray(elemNv(:),F(:),[N 1]);
end
kk = sparse(ii,jj,ss,N,N);
```

2.5.5 边界条件的处理

下面考虑 Neumann 边界条件. 用梯形公式近似 (或用中心格式), 例如

$$\int_{\Gamma^e}\frac{\partial u}{\partial n}\phi_1 \mathrm{d}s = \frac{h_e}{2}\left(\frac{\partial u}{\partial n}(z_1)\phi_1(z_1)+\frac{\partial u}{\partial n}(z_2)\phi_1(z_2)\right) = \frac{h_e}{2}\frac{\partial u}{\partial n}(z_1) = \frac{h_e q(z_1)}{2},$$

$$\int_{\Gamma^e}\frac{\partial u}{\partial n}\phi_2 \mathrm{d}s = \frac{h_e}{2}\left(\frac{\partial u}{\partial n}(z_1)\phi_2(z_1)+\frac{\partial u}{\partial n}(z_2)\phi_2(z_2)\right) = \frac{h_e}{2}\frac{\partial u}{\partial n}(z_2) = \frac{h_e q(z_2)}{2},$$

其中

$$h_e = |z_j - z_i| = \sqrt{(x_j-x_i)^2+(y_j-y_i)^2}.$$

为此, 我们必须给出在每条 Neumann 边 Γ^e 上 $g_N = \partial_n u$ 的值. 注意到

$$h_e\frac{\partial u}{\partial n} = h_e\nabla u\cdot\vec{n} = \nabla u\cdot(h_e\vec{n}),$$

而 $h_e\vec{n}$ 恰是定向边 e 顺时针旋转 $90°$ 或 $-e$ 逆时针旋转 $90°$ 所得. 这样, 设 $-e$ 的坐标为 (a,b), 则 $h_e\vec{n}$ 的坐标为 $(-b,a)$.

我们在 Poissondata.m 定义 $g_N = [u_x(x,y),\ u_y(x,y)]$, 结合一维问题的装配算法, 我们有 Neumann 边界条件的如下处理.

```
% ------------ Neumann boundary condition ----------------
elemN = bdStruct.elemN;
if ¬isempty(elemN)
    g_N = pde.g_N;
    z1 = node(elemN(:,1),:); z2 = node(elemN(:,2),:);
    e = z1-z2;  % e = z2-z1
    ne = [-e(:,2),e(:,1)]; % scaled ne
    F1 = sum(ne.*g_N(z1),2)./2;
    F2 = sum(ne.*g_N(z2),2)./2;
    FN = [F1,F2];
    ff = ff + accumarray(elemN(:), FN(:),[N 1]);
end
```

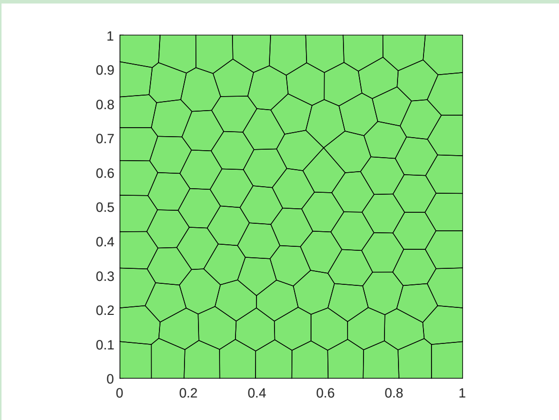
Dirichlet 边界条件比较容易, 如下

```
% ------------ Dirichlet boundary condition ----------------
g_D = pde.g_D;  eD = bdStruct.eD;
isBdNode = false(N,1); isBdNode(eD) = true;
bdNode = find(isBdNode); freeNode = find(¬isBdNode);
pD = node(bdNode,:);
u = zeros(N,1); uD = g_D(pD); u(bdNode) = uD(:);
ff = ff - kk*u;
```

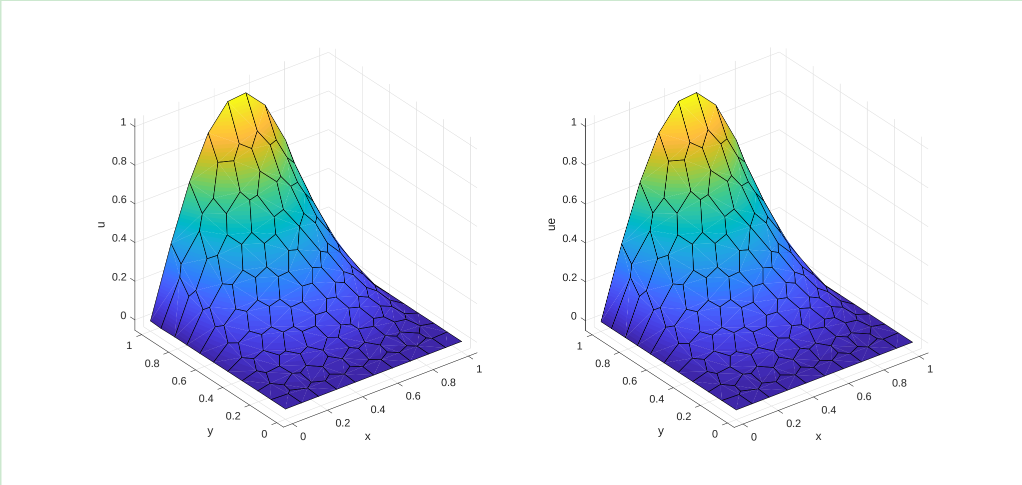
2.5.6 程序整理

例 2.1 设精确解为 $u(x,y)=y^2\sin(\pi x)$, $\Omega=(0,1)^2$.

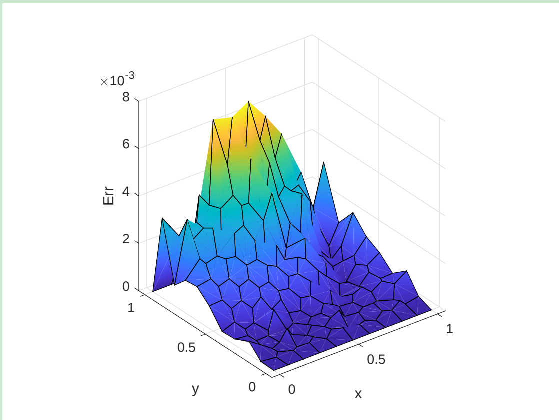
网格剖分由 tool 文件夹下的 meshfun.m 生成, 保存为 meshdata.mat, 网格图示如下



设区域的左侧和右侧为 Neumann 边界, 数值解和精确解的图像如下



绝对误差为



主程序为

CODE 14. main_PoissonVEM.m

```
clc;clear;close all
tic;
% ------------- Mesh --------------------
% generated by meshfun.m in tool
load meshdata.mat;
figure, showmesh(node,elem);
% findnode(node); % findelem(node,elem);

% ---------------- PDE data ----------------------
pde = Poissondata();
bdNeumann = 'abs(x-1)<1e-4 | abs(x-0)<1e-4'; % string for Neumann
bdStruct = setboundary(node,elem,bdNeumann);

% ------- Solve the problem -----
u = PoissonVEM(node,elem,pde,bdStruct);

% ------- error analysis ----------
uexact = pde.uexact; ue = uexact(node);
figure,
```

```
subplot(1,2,1), showsolution(node,elem,u);  zlabel('u');
subplot(1,2,2), showsolution(node,elem,ue); zlabel('ue');
Eabs = abs(u-ue);
figure,showsolution(node,elem,Eabs); zlim('auto'); zlabel('Err');
format shorte;
Err = norm(Eabs)/norm(ue)
toc
```

函数文件为

CODE 15. PoissonVEM.m

```
function u = PoissonVEM(node,elem,pde,bdStruct)

aux = auxgeometry(node,elem);
node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid; area = aux.area;

N = size(node,1);  NT = size(elem,1);
% ----------------- D,Bs,G,Gs,H ---------------------------
[D,Bs,G,Gs,H] = VEM_MAT_Poisson(aux);

% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    % Projection
    Pis = Gs{iel}\Bs{iel};   Pi  = D{iel}*Pis;   I = eye(size(Pi));
    % Stiff matrix
    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
    BK = pde.c*Pis'*H{iel}*Pis;  % Pis = Pi0s;
    ABelem{iel} = reshape(AK+BK,1,[]); % straighten as row vector for easy ...
        assembly
    % Load vector
    fK = Pis'*[pde.f(elemCentroid(iel,:))*area(iel);0;0];
    belem{iel} = fK'; % straighten as row vector for easy assembly
end

% --------- Assemble the stiff matrix and load vector ---------
elemLen = cellfun('length',elem); vertNum = unique(elemLen);
nnz = sum(elemLen.^2);
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);

id = 0; ff = zeros(N,1);
for Nv = vertNum(:)' % only valid for row vector

    % assemble the matrix
    idNv = find(elemLen == Nv); % find polygons with Nv vertices
    NTv = length(idNv); % number of elements with Nv vertices

    elemNv = cell2mat(elem(idNv)); % elem
    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
```

```
    s = 1; Ndof = Nv;
    for i = 1:Ndof
        for j = 1:Ndof
            ii(id+1:id+NTv) = elemNv(:,i); % zi
            jj(id+1:id+NTv) = elemNv(:,j); % zj
            ss(id+1:id+NTv) = K(:,s);
            id = id + NTv; s = s+1;
        end
    end

    % assemble the vector
    ff = ff +  accumarray(elemNv(:),F(:),[N 1]);
end
kk = sparse(ii,jj,ss,N,N);

% ------------ Neumann boundary condition ----------------
elemN = bdStruct.elemN;
if ¬isempty(elemN)
    g_N = pde.g_N;
    z1 = node(elemN(:,1),:); z2 = node(elemN(:,2),:);
    e = z1-z2;  % e = z2-z1
    ne = [-e(:,2),e(:,1)]; % scaled ne
    F1 = sum(ne.*g_N(z1),2)./2;
    F2 = sum(ne.*g_N(z2),2)./2;
    FN = [F1,F2];
    ff = ff + accumarray(elemN(:), FN(:),[N 1]);
end

% ------------ Dirichlet boundary condition ----------------
g_D = pde.g_D;  eD = bdStruct.eD;
isBdNode = false(N,1); isBdNode(eD) = true;
bdNode = find(isBdNode); freeNode = find(¬isBdNode);
pD = node(bdNode,:);
u = zeros(N,1); uD = g_D(pD); u(bdNode) = uD(:);
ff = ff - kk*u;

% ------------------ Solver -------------------
u(freeNode) = kk(freeNode,freeNode)\ff(freeNode);
```

注 2.9 涉及到的其他函数文件见 GitHub. 上面的主程序也可直接改为其他区域的例子. 例如, 我们给出了单位圆的剖分数据 meshdataCircle.m, 可规定上半圆部分为 Neumann 边界:

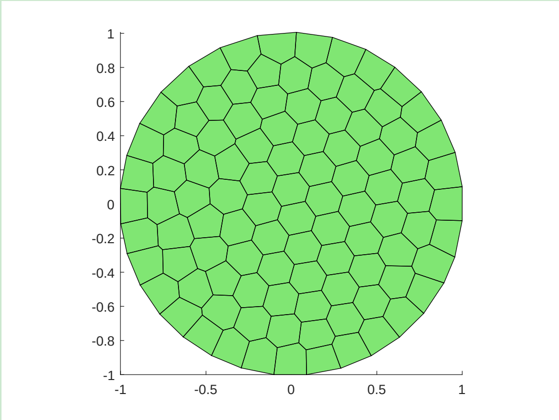
```
bdNeumann = '(abs(x.^2+y.^2-1)<1e-1)& (y≥0)'; %string for Neumann (circle)
```

或直接写为

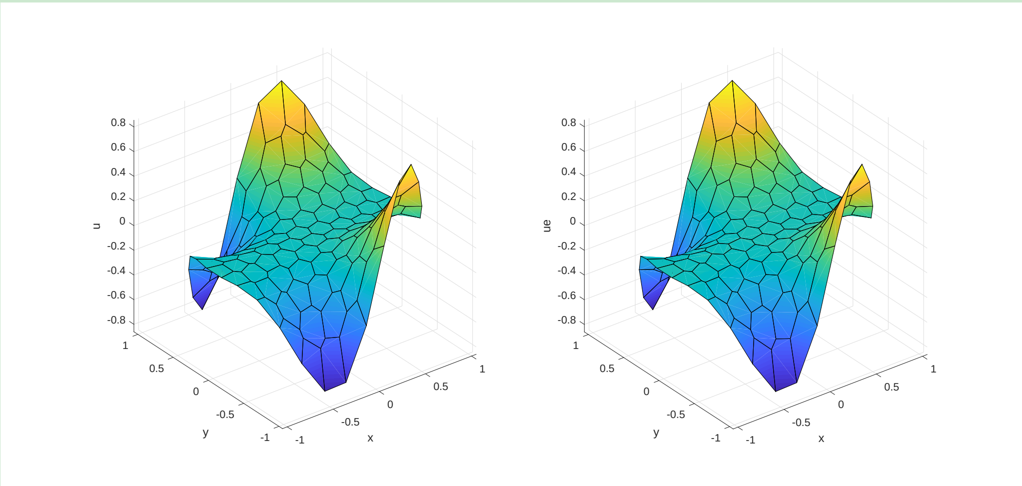
```
bdNeumann = 'y≥0'; %string for Neumann (circle)
```

这种写法更好, 因为对较粗的剖分, 边界上的部分离单位圆边界差很远, 而且我们的判断条件是针对剖分的边界 `bdEdge` 进行的.

网格剖分如下



数值解与精确解为



3 线弹性边值问题的一阶虚拟元方法

3.1 线弹性边值问题简介

3.1.1 问题说明

设弹性体未受外力时所在区域为 $\Omega \subset \mathbb{R}^3$. 当它受体力 $\boldsymbol{f}$ (在 Ω 中), 在 Ω 的一部分边界 Γ_1 受表面力 $\boldsymbol{g}$, 而在另一部分边界 Γ_0 上固定位移 $\boldsymbol{u} = \boldsymbol{0}$ 时, 弹性体就产生位移 $\boldsymbol{u}$. 在平衡状态下, 位移 $\boldsymbol{u}$ 所满足的边值问题为

$$\begin{cases} -\partial_j \sigma_{ij}(\boldsymbol{u}) = f_i, \quad i = 1,2,3 & \text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on} \quad \Gamma_0, \\ \sigma_{ij}(\boldsymbol{u}) n_j = g_i, \quad i = 1,2,3 & \text{on} \quad \Gamma_1, \end{cases} \tag{3.17}$$

这里**约定: 凡在每一项中指标重复出现意味着从 1 到 3 (三维) 或 2 (二维) 求和**. 上面的方程也可写为

$$\begin{cases} -\operatorname{div} \boldsymbol{\sigma} = \boldsymbol{f} & \text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on} \quad \Gamma_0, \\ \boldsymbol{\sigma} \boldsymbol{n} = \boldsymbol{g} & \text{on} \quad \Gamma_1, \end{cases}$$

其中向量规定为列向量, 而 $\operatorname{div} \boldsymbol{\sigma}$ 是对矩阵 $\boldsymbol{\sigma}$ 的每行进行 (注意它是对称矩阵).

在以上式子中, 第一式为平衡方程, 其中的 σ_{ij} 为应力张量, 它与应变张量 $\varepsilon_{ij}(\boldsymbol{u})$ 满足如下的本构关系 (均匀的各项同性弹性体的 Hooke 定律)

$$\sigma_{ij}(\boldsymbol{u}) = \sigma_{ji}(\boldsymbol{u}) = \lambda \varepsilon_{kk}(\boldsymbol{u}) \delta_{ij} + 2\mu \varepsilon_{ij}(\boldsymbol{u}), \tag{3.18}$$

$$\varepsilon_{ij}(\boldsymbol{u}) = \varepsilon_{ji}(\boldsymbol{u}) = \frac{1}{2}(\partial_j u_i + \partial_i u_j), \tag{3.19}$$

而 λ 和 μ 为 Lamé 系数. 注意, 这里 $\varepsilon_{kk}(\boldsymbol{u})$ 按规定是对指标求和的, 显然有 (标量)

$$\varepsilon_{kk}(\boldsymbol{u}) = \partial_k u_k = \operatorname{div} \boldsymbol{u}.$$

这样, 平衡方程也可写为如下的紧凑形式

$$-\operatorname{div}\, (\lambda (\operatorname{div} \boldsymbol{u}) \boldsymbol{I} + 2\mu \boldsymbol{\varepsilon}(\boldsymbol{u})) = \boldsymbol{f} \quad \text{in} \quad \Omega.$$

把 (3.18)-(3.19) 代入 (3.17), 可获得平衡方程的另一形式

$$-\mu \Delta \boldsymbol{u} - (\lambda + \mu) \operatorname{grad}(\operatorname{div} \boldsymbol{u}) = \boldsymbol{f} \quad \text{in} \quad \Omega, \tag{3.20}$$

有时候会直接考虑该方程, 因为其中每个算子都是熟悉的.

3.1.2 连续变分问题

设

$$V = \left\{ \boldsymbol{v} \in H^1(\Omega)^3 : \ \boldsymbol{v} = \boldsymbol{0} \quad \text{on} \ \ \Gamma_0 \right\}, \tag{3.21}$$

令 $\boldsymbol{v} = (v_1, v_2, v_3)^T \in V$, 在 (3.17) 的平衡方程的两边乘以 v_i, 有

$$\int_\Omega -\partial_j \sigma_{ij}(\boldsymbol{u}) v_i \mathrm{d}x = \int_\Omega f_i v_i \mathrm{d}x,$$

这里遵循求和约定, 即上式实际上是求和. 分部积分有

$$-\int_{\partial\Omega}\sigma_{ij}(\boldsymbol{u})v_in_j\mathrm{d}s+\int_\Omega\sigma_{ij}(\boldsymbol{u})\partial_jv_i\mathrm{d}x=\int_\Omega f_iv_i\mathrm{d}x$$

或

$$-\int_{\partial\Omega}g_iv_i\mathrm{d}s+\int_\Omega\sigma_{ij}(\boldsymbol{u})\partial_jv_i\mathrm{d}x=\int_\Omega f_iv_i\mathrm{d}x.$$

由对称性,

$$\sigma_{ij}(\boldsymbol{u})\partial_jv_i=\sigma_{ij}(\boldsymbol{u})\frac{1}{2}(\partial_jv_i+\partial_iv_j)=\sigma_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v}),$$

故

$$\int_\Omega\sigma_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})\mathrm{d}x=\int_\Omega\boldsymbol{f}\cdot\boldsymbol{v}\mathrm{d}x+\int_{\Gamma_1}\boldsymbol{g}\cdot\boldsymbol{v}\mathrm{d}s.$$

于是, 变分问题为: 求 $\boldsymbol{u}\in V$ 使得,

$$a(\boldsymbol{u},\boldsymbol{v})=\ell(\boldsymbol{v}),\quad \boldsymbol{v}\in V,$$

式中,

$$a(\boldsymbol{u},\boldsymbol{v})=\int_\Omega\sigma_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})\mathrm{d}x,\qquad \ell(\boldsymbol{v})=\int_\Omega\boldsymbol{f}\cdot\boldsymbol{v}\mathrm{d}x+\int_{\Gamma_1}\boldsymbol{g}\cdot\boldsymbol{v}\mathrm{d}s.$$

文献中一般习惯引入如下记号

$$\boldsymbol{A}:\boldsymbol{B}=\sum_{ij}a_{ij}b_{ij},\quad \boldsymbol{A}=(a_{ij}),\quad \boldsymbol{B}=(b_{ij}),$$

从而双线性形式可写为

$$a(\boldsymbol{u},\boldsymbol{v})=\int_\Omega\boldsymbol{\sigma}(\boldsymbol{u}):\boldsymbol{\varepsilon}(\boldsymbol{v})\mathrm{d}x.$$

注意到

$$\begin{aligned}\sigma_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})&=(\lambda\varepsilon_{kk}(\boldsymbol{u})\delta_{ij}+2\mu\varepsilon_{ij}(\boldsymbol{u}))\varepsilon_{ij}(\boldsymbol{v})\\&=(\lambda\partial_ku_k\delta_{ij}+2\mu\varepsilon_{ij}(\boldsymbol{u}))\varepsilon_{ij}(\boldsymbol{v})\\&=2\mu\varepsilon_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})+\lambda\partial_ku_k\varepsilon_{ii}(\boldsymbol{v})\\&=2\mu\varepsilon_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})+\lambda\partial_ku_k\partial_iu_i,\end{aligned}$$

双线性形式还可写为

$$\begin{aligned}a(\boldsymbol{u},\boldsymbol{v})&=2\mu\int_\Omega\varepsilon_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})\mathrm{d}x+\lambda\int_\Omega\partial_iu_i\partial_ju_j\mathrm{d}x,\\&=2\mu\int_\Omega\boldsymbol{\varepsilon}(\boldsymbol{u}):\boldsymbol{\varepsilon}(\boldsymbol{v})\mathrm{d}x+\lambda\int_\Omega(\mathrm{div}\,\boldsymbol{u})(\mathrm{div}\,\boldsymbol{v})\mathrm{d}x.\end{aligned}\tag{3.22}$$

注 3.1 也可采用 (3.20) 的形式, 具体写出来即

$$\begin{cases}-\mu\Delta u_1-(\lambda+\mu)\partial_x(\mathrm{div}\,\boldsymbol{u})=f_1,\\-\mu\Delta u_2-(\lambda+\mu)\partial_y(\mathrm{div}\,\boldsymbol{u})=f_2,\end{cases}$$

注意 $\mathrm{div}\,\boldsymbol{u}$ 是标量. 第一式乘以 v_1, 并分部积分有

$$\begin{aligned}&\mu\left(\int_\Omega\nabla u_1\cdot\nabla v_1\mathrm{d}x-\int_{\partial\Omega}\partial_nu_1v_1\mathrm{d}s\right)\\&\quad-(\lambda+\mu)\left(\int_{\partial\Omega}(\mathrm{div}\,\boldsymbol{u})v_1n_x\mathrm{d}s-\int_\Omega(\mathrm{div}\,\boldsymbol{u})\partial_xv_1\mathrm{d}x\right)=\int_\Omega f_1v_1\mathrm{d}x.\end{aligned}$$

类似可获得第二个式子对应的结果, 将它们相加有

$$\begin{aligned}&\mu\int_\Omega \nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}\mathrm{d}x+(\lambda+\mu)\int_\Omega(\operatorname{div}\boldsymbol{u})(\operatorname{div}\boldsymbol{v})\mathrm{d}x\\&\quad-\mu\int_{\partial\Omega}\partial_n\boldsymbol{u}\cdot\boldsymbol{v}\mathrm{d}s-(\lambda+\mu)\int_{\partial\Omega}(\operatorname{div}\boldsymbol{u})(\boldsymbol{v}\cdot\boldsymbol{n})\mathrm{d}s=\int_\Omega\boldsymbol{f}\cdot\boldsymbol{v}\mathrm{d}x.\end{aligned}\tag{3.23}$$

这里, $\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}=\nabla u_1\cdot\nabla v_1+\nabla u_2\cdot\nabla v_2$.

3.1.3 近似变分形式 I

先考虑 (3.23) 的近似变分形式, 令

$$\begin{aligned}a^K(\boldsymbol{u},\boldsymbol{v})&=\mu\int_K\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}\mathrm{d}x+(\lambda+\mu)\int_K(\operatorname{div}\boldsymbol{u})(\operatorname{div}\boldsymbol{v})\mathrm{d}x\\&=:\mu a_\nabla^K(\boldsymbol{u},\boldsymbol{v})+(\lambda+\mu)a_{\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v})\end{aligned}$$

式中,

$$a_\nabla^K(\boldsymbol{u},\boldsymbol{v})=\int_K\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}\mathrm{d}x,\quad a_{\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v})=\int_K(\operatorname{div}\boldsymbol{u})(\operatorname{div}\boldsymbol{v})\mathrm{d}x.$$

每个分量的虚拟元空间仍取为 $V_k(K)$, 从而向量型问题的虚拟元空间为

$$\boldsymbol{V}_k(K)=(V_k(K))^2.$$

设 $a_\nabla^K(\boldsymbol{u},\boldsymbol{v})$ 和 $a_{\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v})$ 对应的近似双线性形式分别为 $a_{h,\nabla}^K(\boldsymbol{u},\boldsymbol{v})$ 和 $a_{h,\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v})$, 则虚拟元方法的近似双线性形式为

$$a_h^K(\boldsymbol{u},\boldsymbol{v}):=\mu a_{h,\nabla}^K(\boldsymbol{u},\boldsymbol{v})+(\lambda+\mu)a_{h,\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v}).$$

根据文献,

$$\begin{aligned}a_{h,\nabla}^K(\boldsymbol{u},\boldsymbol{v})&=a_\nabla^K(\Pi^\nabla\boldsymbol{u},\Pi^\nabla\boldsymbol{v})+S^K(\boldsymbol{u}-\Pi^\nabla\boldsymbol{u},\boldsymbol{v}-\Pi^\nabla\boldsymbol{v}),\\a_{h,\operatorname{div}}^K(\boldsymbol{u},\boldsymbol{v})&=\int_K\Pi_{k-1}^0(\operatorname{div}\boldsymbol{u})\Pi_{k-1}^0(\operatorname{div}\boldsymbol{v})\mathrm{d}x,\end{aligned}$$

而稳定双线性形式为 $(k=1)$

$$S^K(\boldsymbol{u},\boldsymbol{v})=\sum_{i=1}^{N_{dof}}\mathrm{dof}_i(\boldsymbol{u})\cdot\mathrm{dof}_i(\boldsymbol{v}).$$

稍后解释这里的投影算子.

3.1.4 近似变分形式 II

考虑 (3.22) 的变分问题, 令

$$\begin{aligned}a^K(\boldsymbol{u},\boldsymbol{v})&=2\mu\int_K\varepsilon_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})\mathrm{d}x+\lambda\int_K\partial_iu_i\partial_ju_j\mathrm{d}x\\&=2\mu\int_K\boldsymbol{\varepsilon}(\boldsymbol{u}):\boldsymbol{\varepsilon}(\boldsymbol{v})\mathrm{d}x+\lambda\int_K(\operatorname{div}\boldsymbol{u})(\operatorname{div}\boldsymbol{v})\mathrm{d}x\\&=:2\mu a_\mu^K(\boldsymbol{u},\boldsymbol{v})+\lambda a_\lambda^K(\boldsymbol{u},\boldsymbol{v}),\end{aligned}$$

式中,

$$a_\mu^K(\boldsymbol{u},\boldsymbol{v})=\int_K\varepsilon_{ij}(\boldsymbol{u})\varepsilon_{ij}(\boldsymbol{v})\mathrm{d}x=\int_K\boldsymbol{\varepsilon}(\boldsymbol{u}):\boldsymbol{\varepsilon}(\boldsymbol{v})\mathrm{d}x,$$

于是

$$a_\lambda^K(\boldsymbol{u},\boldsymbol{v}) = \int_K \partial_i u_i \partial_j u_j \mathrm{d}x = \int_K (\mathrm{div}\ \boldsymbol{u})(\mathrm{div}\ \boldsymbol{v})\mathrm{d}x.$$

设 $a_\mu^K(\boldsymbol{u},\boldsymbol{v})$ 和 $a_\lambda^K(\boldsymbol{u},\boldsymbol{v})$ 对应的近似双线性形式分别为 $a_{h,\mu}^K(\boldsymbol{u},\boldsymbol{v})$ 和 $a_{h,\lambda}^K(\boldsymbol{u},\boldsymbol{v})$, 则虚拟元方法的近似双线性形式为

$$a_h^K(\boldsymbol{u},\boldsymbol{v}) := 2\mu a_{h,\mu}^K(\boldsymbol{u},\boldsymbol{v}) + \lambda a_{h,\lambda}^K(\boldsymbol{u},\boldsymbol{v}).$$

根据文献,

$$a_{h,\mu}^K(\boldsymbol{u},\boldsymbol{v}) = a_\mu^K(\Pi^a\boldsymbol{u},\Pi^a\boldsymbol{v}) + S^K(\boldsymbol{u}-\Pi^a\boldsymbol{u},\boldsymbol{v}-\Pi^a\boldsymbol{v}),$$

$$a_{h,\lambda}^K(\boldsymbol{u},\boldsymbol{v}) = \int_K \Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{u})\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{v})\mathrm{d}x,$$

稍后解释这里的投影算子.

3.2 刚度矩阵和载荷向量的装配

3.2.1 矩阵分析法

对 $\boldsymbol{u} = [u_1, u_2]^T$, 为了避免下标的混乱, 我们记 $\overline{u} = u_1$ 和 $\underline{u} = u_2$, 相应的节点基展开为

$$\overline{u} = \overline{u}_1\varphi_1 + \cdots + \overline{u}_N\varphi_N = \Psi\overline{U}, \quad \underline{u} = \underline{u}_1\varphi_1 + \cdots + \underline{u}_N\varphi_N = \Psi\underline{U},$$

其中,

$$\Psi = [\varphi_1, \cdots, \varphi_N]^T, \ \ \overline{U} = [\overline{u}_1, \cdots, \overline{u}_N]^T, \ \ \underline{U} = [\underline{u}_1, \cdots, \underline{u}_N]^T.$$

于是,

$$\boldsymbol{u} = \sum_{i=1}^N \overline{u}_i\overline{\varphi}_i + \sum_{i=1}^N \underline{u}_i\underline{\varphi}_i,$$

式中,

$$\overline{\varphi}_j = \begin{bmatrix} \varphi_j \\ 0 \end{bmatrix}, \quad \underline{\varphi}_j = \begin{bmatrix} 0 \\ \varphi_j \end{bmatrix}, \quad j = 1, \cdots, N.$$

为了方便, 用 a 代替 a_h, 有

$$a(\boldsymbol{u},\boldsymbol{v}) = a(\overline{\boldsymbol{u}},\overline{\boldsymbol{v}}) + a(\overline{\boldsymbol{u}},\underline{\boldsymbol{v}}) + a(\underline{\boldsymbol{u}},\overline{\boldsymbol{v}}) + a(\underline{\boldsymbol{u}},\underline{\boldsymbol{v}}).$$

而

$$a(\overline{\boldsymbol{u}},\overline{\boldsymbol{v}}) = \sum_{i,j=1}^N a(\overline{\varphi}_i,\overline{\varphi}_j)\overline{u}_i\overline{v}_j = \overline{V}^T A_{11}\overline{U}, \quad A_{11}(j,i) = a(\overline{\varphi}_i,\overline{\varphi}_j),$$

$$a(\overline{\boldsymbol{u}},\underline{\boldsymbol{v}}) = \sum_{i,j=1}^N a(\overline{\varphi}_i,\underline{\varphi}_j)\overline{u}_i\underline{v}_j = \underline{V}^T A_{21}\overline{U}, \quad A_{21}(j,i) = a(\overline{\varphi}_i,\underline{\varphi}_j),$$

$$a(\underline{\boldsymbol{u}},\overline{\boldsymbol{v}}) = \overline{V}^T A_{12}\underline{U}, \quad A_{12}(j,i) = a(\underline{\varphi}_i,\overline{\varphi}_j),$$

$$a(\underline{\boldsymbol{u}},\underline{\boldsymbol{v}}) = \underline{V}^T A_{22}\underline{U}, \quad A_{22}(j,i) = a(\underline{\varphi}_i,\underline{\varphi}_j),$$

于是

$$\begin{aligned} a(\boldsymbol{u},\boldsymbol{v}) &= \overline{V}^T A_{11}\overline{U} + \underline{V}^T A_{21}\overline{U} + \overline{V}^T A_{12}\underline{U} + \underline{V}^T A_{22}\underline{U} \\ &= [\overline{V}^T, \underline{V}^T] \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \overline{U} \\ \underline{U} \end{bmatrix} =: V^T A U. \end{aligned}$$

特别地, 把 $a(\boldsymbol{u},\boldsymbol{v})$ 换成 $a_K(\boldsymbol{u},\boldsymbol{v})$, 相应地有单元上的矩阵 A_{ij}^K, 它贡献给 A_{ij}, 且按照标量情形进行装配.

对右端的线性形式, 有

$$\ell(\boldsymbol{v}) = \ell(\overline{\boldsymbol{v}}) + \ell(\underline{\boldsymbol{v}}),$$

而

$$\ell(\overline{\boldsymbol{v}}) = \sum_{i=1}^N \overline{v}_i \ell(\overline{\varphi}_i) = \overline{V}^T F_1, \quad F_1 = [\ell(\overline{\varphi}_1), \cdots, \ell(\overline{\varphi}_N)]^T,$$

$$\ell(\underline{\boldsymbol{v}}) = \sum_{i=1}^N \underline{v}_i \ell(\underline{\varphi}_i) = \underline{V}^T F_2, \quad F_2 = [\ell(\underline{\varphi}_1), \cdots, \ell(\underline{\varphi}_N)]^T,$$

于是

$$\ell(\boldsymbol{v}) = [\overline{V}^T, \underline{V}^T] \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} =: V^T F.$$

注 3.2 对 A_{ij}, 例如, $A_{21} = a(\overline{\varphi}_i, \underline{\varphi}_j)$, 双线性形式中只会留下 $(\underline{v}, \overline{u})$ 这样的配对项. 这对应标量情形的计算. 其他项以及右端类似.

3.2.2 sparse 装配指标

先考虑三角形单元. 刚度矩阵

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

的每个块都可直接按照标量情形的方法进行装配, 即用 `sparse(ii,jj,ss,N,N)` 生成, 其中的 `ii,jj` 是通用的, 由 CODE. 12 给出. 而 `ss` 如下获得: 将单元刚度矩阵行拉直, 逐个单元拼成一个矩阵 (每行对应一个单元的拉直向量), 然后再拉直为一个列向量. 若把这些块给出的拉直向量分别记为 `ss11, ss12, ss21, ss22`, 则可如下装配

```
A11 = sparse(ii,jj,ss11,N,N); A12 = sparse(ii,jj,ss12,N,N);
A21 = sparse(ii,jj,ss21,N,N); A22 = sparse(ii,jj,ss22,N,N);
A = [A11,A12; A21,A22];
```

为了避免对稀疏矩阵进行运算, 上面分块装配可如下改进

CODE 16. 向量方程的分块 sparse 装配指标 (三角剖分)

```
ii11 = ii;   jj11 = jj;  ii12 = ii;   jj12 = jj+N;
ii21 = ii+N; jj21 = jj;  ii22 = ii+N; jj22 = jj+N;
ii = [ii11; ii12; ii21; ii22];
jj = [jj11; jj12; jj21; jj22];
ss = [ss11; ss12; ss21; ss22];
A = sparse(ii,jj,ss,2*N,2*N);
```

对有些问题, 单元刚度矩阵

$$A_K = \begin{bmatrix} A_{11}^K & A_{12}^K \\ A_{21}^K & A_{22}^K \end{bmatrix}$$

并不是分块各自生成, 而是直接生成 (例如 VEM), 此时可找到每个块执行上面的装配, 即将 A_{ij}^K 行拉直拼接. 为了方便, 我们将 A_K 看成整体进行装配. 回忆一下装配指标, 对单元刚度矩阵

$[K^e] = [k_{ij}]_{3\times 3}$, 装配指标首先给出 k_{11} 所有单元的 (i,j,s), 记为 $\mathbf{i}_{11}, \mathbf{j}_{11}, \mathbf{s}_{11}$, 接着按行给出其他位置的稀疏指标, 它们拼接如下

$$\begin{bmatrix} \mathbf{i}_{11} & \mathbf{j}_{11} & \mathbf{s}_{11} \\ \mathbf{i}_{12} & \mathbf{j}_{12} & \mathbf{s}_{12} \\ \vdots & \vdots & \vdots \\ \mathbf{i}_{33} & \mathbf{j}_{33} & \mathbf{s}_{33} \end{bmatrix}.$$

类似地, 对这里的 A_K, 也要按行逐个给出每个元素所有单元的稀疏指标, 然后拼接起来. 此时的

$$A_K = [k_{ij}]_{(2N_{dof})\times(2N_{dof})}, \quad N_{dof} = 3,$$

可如下给出装配指标

CODE 17. 向量方程的 sparse 装配指标 (三角剖分)

```
NT = size(elem,1); N = size(node,1);
Ndof = 3; Ndof2 = 2*Ndof; nnz = NT*Ndof2^2;
ii = zeros(nnz,1); jj = zeros(nnz,1);
id = 0;
for i = 1:Ndof2
    for j = 1:Ndof2
        iN = i>Ndof; jN = j>Ndof;
        i1 = i-iN*Ndof; j1 = j-jN*Ndof;
        ii(id+1:id+NT) = elem(:,i1) + iN*N;   % zi
        jj(id+1:id+NT) = elem(:,j1) + jN*N;   % zj
        id = id + NT;
    end
end
```

类似 CODE. 18, 对多角形剖分, 我们可如下给出装配指标

CODE 18. 弹性方程 VEM 的 sparse 装配指标

```
% --------- Assemble the stiff matrix and load vector ---------
elemLen = cellfun('length',elem); vertNum = unique(elemLen);
nnz = sum(4*elemLen.^2);
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);

id = 0; ff = zeros(2*N,1);
for Nv = vertNum(:)' % only valid for row vector

    % assemble the matrix
    idNv = find(elemLen == Nv); % find polygons with Nv vertices
    NTv = length(idNv); % number of elements with Nv vertices

    elemNv = cell2mat(elem(idNv)); % elem
    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
    s = 1; Ndof = Nv;
    for i = 1:2*Ndof
        for j = 1:2*Ndof
            iN = i>Ndof; jN = j>Ndof;
            i1 = i-iN*Ndof; j1 = j-jN*Ndof;
```

```
            ii(id+1:id+NTv) = elemNv(:,i1) + iN*N; % zi
            jj(id+1:id+NTv) = elemNv(:,j1) + jN*N; % zj
            ss(id+1:id+NTv) = K(:,s);
            id = id + NTv; s = s+1;
        end
    end

    % assemble the vector
    ff = ff +  accumarray([elemNv(:);elemNv(:)+N],F(:),[2*N 1]);
end
kk = sparse(ii,jj,ss,2*N,2*N);
```

3.3 变分形式 I: 位移型

3.3.1 过渡矩阵

设分量空间 $V_k(K)$ 的基函数为 $\phi_1,\cdots,\phi_N$, 则向量空间 $\boldsymbol{V}_k(K)$ 的基函数为

$$\overline{\phi}_1,\cdots,\overline{\phi}_N,\underline{\phi}_1,\cdots,\underline{\phi}_N,$$

式中,

$$\overline{\phi}_i=\begin{bmatrix}\phi_i\\0\end{bmatrix},\qquad \underline{\phi}_i=\begin{bmatrix}0\\\phi_i\end{bmatrix},\qquad i=1,\cdots,N.$$

对任意的 $\boldsymbol{u}=(u_1,u_2)=:(\overline{u},\underline{u})\in\boldsymbol{V}_k(K)$, 有

$$\boldsymbol{u}=\sum_{i=1}^{N_k}\overline{u}_i\overline{\phi}_i+\sum_{i=1}^{N_k}\underline{u}_i\underline{\phi}_i,\qquad \overline{u}_i=\chi_i(\overline{u}),\qquad \underline{u}_i=\chi_i(\underline{u}),$$

于是

$$\mathrm{dof}_i(\boldsymbol{u})=\chi_i(u_1),\quad \mathrm{dof}_{i+N}(\boldsymbol{u})=\chi_i(u_2),\quad 1\le i\le N.$$

类似地, 设

$$\overline{m}_\alpha=\begin{bmatrix}m_\alpha\\0\end{bmatrix},\qquad \underline{m}_\alpha=\begin{bmatrix}0\\m_\alpha\end{bmatrix},\qquad \alpha=1,2,3,$$

并把它们排列为

$$\boldsymbol{m}^T=[\overline{m}_1,\overline{m}_2,\overline{m}_3,\underline{m}_1,\underline{m}_2,\underline{m}_3]=:[\overline{m}^T,\underline{m}^T],$$
$$\boldsymbol{\phi}^T=[\overline{\phi}_1,\cdots,\overline{\phi}_{N_k},\underline{\phi}_1,\cdots,\underline{\phi}_{N_k}]=:[\overline{\phi}^T,\underline{\phi}^T].$$

设过渡矩阵为 $\boldsymbol{D}$, 即

$$\boldsymbol{m}^T=\boldsymbol{\phi}^T\boldsymbol{D},$$

由

$$\boldsymbol{m}^T=[\overline{m}^T,\underline{m}^T]=\begin{bmatrix}m^T&\mathbf{0}^T\\\mathbf{0}^T&m^T\end{bmatrix}=\begin{bmatrix}\phi^TD&\mathbf{0}^T\\\mathbf{0}^T&\phi^TD\end{bmatrix}$$
$$=\begin{bmatrix}\phi^T&\mathbf{0}^T\\\mathbf{0}^T&\phi^T\end{bmatrix}\begin{bmatrix}D&\\&D\end{bmatrix}=\boldsymbol{\phi}^T\boldsymbol{D},$$

知

$$\boldsymbol{D} = \begin{bmatrix} D & \\ & D \end{bmatrix}.$$

3.3.2 椭圆投影

椭圆投影 Π^∇ 类似 Poisson 方程定义, 只不过此时是向量型的, 即

$$\Pi^\nabla: \quad \boldsymbol{V}_k(K) \to (\mathbb{P}_k(K))^2, \quad \boldsymbol{v} \mapsto \Pi^\nabla \boldsymbol{v},$$

满足

$$\begin{cases} a_\nabla^K(\Pi^\nabla \boldsymbol{v}, \boldsymbol{p}) = a_\nabla^K(\boldsymbol{v}, \boldsymbol{p}), \quad \boldsymbol{p} \in (\mathbb{P}_k(K))^2, \\ \displaystyle\int_{\partial K} \Pi^\nabla \boldsymbol{v} \mathrm{d}s = \int_{\partial K} \boldsymbol{v} \mathrm{d}s \end{cases}$$

或

$$\begin{cases} \displaystyle\int_K \nabla \Pi^\nabla \boldsymbol{v} \cdot \nabla \boldsymbol{p} \mathrm{d}x = \int_K \nabla \boldsymbol{v} \cdot \nabla \boldsymbol{p} \mathrm{d}x, \quad \boldsymbol{p} \in (\mathbb{P}_k(K))^2, \\ \displaystyle\int_{\partial K} \Pi^\nabla \boldsymbol{v} \mathrm{d}s = \int_{\partial K} \boldsymbol{v} \mathrm{d}s. \end{cases} \tag{3.24}$$

定义的向量形式为

$$\begin{cases} a_\nabla^K(\boldsymbol{m}, \Pi^\nabla \boldsymbol{\phi}^T) = a_\nabla^K(\boldsymbol{m}, \boldsymbol{\phi}^T), \\ P_0(\Pi^\nabla \boldsymbol{\phi}^T) = P_0(\boldsymbol{\phi}^T) \end{cases}$$

或

$$\begin{cases} \displaystyle\int_K \nabla \boldsymbol{m} \cdot \nabla \Pi^\nabla \boldsymbol{\phi}^T \mathrm{d}x = \int_K \nabla \boldsymbol{m} \cdot \nabla \boldsymbol{\phi}^T \mathrm{d}x, \\ P_0(\Pi^\nabla \boldsymbol{\phi}^T) = P_0(\boldsymbol{\phi}^T), \end{cases}$$

设 Π^∇ 在基 $\boldsymbol{\phi}^T$ 下的矩阵为 $\boldsymbol{\Pi}^\nabla$, 即

$$\Pi^\nabla \boldsymbol{\phi}^T = \boldsymbol{\phi}^T \boldsymbol{\Pi}^\nabla, \quad \boldsymbol{\Pi}^\nabla = \left(\mathrm{dof}_i(\Pi^\nabla \boldsymbol{\phi}_j)\right).$$

根据前面自由度的说明, 有

$$\boldsymbol{\Pi}^\nabla = \begin{bmatrix} \left(\chi_i(\Pi^\nabla \phi_j)\right) & \\ & \left(\chi_i(\Pi^\nabla \phi_j)\right) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Pi}_\nabla & \\ & \boldsymbol{\Pi}_\nabla \end{bmatrix}.$$

这里为了区别, 设 $\Pi^\nabla = (\Pi_\nabla, \Pi_\nabla)^T$, 即把分量的投影写为下标, 且记 Π_∇ 在基 ϕ^T 下的矩阵为 $\boldsymbol{\Pi}_\nabla$, 即

$$\Pi_\nabla \phi^T = \phi^T \boldsymbol{\Pi}_\nabla, \quad \boldsymbol{\Pi}_\nabla = \left(\chi_i(\Pi^\nabla \phi_j)\right). \tag{3.25}$$

也可如下获得. 由 (3.25),

$$\begin{aligned} \Pi^\nabla \boldsymbol{\phi}^T &= [\Pi^\nabla \overline{\phi}^T, \Pi^\nabla \underline{\phi}^T] = \begin{bmatrix} \Pi_\nabla \phi^T & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \Pi_\nabla \phi^T \end{bmatrix} \\ &= \begin{bmatrix} \phi^T \boldsymbol{\Pi}_\nabla & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \phi^T \boldsymbol{\Pi}_\nabla \end{bmatrix} = \begin{bmatrix} \phi^T & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \phi^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\Pi}_\nabla & \\ & \boldsymbol{\Pi}_\nabla \end{bmatrix} \\ &= [\overline{\phi}^T, \underline{\phi}^T] \begin{bmatrix} \boldsymbol{\Pi}_\nabla & \\ & \boldsymbol{\Pi}_\nabla \end{bmatrix} = \boldsymbol{\phi}^T \begin{bmatrix} \boldsymbol{\Pi}_\nabla & \\ & \boldsymbol{\Pi}_\nabla \end{bmatrix} =: \boldsymbol{\phi}^T \boldsymbol{\Pi}^\nabla, \end{aligned}$$

此即 Π^∇ 在基 $\boldsymbol{\phi}^T$ 下的矩阵为

$$\boldsymbol{\Pi}^\nabla = \begin{bmatrix} \boldsymbol{\Pi}_\nabla & \\ & \boldsymbol{\Pi}_\nabla \end{bmatrix}.$$

再设投影向量 $\Pi_\nabla \phi^T$ 在多项式基下的矩阵为 $\boldsymbol{\Pi}_{a*}$, 即

$$\Pi_\nabla \phi^T = m^T \boldsymbol{\Pi}_{a*}, \tag{3.26}$$

则有

$$\boldsymbol{\Pi}_\nabla = D\boldsymbol{\Pi}_{\nabla*},$$

其中, D 为过渡矩阵. 类似由 (3.25) 得

$$\Pi^\nabla \boldsymbol{\phi}^T = \boldsymbol{m}^T \boldsymbol{\Pi}^{\nabla*}, \quad \boldsymbol{\Pi}^{\nabla*} = \begin{bmatrix} \boldsymbol{\Pi}_{\nabla*} & \\ & \boldsymbol{\Pi}_{\nabla*} \end{bmatrix}.$$

显然,

$$\boldsymbol{\Pi}^\nabla = \boldsymbol{D}\boldsymbol{\Pi}^{\nabla*}, \quad \boldsymbol{D} = \begin{bmatrix} D & \\ & D \end{bmatrix}.$$

代入前面的矩阵表示, 有

$$\begin{cases} \boldsymbol{G}\boldsymbol{\Pi}^{\nabla*} = \boldsymbol{B}, \\ P_0(\boldsymbol{m}^T)\boldsymbol{\Pi}^{\nabla*} = P_0(\boldsymbol{\phi}^T), \end{cases} \tag{3.27}$$

其中,

$$\boldsymbol{G} = a_\nabla^K(\boldsymbol{m}, \boldsymbol{m}^T) = \int_K \nabla \boldsymbol{m} \cdot \nabla \boldsymbol{m}^T \mathrm{d}x,$$

$$\boldsymbol{B} = a_\nabla^K(\boldsymbol{m}, \boldsymbol{\phi}^T) = \int_K \nabla \boldsymbol{m} \cdot \nabla \boldsymbol{\phi}^T \mathrm{d}x.$$

直接计算有

$$\begin{aligned} \boldsymbol{G} = \int_K \nabla \boldsymbol{m} \cdot \nabla \boldsymbol{m}^T \mathrm{d}x &= \begin{bmatrix} (\nabla \overline{m}, \nabla \overline{m}^T)_K & (\nabla \overline{m}, \nabla \underline{m}^T)_K \\ (\nabla \underline{m}, \nabla \overline{m}^T)_K & (\nabla \underline{m}, \nabla \underline{m}^T)_K \end{bmatrix} \\ &= \begin{bmatrix} (\nabla m, \nabla m^T)_K & \\ & (\nabla m, \nabla m^T)_K \end{bmatrix} = \begin{bmatrix} G & \\ & G \end{bmatrix}, \end{aligned}$$

类似地,

$$\boldsymbol{B} = \begin{bmatrix} B & \\ & B \end{bmatrix}.$$

根据 Poisson 方程, G 不可逆, 它的第一行全为零, 为此 $\boldsymbol{G}$ 的分块的第一行都为零. 注意到 (3.27) 中的

$$P_0(\boldsymbol{m}^T) = [P_0(\overline{m}^T), P_0(\underline{m}^T)] = \begin{bmatrix} P_0(m^T) & \boldsymbol{0}^T \\ \boldsymbol{0}^T & P_0(m^T) \end{bmatrix},$$

$$P_0(\boldsymbol{\phi}^T) = [P_0(\overline{\phi}^T), P_0(\underline{\phi}^T)] = \begin{bmatrix} P_0(\phi^T) & \boldsymbol{0}^T \\ \boldsymbol{0}^T & P_0(\phi^T) \end{bmatrix},$$

它们是两行的方程, 用它们分别替换分块的首行, 则有

$$\tilde{\boldsymbol{G}}\boldsymbol{\Pi}^{\nabla*} = \tilde{\boldsymbol{B}},$$

其中,

$$\tilde{\boldsymbol{G}}=\begin{bmatrix}\tilde{G} & \\ & \tilde{G}\end{bmatrix},\quad \tilde{\boldsymbol{B}}=\begin{bmatrix}\tilde{B} & \\ & \tilde{B}\end{bmatrix}.$$

由分量矩阵的关系, 我们有

引理 3.1

$$\boldsymbol{\Pi}^{\nabla *}=\tilde{\boldsymbol{G}}^{-1}\tilde{\boldsymbol{B}},\qquad \boldsymbol{\Pi}^{\nabla}=\boldsymbol{D}\boldsymbol{\Pi}^{\nabla *}.$$

$$\boldsymbol{G}=\boldsymbol{B}\boldsymbol{D},\qquad \tilde{\boldsymbol{G}}=\tilde{\boldsymbol{B}}\boldsymbol{D}.$$

注 3.3 正因为向量情形的矩阵只是分量矩阵的对角分块, 投影矩阵自然也是, 从而不必再计算向量情形的各种矩阵.

3.3.3 L^2 投影

对 Poisson 方程, L^2 投影 Π_k^0 和 Π_{k-1}^0 都不能定义, 因为它们用到 K 上的 $\geq k-1$ 的矩量. 但对这里的

$$a_{\mathrm{div}}^K(\boldsymbol{u},\boldsymbol{v})=\int_K(\mathrm{div}\ \boldsymbol{u})(\mathrm{div}\ \boldsymbol{v})\mathrm{d}x,$$

可用 $\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{u})$ 近似 $\mathrm{div}\ \boldsymbol{u}$, 即可定义

$$\Pi_{k-1}^0\mathrm{div}\ :\ \boldsymbol{v}\mapsto\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{v}),$$

$$\int_K\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{v})p\mathrm{d}x=\int_K(\mathrm{div}\ \boldsymbol{v})p\mathrm{d}x,\quad p\in\mathbb{P}_{k-1}(K),$$

这是因为此时的右端可计算. 事实上,

$$\int_K(\mathrm{div}\ \boldsymbol{v})p\mathrm{d}x=-\int_K\boldsymbol{v}\cdot\nabla p\mathrm{d}x+\int_{\partial K}(\boldsymbol{v}\cdot\boldsymbol{n})p\mathrm{d}s,\quad p\in\mathbb{P}_{k-1}(K),$$

右端第一项的 $\nabla p\in(\mathbb{P}_{k-2}(K))^2$, 从而能表为自由度的组合, 而边界项属于有限元问题.

定义的向量形式为

$$\int_K m^0\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{\phi}^T)\mathrm{d}x=\int_K m^0(\mathrm{div}\ \boldsymbol{\phi}^T)\mathrm{d}x,$$

其中, m^0 是阶不大于 $k-1$ 次的尺度单项式的列向量. 设 $\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{\phi}^T)$ 在多项式基 $(m^0)^T$ 下的矩阵为 Π_{0*}, 即

$$\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{\phi}^T)=(m^0)^T\Pi_{0*},$$

则投影的矩阵形式为

$$H_0\Pi_{0*}=C_0,$$

式中,

$$H_0=\int_K m^0(m^0)^T\mathrm{d}x,\quad C_0=\int_K m^0(\mathrm{div}\ \boldsymbol{\phi}^T)\mathrm{d}x.$$

显然 H_0 是 H 的前若干行若干列组成.

对 $k=1$, 有 $m^0=m_1=1$, 且

$$C_0=\int_K m^0(\mathrm{div}\ \boldsymbol{\phi}^T)\mathrm{d}x=\int_K\mathrm{div}\ \boldsymbol{\phi}^T\mathrm{d}x=\int_{\partial K}\boldsymbol{\phi}^T\cdot\boldsymbol{n}\mathrm{d}s,\tag{3.28}$$

这里

$$\int_{\partial K} \boldsymbol{\phi}^T \cdot \boldsymbol{n} \mathrm{d}s = \int_{\partial K} [\overline{\phi}^T \cdot \boldsymbol{n}, \underline{\phi}^T \cdot \boldsymbol{n}] \mathrm{d}s = \int_{\partial K} [\phi^T \cdot n_x, \phi^T \cdot n_y] \mathrm{d}s.$$

由 Poisson 方程那里的 (2.8),

$$\int_{\partial K} \phi_i \cdot \boldsymbol{n} \mathrm{d}s = \frac{1}{2} \left(|e_{i-1}| \boldsymbol{n}_{e_{i-1}} + |e_i| \boldsymbol{n}_{e_i} \right) = \frac{1}{2} |\hat{e}_i| \boldsymbol{n}_{\hat{e}_i},$$

此即

$$\int_{\partial K} [\phi_i \cdot n_x, \phi_i \cdot n_y]^T \mathrm{d}s = \frac{1}{2} |\hat{e}_i| \boldsymbol{n}_{\hat{e}_i},$$

由此可获得 C_0. 显然此时有 $H_0 = |K|$, 于是

$$\Pi_{0*} = H_0^{-1} C_0 = |K|^{-1} C_0.$$

前面需要的矩阵可如下生成

```
function [D,Bs,G,Gs,H0,C0] = VEM_MAT_elasticity_displacement(aux)

% aux = auxgeometry(node,elem);

node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid;
diameter = aux.diameter; area = aux.area;
NT = size(elem,1);

D = cell(NT,1);
% B = cell(NT,1);  % it is not used in the computation
Bs = cell(NT,1);
G = cell(NT,1);  Gs = cell(NT,1);
H0 = cell(NT,1); C0 = cell(NT,1);
for iel = 1:NT
    index = elem{iel};     Nv = length(index);
    xK = elemCentroid(iel,1); yK = elemCentroid(iel,2); hK = diameter(iel);

    m2 = @(x,y) (x-xK)./hK;   m3 = @(x,y) (y-yK)./hK;
    m = @(x,y) [m2(x,y), m3(x,y)];  % m2,m3
    x = node(index,1); y = node(index,2);

    % D
    D1 = zeros(Nv,3); D1(:,1) = 1;
    D1(:,2:3) = m(x,y);   D{iel} = D1;

    % B, Bs, G, Gs
    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
    Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of edge vector
    B1 = Gradm*normVec; B1s = B1; B1s(1,:) = 1/Nv;
    % B{iel} = B1;
    Bs{iel} = B1s;
    G{iel} = B1*D1;     Gs{iel} = B1s*D1;
```

```

    % H0,C0
    H0{iel} = area(iel);
    C0{iel} = reshape(normVec',1,[]);
end
```

3.3.4 单元刚度矩阵和载荷向量的计算

类似 Poisson 方程, 有

$$\boldsymbol{A}_K^1 = a_\nabla^K(\Pi^\nabla \boldsymbol{\phi}, \Pi^\nabla \boldsymbol{\phi}^T) = (\boldsymbol{\Pi}^{\nabla *})^T a_\nabla^K(\boldsymbol{m}, \boldsymbol{m}^T)\boldsymbol{\Pi}^{\nabla *} = (\boldsymbol{\Pi}^{\nabla *})^T \boldsymbol{G}\boldsymbol{\Pi}^{\nabla *},$$

$$\boldsymbol{A}_K^2 = (\boldsymbol{I} - \boldsymbol{\Pi}^\nabla)^T(\boldsymbol{I} - \boldsymbol{\Pi}^\nabla),$$

$$\boldsymbol{B}_K = \int_K \Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{\phi})\Pi_{k-1}^0(\mathrm{div}\ \boldsymbol{\phi}^T)\mathrm{d}x = (\Pi_{0*})^T H_0 \Pi_{0*}.$$

由矩阵的分块性质知, $\boldsymbol{A}_K^1$ 和 $\boldsymbol{A}_K^2$ 也是分块对角的, 即

$$\boldsymbol{A}_K^i = \begin{bmatrix} A_K^i & \\ & A_K^i \end{bmatrix}, \quad i = 1, 2,$$

这里,

$$A_K^1 = (\boldsymbol{\Pi}_{\nabla *})^T G \boldsymbol{\Pi}_{\nabla *}, \qquad A_K^2 = (\boldsymbol{I} - \boldsymbol{\Pi}_\nabla)^T(\boldsymbol{I} - \boldsymbol{\Pi}_\nabla),$$

且 $\boldsymbol{\Pi}_\nabla$ 就是 Poisson 方程情形的 $\boldsymbol{\Pi}_k^\nabla$, 而 $\boldsymbol{\Pi}_{\nabla *}$ 就是那里的 $\boldsymbol{\Pi}_{k*}^\nabla$.

再考虑右端的计算. 右端的向量形式为

$$\boldsymbol{F}_K = \ell(\boldsymbol{\phi}) = \int_K \Pi_{k-2}^0 \boldsymbol{f} \cdot \boldsymbol{\phi}\mathrm{d}x = \int_K \begin{bmatrix} \Pi_{k-2}^0 f_1 \cdot \phi \\ \Pi_{k-2}^0 f_2 \cdot \phi \end{bmatrix} \mathrm{d}x,$$

为此只要考虑

$$F_K(\phi) = \int_K \Pi_{k-2}^0 f\phi\mathrm{d}x, \qquad f = f_1, f_2.$$

当 $k = 1$ 时, 规定 $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (2.9) 定义, 即单元顶点值的平均. 注意到

$$F_K(\phi) = \int_K \Pi_0^0 f\phi\mathrm{d}x = \int_K \Pi_0^0 f \Pi_k^0 \phi\mathrm{d}x,$$

我们有

$$F_K = \int_K \Pi_0^0 f \Pi_0^0 \phi\mathrm{d}x = f(x_K, y_K)\,|K|\,\frac{1}{N_v}[1, \cdots, 1]_{N_k}^T,$$

这里为了方便, $\Pi_0^0 f$ 用中点值近似.

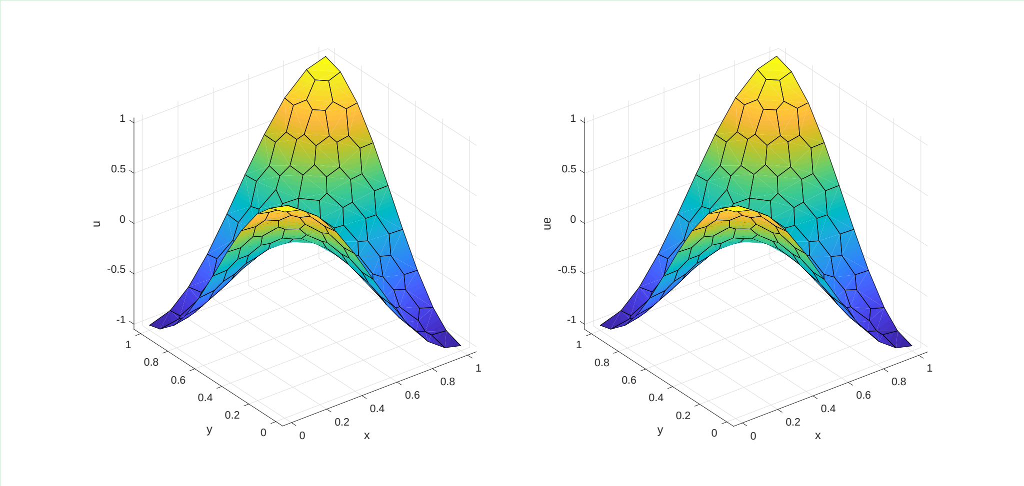
如下获得刚度矩阵和载荷向量

```
% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    % Projection
    Pis = Gs{iel}\Bs{iel};    Pi  = D{iel}*Pis;    I = eye(size(Pi));
    Pi0s = H0{iel}\C0{iel};
    % Stiffness matrix
```

```
    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi); AK = blkdiag(AK,AK);
    BK = PiOs'*HO{iel}*PiOs;
    ABelem{iel} = reshape(pde.mu*AK+(pde.mu+pde.lambda)*BK,1,[]); %straighten
    % Load vector
    Nv = length(elem{iel});
    fK = pde.f(elemCentroid(iel,:))*area(iel)/Nv; fK = repmat(fK,Nv,1);
    belem{iel} = fK(:)'; % straighten
end
```

3.3.5 程序整理

数值解和精确解的图像如下



主程序为

CODE 19. main_elasticityVEM_displacement.m

```
clc;clear;close all
tic;
% ------------- Mesh --------------------
% generated by meshfun.m in tool
load meshdata.mat; % rectangle
%load meshdataCircle.mat;  % circle
figure, showmesh(node,elem);
findnode(node); % findelem(node,elem);

% ---------------- PDE data -----------------------
pde = elasticitydata();
bdStruct = setboundary(node,elem);

% ------- Solve the problem -----
u = elasticityVEM_displacement(node,elem,pde,bdStruct);

% --------- error analysis -------
u = reshape(u,[],2);
uexact = pde.uexact;  ue = uexact(node);
id = 1;
```

```
figure,
subplot(1,2,1), showsolution(node,elem,u(:,id));
zlabel('u');
subplot(1,2,2), showsolution(node,elem,ue(:,id));
zlabel('ue');
Eabs = u-ue;  % Absolute errors
figure,showsolution(node,elem,Eabs(:,id)); zlim('auto');
format shorte
Err = norm(Eabs)./norm(ue)
toc
```

函数文件为

CODE 20. elasticityVEM_displacement.m

```
function u = elasticityVEM_displacement(node,elem,pde,bdStruct)

aux = auxgeometry(node,elem);
node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid; area = aux.area;

N = size(node,1);  NT = size(elem,1);
% ----------------- D,Bs,G,Gs,H0,C0 ---------------------------
[D,Bs,G,Gs,H0,C0] = VEM_MAT_elasticity_displacement(aux);

% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    % Projection
    Pis = Gs{iel}\Bs{iel};   Pi  = D{iel}*Pis;   I = eye(size(Pi));
    Pi0s = H0{iel}\C0{iel};
    % Stiffness matrix
    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi); AK = pde.mu*blkdiag(AK,AK);
    BK = (pde.mu+pde.lambda)*Pi0s'*H0{iel}*Pi0s;
    ABelem{iel} = reshape(AK+BK,1,[]); % straighten
    % Load vector
    Nv = length(elem{iel});
    fK = pde.f(elemCentroid(iel,:))*area(iel)/Nv; fK = repmat(fK,Nv,1);
    belem{iel} = fK(:)'; % straighten
end

% --------- Assemble the stiffness matrix and load vector ---------
elemLen = cellfun('length',elem); vertNum = unique(elemLen);
nnz = sum(4*elemLen.^2);
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);

id = 0; ff = zeros(2*N,1);
for Nv = vertNum(:)' % only valid for row vector

    % assemble the matrix
    idNv = find(elemLen == Nv); % find polygons with Nv vertices
```

```
    NTv = length(idNv); % number of elements with Nv vertices

    elemNv = cell2mat(elem(idNv)); % elem
    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
    s = 1; Ndof = Nv;
    for i = 1:2*Ndof
        for j = 1:2*Ndof
            iN = i>Ndof; jN = j>Ndof;
            i1 = i-iN*Ndof; j1 = j-jN*Ndof;
            ii(id+1:id+NTv) = elemNv(:,i1) + iN*N; % zi
            jj(id+1:id+NTv) = elemNv(:,j1) + jN*N; % zj
            ss(id+1:id+NTv) = K(:,s);
            id = id + NTv; s = s+1;
        end
    end

    % assemble the vector
    ff = ff +  accumarray([elemNv(:);elemNv(:)+N],F(:),[2*N 1]);
end
kk = sparse(ii,jj,ss,2*N,2*N);

% ------------ Dirichlet boundary condition ----------------
g_D = pde.g_D;  eD = bdStruct.eD;
isBdNode = false(N,1); isBdNode(eD) = true;
bdNode = find(isBdNode); freeNode = find(¬isBdNode);
pD = node(bdNode,:);
bdDof = [bdNode; bdNode+N]; freeDof = [freeNode;freeNode+N];
u = zeros(2*N,1); uD = g_D(pD); u(bdDof) = uD(:);
ff = ff - kk*u;

% ------------------ Solver -------------------
u(freeDof) = kk(freeDof,freeDof)\ff(freeDof);
```

3.4 变分形式 II: 张量型

只要考虑投影投影, 其他部分与变分形式 I 相同.

3.4.1 椭圆投影的定义

设 $\boldsymbol{u}_h \in \boldsymbol{V}_k(K)$, $\boldsymbol{p} \in (\mathbb{P}_k(K))^2$, 分部积分有

$$\begin{aligned}
&\int_K \varepsilon_{ij}(\boldsymbol{u}_h)\varepsilon_{ij}(\boldsymbol{p})\mathrm{d}x \\
&= \frac{1}{2}\int_K (\partial_i u_j + \partial_j u_i)\varepsilon_{ij}(\boldsymbol{p})\mathrm{d}x \\
&= \frac{1}{2}\int_{\partial K} (u_j\varepsilon_{ij}(\boldsymbol{p})n_i + u_i\varepsilon_{ij}(\boldsymbol{p})n_j)\mathrm{d}x - \frac{1}{2}\int_K (u_j\partial_i\varepsilon_{ij}(\boldsymbol{p}) + u_i\partial_j\varepsilon_{ij}(\boldsymbol{p}))\mathrm{d}x \\
&= \frac{1}{2}\int_{\partial K} (u_j\varepsilon_{ij}(\boldsymbol{p})n_i + u_i\varepsilon_{ij}(\boldsymbol{p})n_j)\mathrm{d}x - \frac{1}{2}\int_K (u_j\partial_i\varepsilon_{ij}(\boldsymbol{p}) + u_i\partial_j\varepsilon_{ij}(\boldsymbol{p}))\mathrm{d}x.
\end{aligned}$$

由对称性,

$$
\begin{aligned}
&\int_K \varepsilon_{ij}(\boldsymbol{u}_h)\varepsilon_{ij}(\boldsymbol{p})\mathrm{d}x\\
&=\frac{1}{2}\int_{\partial K}(u_j\varepsilon_{ji}(\boldsymbol{p})n_i+u_i\varepsilon_{ij}(\boldsymbol{p})n_j)\mathrm{d}x-\frac{1}{2}\int_K(u_j\partial_i\varepsilon_{ji}(\boldsymbol{p})+u_i\partial_j\varepsilon_{ij}(\boldsymbol{p}))\mathrm{d}x\\
&=\int_{\partial K}u_i\varepsilon_{ij}(\boldsymbol{p})n_j\mathrm{d}x-\int_K u_i\partial_j\varepsilon_{ij}(\boldsymbol{p})\mathrm{d}x,
\end{aligned}
$$

此即

$$
a_\mu^K(\boldsymbol{u}_h,\boldsymbol{p})=\int_K\boldsymbol{\varepsilon}(\boldsymbol{u}_h):\boldsymbol{\varepsilon}(\boldsymbol{p})\mathrm{d}x=-\int_K\mathrm{div}(\boldsymbol{\varepsilon}(\boldsymbol{p}))\cdot\boldsymbol{u}_h\mathrm{d}x+\int_{\partial K}(\boldsymbol{\varepsilon}(\boldsymbol{p})\cdot\boldsymbol{n})\cdot\boldsymbol{u}_h\mathrm{d}s, \tag{3.29}
$$

这里 $\cdot\boldsymbol{u}_h$ 中的 $\cdot$ 表示向量内积. 注意到 $\mathrm{div}(\boldsymbol{\varepsilon}(\boldsymbol{p}))\in(\mathbb{P}_{k-2}(K))^2$, 右端第一项可表为 $\boldsymbol{u}_h$ 的自由度, 而边界项属于有限元问题, 故上面的双线性形式可计算 ($\boldsymbol{v}=\boldsymbol{p}\in(\mathbb{P}_k(K))^2$).

为此, 定义椭圆投影

$$
\Pi^a:\ \boldsymbol{V}_k(K)\to(\mathbb{P}_k(K))^2,\quad \boldsymbol{v}\mapsto\Pi^a\boldsymbol{v},
$$

满足

$$
a_\mu^K(\Pi^a\boldsymbol{v},\boldsymbol{p})=a_\mu^K(\boldsymbol{v},\boldsymbol{p}),\quad \boldsymbol{p}\in(\mathbb{P}_k(K))^2. \tag{3.30}
$$

显然这样定义的投影不唯一. 事实上, 若 $\Pi^a\boldsymbol{v}+\boldsymbol{p}$ 也符合定义, 则有

$$
\int_K\boldsymbol{\varepsilon}(\boldsymbol{p}):\boldsymbol{\varepsilon}(\boldsymbol{p})\mathrm{d}x=\int_K\varepsilon_{ij}(\boldsymbol{p})\varepsilon_{ij}(\boldsymbol{p})\mathrm{d}x=0,
$$

即

$$
\varepsilon_{ij}(\boldsymbol{p})=0\quad\Rightarrow\quad\partial_ip_j+\partial_jp_i=0.
$$

由

$$
\partial_ip_i=0,\quad i=1,2\quad\Rightarrow\quad p_1=c_1,\ c_2y,\quad p_2=c_3,\ c_4x,
$$

再结合 $\partial_ip_j+\partial_jp_i=0\ (i\neq j)$ 知,

$$
\boldsymbol{p}\in\mathrm{span}\left\{\begin{bmatrix}1\\0\end{bmatrix},\ \begin{bmatrix}0\\1\end{bmatrix},\ \begin{bmatrix}-y\\x\end{bmatrix}\right\}=:K_0. \tag{3.31}
$$

注意到上式中空间的自由度为 3, 为了保证唯一性, 我们可要求

$$
\int_{\partial K}(\Pi^a\boldsymbol{v})\boldsymbol{p}\mathrm{d}x=\int_{\partial K}\boldsymbol{v}\boldsymbol{p}\mathrm{d}x,\quad \boldsymbol{p}\in K_0.
$$

或更简单的

$$
\sum_{i=1}^{N_v}(\Pi^a\boldsymbol{v}(z_i),\boldsymbol{p}(z_i))=\sum_{i=1}^{N_v}(\boldsymbol{v}(z_i),\boldsymbol{p}(z_i)),\quad \boldsymbol{p}\in K_0, \tag{3.32}
$$

式中的配对表示欧式内积.

定义 (3.30) 等价于如下的向量形式

$$
a_\mu^K(\boldsymbol{m},\Pi^a\boldsymbol{\phi}^T)=a_\mu^K(\boldsymbol{m},\boldsymbol{\phi}^T)
$$

或

$$
\int_K\varepsilon_{ij}(\boldsymbol{m})\varepsilon_{ij}(\Pi^a\boldsymbol{\phi}^T)\mathrm{d}x=\int_K\varepsilon_{ij}(\boldsymbol{m})\varepsilon_{ij}(\boldsymbol{\phi}^T)\mathrm{d}x.
$$

记 K_0 三个基的列向量为 $\{\boldsymbol{p}\}$, 则限制条件为 (三行方程)

$$\sum_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \Pi^a\boldsymbol{\phi}^T(z_i))=\sum_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \boldsymbol{\phi}^T(z_i)).$$

设 Π^a 在基 $\boldsymbol{\phi}^T$ 下的矩阵为 $\boldsymbol{\Pi}^a$, 即

$$\Pi^a\boldsymbol{\phi}^T=\boldsymbol{\phi}^T\boldsymbol{\Pi}^a,$$

而 $\Pi^a\boldsymbol{\phi}^T$ 在多项式基 $\boldsymbol{m}^T$ 下的矩阵为 $\boldsymbol{\Pi}^{a*}$, 即

$$\Pi^a\boldsymbol{\phi}^T=\boldsymbol{m}^T\boldsymbol{\Pi}^{a*},$$

易知有

$$\boldsymbol{\Pi}^a=\boldsymbol{D}\boldsymbol{\Pi}^{a*},\quad \boldsymbol{D}=\begin{bmatrix} D & \\ & D\end{bmatrix}.$$

将以上表达式代入向量形式, 有

$$\begin{cases}\boldsymbol{G}\boldsymbol{\Pi}^{a*}=\boldsymbol{B},\\ \displaystyle\sum_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \boldsymbol{m}^T(z_i))\boldsymbol{\Pi}^{a*}=\sum_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \boldsymbol{\phi}^T(z_i))\end{cases}$$

式中,

$$\boldsymbol{G}=a_\mu^K(\boldsymbol{m},\boldsymbol{m}^T)=\int_K\varepsilon_{ij}(\boldsymbol{m})\varepsilon_{ij}(\boldsymbol{m}^T)\mathrm{d}x,$$

$$\boldsymbol{B}=a_\mu^K(\boldsymbol{m},\boldsymbol{\phi}^T)=\int_K\varepsilon_{ij}(\boldsymbol{m})\varepsilon_{ij}(\boldsymbol{\phi}^T)\mathrm{d}x.$$

而 $\boldsymbol{G}$ 和 $\boldsymbol{B}$ 的某些行用限制条件替换后记为 $\tilde{\boldsymbol{G}}$ 和 $\tilde{\boldsymbol{B}}$. 类似地, 我们有一致性关系

引理 3.2

$$\boldsymbol{G}=\boldsymbol{B}\boldsymbol{D},\qquad \tilde{\boldsymbol{G}}=\tilde{\boldsymbol{B}}\boldsymbol{D}.$$

3.4.2 椭圆投影的计算

当 $k=1$ 时, $\varepsilon_{ij}(m_\alpha)$ 是常数, 而且 (3.29) 右端的第一项为零, 于是

$$\boldsymbol{G}=|K|\cdot\varepsilon_{ij}(\boldsymbol{m})\varepsilon_{ij}(\boldsymbol{m}^T),\qquad \boldsymbol{B}=\int_{\partial K}(\boldsymbol{\varepsilon}(\boldsymbol{m})\cdot\boldsymbol{n})\cdot\boldsymbol{\phi}^T\mathrm{d}s.$$

先考虑 $\boldsymbol{G}$. 根据一致性关系, 它不需要直接计算, 这里简单计算一下, 以考察其可逆性. 直接计算有

$$\varepsilon_{11}(\boldsymbol{m})=[0,\frac{1}{h_K},0,0,0,0]^T,\quad \varepsilon_{22}(\boldsymbol{m})=[0,0,0,0,0,\frac{1}{h_K}]^T,$$

$$\varepsilon_{12}(\boldsymbol{m})=\varepsilon_{21}(\boldsymbol{m})=[0,0,\frac{1}{h_K},0,\frac{1}{h_K},0]^T,$$

由此获得 $\boldsymbol{G}$

```
E = zeros(6,4); % E = [E11,E12,E21,E22]
E(2,1) = 1/hK;  E([3,5],[2,3]) = 1/hK; E(6,4) = 1/hK;
G =  E(:,1)*E(:,1)'+E(:,2)*E(:,2)'+E(:,3)*E(:,3)'+E(:,4)*E(:,4)';
G = area(iel)*G;
```

计算可以看到, $\boldsymbol{G}$ 是 6×6 的矩阵, 它的第 1 行和第 4 行全为零, 而第 3 行和第 5 行相同, 为此可用限制条件的三行分别替换第 1,3,4 行, 所得记为 $\tilde{\boldsymbol{G}}$.

现计算 $\boldsymbol{B}$. 注意向量的内积, 有

$$\begin{aligned}\boldsymbol{B}_{ij} &= \int_{\partial K}(\boldsymbol{\varepsilon}(\boldsymbol{m}_i)\cdot\boldsymbol{n})\cdot\boldsymbol{\phi}_j = \int_{\partial K}\begin{bmatrix}\varepsilon_{11}(\boldsymbol{m}_i)n_x+\varepsilon_{12}(\boldsymbol{m}_i)n_y\\ \varepsilon_{21}(\boldsymbol{m}_i)n_x+\varepsilon_{22}(\boldsymbol{m}_i)n_y\end{bmatrix}\cdot\begin{bmatrix}\boldsymbol{\phi}_j(1)\\ \boldsymbol{\phi}_j(2)\end{bmatrix}\\ &= \int_{\partial K}(\varepsilon_{11}(\boldsymbol{m}_i)n_x+\varepsilon_{12}(\boldsymbol{m}_i)n_y)\boldsymbol{\phi}_j(1)+(\varepsilon_{21}(\boldsymbol{m}_i)n_x+\varepsilon_{22}(\boldsymbol{m}_i)n_y)\boldsymbol{\phi}_j(2),\end{aligned}$$

于是

$$\boldsymbol{B}(i,:) = \int_{\partial K}\left[(\varepsilon_{11}(\boldsymbol{m}_i)n_x+\varepsilon_{12}(\boldsymbol{m}_i)n_y)\phi^T,\ \ (\varepsilon_{21}(\boldsymbol{m}_i)n_x+\varepsilon_{22}(\boldsymbol{m}_i)n_y)\phi^T\right],$$

$$\boldsymbol{B} = \int_{\partial K}\left[(\varepsilon_{11}(\boldsymbol{m})n_x+\varepsilon_{12}(\boldsymbol{m})n_y)\phi^T,\ \ (\varepsilon_{21}(\boldsymbol{m})n_x+\varepsilon_{22}(\boldsymbol{m})n_y)\phi^T\right].$$

对 $k=1$, $\varepsilon_{ij}(\boldsymbol{m})$ 是常向量, 且上面已经计算出了, 故

$$\boldsymbol{B} = \left[\varepsilon_{11}(\boldsymbol{m})\int_{\partial K}\phi^T n_x+\varepsilon_{12}(\boldsymbol{m})\int_{\partial K}\phi^T n_y,\ \ \varepsilon_{21}(\boldsymbol{m})\int_{\partial K}\phi^T n_x+\varepsilon_{22}(\boldsymbol{m})\int_{\partial K}\phi^T n_y\right].$$

注意到 (3.28), 即

$$C_0 = \int_{\partial K}\phi^T\cdot\boldsymbol{n}\mathrm{d}s = \int_{\partial K}[\phi^T\cdot n_x, \phi^T\cdot n_y]\mathrm{d}s,$$

从而可获得 $\boldsymbol{B}$. 现考虑限制条件的右端 $\sum\limits_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \boldsymbol{\phi}^T(z_i))$. 直接计算有

$$(\{\boldsymbol{p}\},\ \phi^T) = \begin{bmatrix}\phi_1 & \cdots & \phi_{N_v} & 0 & \cdots & 0\\ 0 & \cdots & 0 & \phi_1 & \cdots & \phi_{N_v}\\ -y\phi_1 & \cdots & -y\phi_{N_v} & x\phi_1 & \cdots & x\phi_{N_v}\end{bmatrix}.$$

由基函数的 Kronecker 性质,

$$\sum_{i=1}^{N_v}(\{\boldsymbol{p}(z_i)\},\ \phi^T(z_i)) = \begin{bmatrix}1 & \cdots & 1 & 0 & \cdots & 0\\ 0 & \cdots & 0 & 1 & \cdots & 1\\ -y_1 & \cdots & -y_{N_v} & x_1 & \cdots & x_{N_v}\end{bmatrix},$$

它们要替换 $\boldsymbol{B}$ 的 1,3,4 行, 从而获得 $\tilde{\boldsymbol{B}}$.

3.4.3 程序整理

数值结果与位移型差不多, 这里省略. 主程序如下

CODE 21. main_elasticityVEM_tensor.m

```
clc;clear;close all
tic;
% ------------- Mesh --------------------
% generated by meshfun.m in tool
load meshdata.mat; % rectangle
%load meshdataCircle.mat;  % circle
%figure, showmesh(node,elem);
%findnode(node); % findelem(node,elem);
```

```
% ---------------- PDE data -----------------------
pde = elasticitydata();
bdStruct = setboundary(node,elem);

% ------- Solve the problem -----
u = elasticityVEM_tensor(node,elem,pde,bdStruct);

% --------- error analysis -------
u = reshape(u,[],2);
uexact = pde.uexact;  ue = uexact(node);
id = 1;
figure,
subplot(1,2,1), showsolution(node,elem,u(:,id));
zlabel('u');
subplot(1,2,2), showsolution(node,elem,ue(:,id));
zlabel('ue');
Eabs = u-ue;  % Absolute errors
figure,showsolution(node,elem,Eabs(:,id)); zlim('auto'); zlabel('Err');
format shorte
Err = norm(Eabs)./norm(ue)
toc
```

函数文件如下

CODE 22. elasticityVEM_tensor.m

```
function u = elasticityVEM_tensor(node,elem,pde,bdStruct)

aux = auxgeometry(node,elem);
node = aux.node; elem = aux.elem;
elemCentroid = aux.elemCentroid; area = aux.area;

N = size(node,1);  NT = size(elem,1);
% ----------------- D,Bs,G,Gs,H0,C0 ---------------------------
[D,Bs,G,Gs,H0,C0] = VEM_MAT_elasticity_tensor(aux);

% -------- Elementwise stiffness matrix and load vector -----------
ABelem = cell(NT,1); belem = cell(NT,1);
for iel = 1:NT
    % Projection
    Pis = Gs{iel}\Bs{iel};   Pi  = D{iel}*Pis;   I = eye(size(Pi));
    Pi0s = H0{iel}\C0{iel};
    % Stiffness matrix
    AK  = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi); AK = 2*pde.mu*AK;
    BK = pde.lambda*Pi0s'*H0{iel}*Pi0s;
    ABelem{iel} = reshape(AK+BK,1,[]); % straighten
    % Load vector
    Nv = length(elem{iel});
    fK = pde.f(elemCentroid(iel,:))*area(iel)/Nv; fK = repmat(fK,Nv,1);
    belem{iel} = fK(:)'; % straighten
```

```
end

% --------- Assemble the stiffness matrix and load vector ---------
elemLen = cellfun('length',elem); vertNum = unique(elemLen);
nnz = sum(4*elemLen.^2);
ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);

id = 0; ff = zeros(2*N,1);
for Nv = vertNum(:)' % only valid for row vector

    % assemble the matrix
    idNv = find(elemLen == Nv); % find polygons with Nv vertices
    NTv = length(idNv); % number of elements with Nv vertices

    elemNv = cell2mat(elem(idNv)); % elem
    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
    s = 1; Ndof = Nv;
    for i = 1:2*Ndof
        for j = 1:2*Ndof
            iN = i>Ndof; jN = j>Ndof;
            i1 = i-iN*Ndof; j1 = j-jN*Ndof;
            ii(id+1:id+NTv) = elemNv(:,i1) + iN*N; % zi
            jj(id+1:id+NTv) = elemNv(:,j1) + jN*N; % zj
            ss(id+1:id+NTv) = K(:,s);
            id = id + NTv; s = s+1;
        end
    end

    % assemble the vector
    ff = ff +  accumarray([elemNv(:);elemNv(:)+N],F(:),[2*N 1]);
end
kk = sparse(ii,jj,ss,2*N,2*N);

% ------------ Dirichlet boundary condition ----------------
g_D = pde.g_D;  eD = bdStruct.eD;
isBdNode = false(N,1); isBdNode(eD) = true;
bdNode = find(isBdNode); freeNode = find(¬isBdNode);
pD = node(bdNode,:);
bdDof = [bdNode; bdNode+N]; freeDof = [freeNode;freeNode+N];
u = zeros(2*N,1); uD = g_D(pD); u(bdDof) = uD(:);
ff = ff - kk*u;

% ------------------ Solver -------------------
u(freeDof) = kk(freeDof,freeDof)\ff(freeDof);
```

References

[1] L. Beirão da Veiga, F. Brezzi, L. D. Marini and A. Russo. The Hitchhiker's Guide to the Virtual Element Method. Mathematical Models and Methods in Applied Sciences, 24(8): 1541-1573, 2014.