MATLAB Programming for Virtual Element Methods

1 二维网格剖分与预处理 **4**

1.1 网格的数据结构及图示 4

1.1.1 基本数据结构 4

1.1.2 补片函数 patch 5

1.1.3 操作 cell 数组的 cellfun 函数 6

1.1.4 showmesh 函数的建立 7

1.1.5 showsolution 函数的建立 8

1.2 Generation of the polygonal meshes 11

1.3 网格的标记 11

1.3.1 节点标记 11

1.3.2 单元标记 12

1.3.3 边的标记 13

1.4 VEM 空间的辅助数据结构与几何量 17

1.4.1 elem2edge 的生成 18

1.4.2 edge2elem 的生成 20

1.4.3 neighbor 的生成 23

1.5 网格相关的几何量 23

1.6 auxstructure 与 auxgeometry 函数 24

1.7 边界设置 26

1.7.1 边界边的定向 26

1.7.2 边界的设置 27

2 PolyMesher: 二维多角形剖分的生成 **30**

2.1 Voronoi 图 30

2.1.1 Voronoi 图的定义 30

2.1.2 Voronoi 图的生成 30

2.1.3 CVTs 及其变分描述 33

2.1.4 计算 CVTs 的 Lloyd 算法 34

2.2 反射集的计算 35

2.2.1 符号距离函数 35

2.2.2 初始种子的生成 35

2.2.3 光滑边界凸区域反射集的计算 37
2.2.4 分段光滑边界凸区域反射集的计算 38
2.2.5 非凸区域反射集的计算 40
2.2.6 反射集计算程序 40
2.3 Lloyd 迭代 41
2.4 获取网格剖分的基本数据集 41
2.5 获取网格剖分 43

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

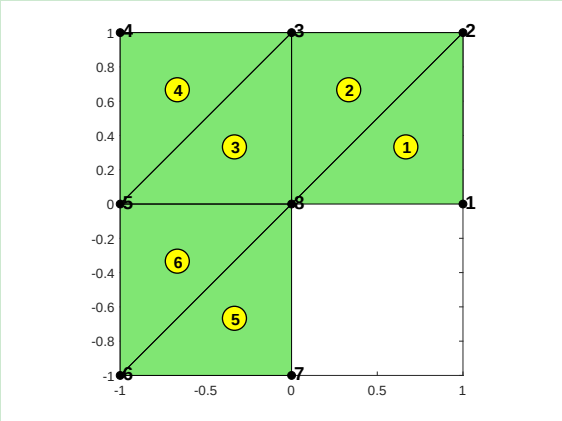


图 1. L 形区域的剖分

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.

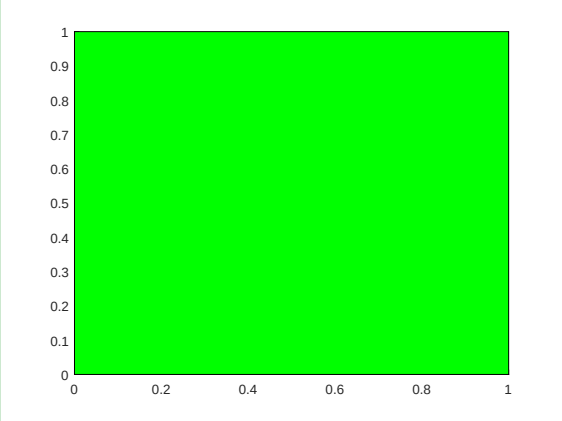
6x3 double

	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

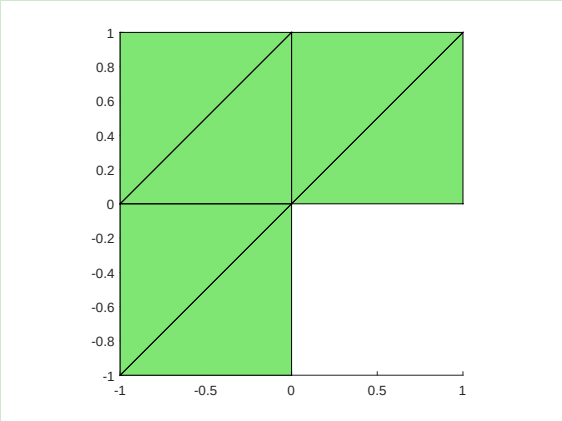
```
node = [0 0; 1 0; 1 1; 0 1];
elem = [1 2 3 4];
patch('Faces',node,'Vertices',elem,'FaceColor','g')
```

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;
```

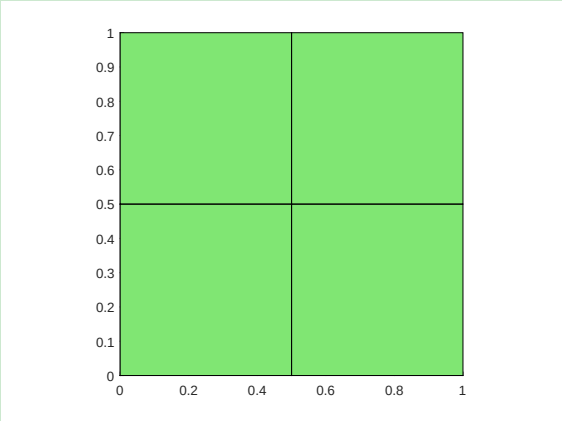
例 1.1 (三角剖分) 对前面的梯形区域可如下调用 showmesh 函数

```
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

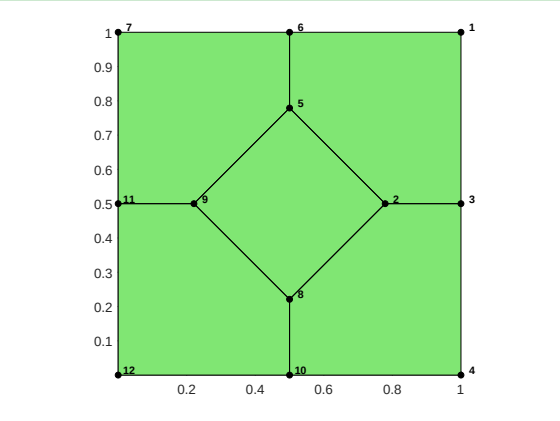


图 2. 多角形网格

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

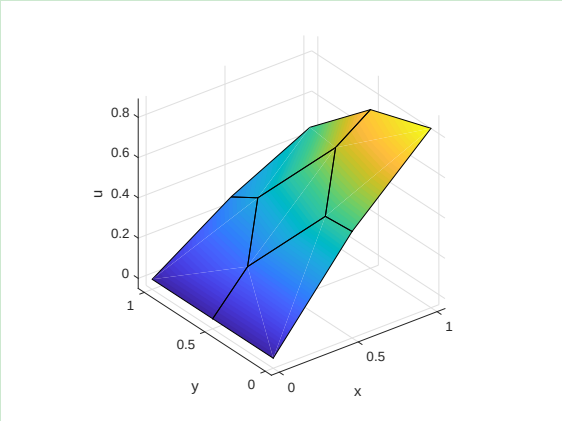
CODE 2. showsolution.m

```
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.

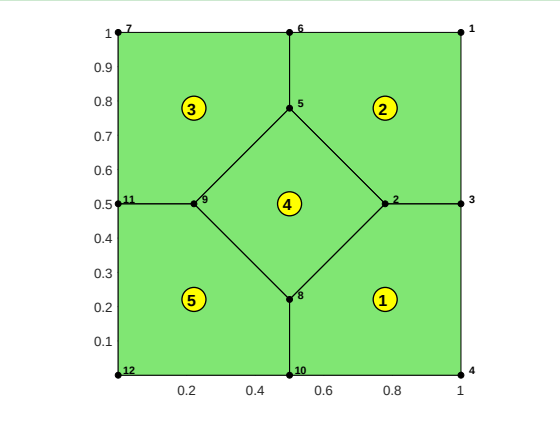


图 3. Polygonal mesh

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.

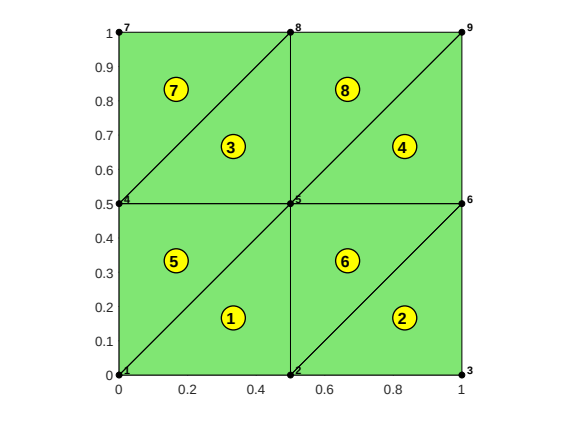
- 只要给出每条边两端的节点编号. 内部边在 `elem` 中会出现两次, 边界边只会出现一次, 我们可用 2 标记内部边, 1 标记边界边.
- 内部边在 `elem` 中会出现两次, 但它们是同一条边. 为了给定一致的标记, 我们规定每条边起点的顶点编号小于终点的顶点编号, 即 `edge(k,1)<edge(k,2)`.
- 规定三角形的第 i 条边对应第 i 个顶点 (不是必须的, 这个规定有利于网格二分程序的实现), 例如, 设第 1 个三角形顶点顺序为 [1,4,5], 那么边的顺序应是 4-5, 5-1, 1-4. 在 MATLAB 中, 有如下对应

```
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)

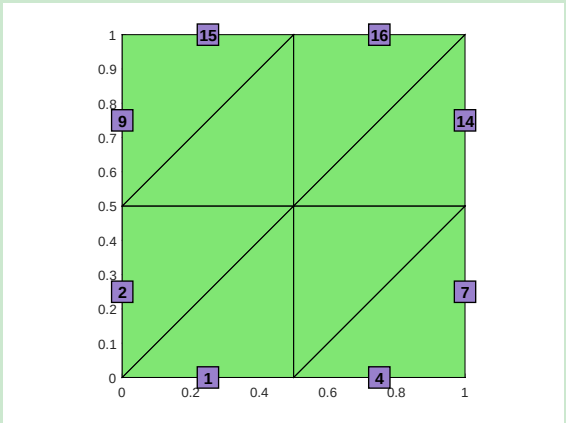
  ```
  sparse(totalEdge(:,1),totalEdge(:,2),1)
  ```

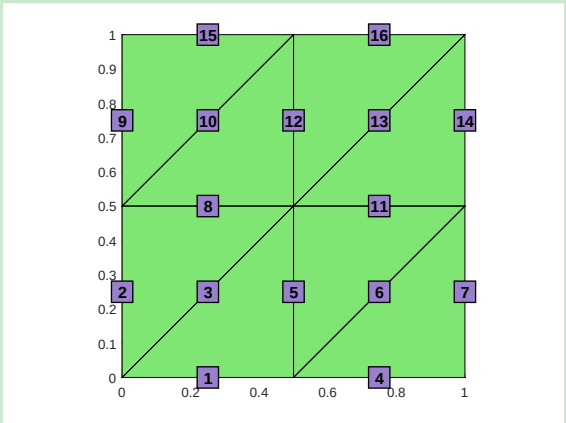
  1-5 边对应的位置 (1,5) 的值就是 2, 而 1-2 对应的位置 (1,2) 为 1, 即重复边的都是 2, 不重复的为 1. 用 `find` 可找到所有非零元素的位置及相应的值 (非零元素只有 1 和 2, 对应边界边和内部边). 显然, `sparse` 命令产生的矩阵, 第一行对应起点 1 的边, 第二行对应起点 2 的边, 等等.

- 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);
```

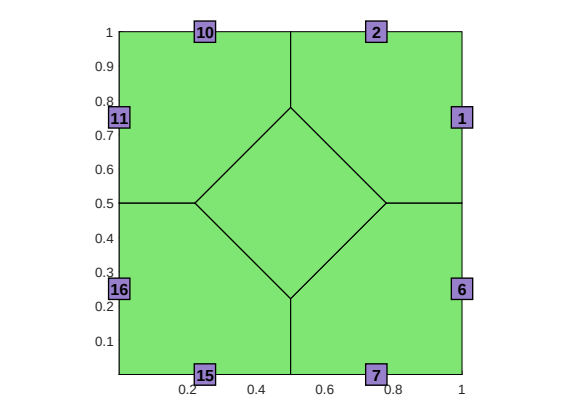
函数文件如下

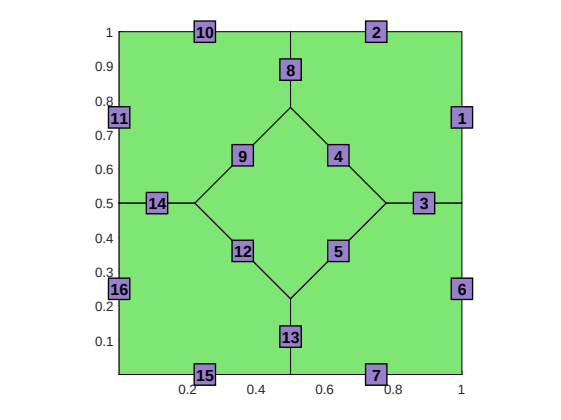
```
function findedgeTr(node,elem,bdInd)
%FindedgeTr highlights edges for triangulation
% bdEdge = 1; % boundary edge;
% other cases: all edges

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

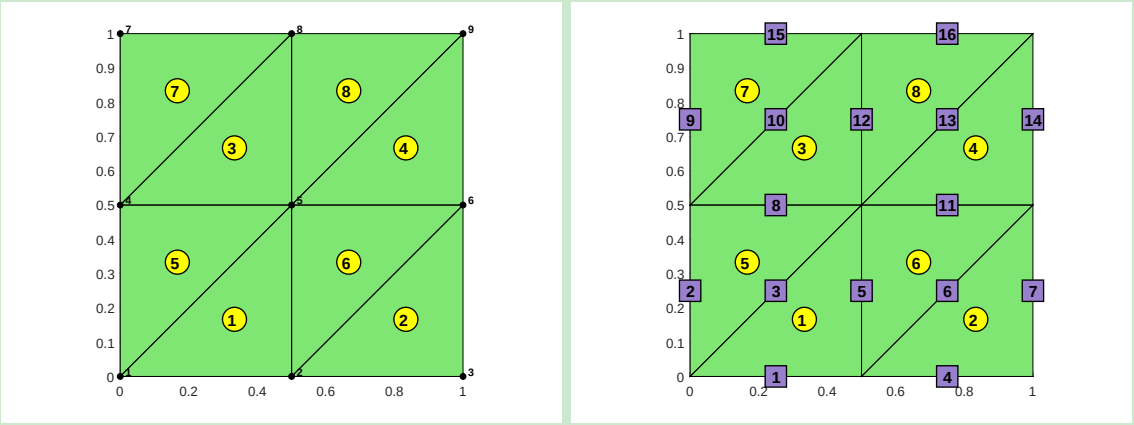
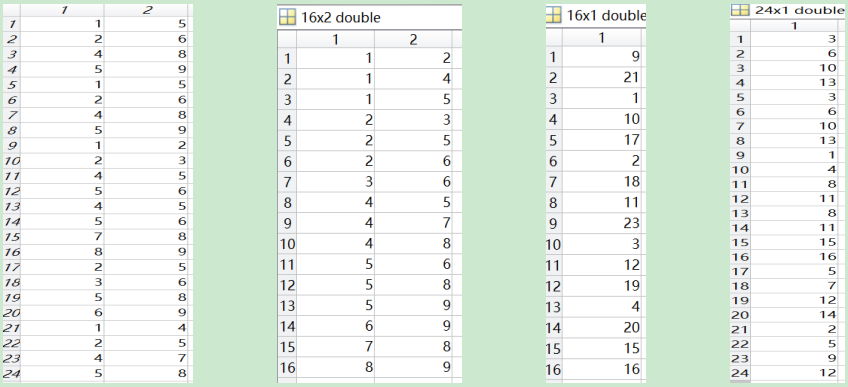


图 4. 三角剖分边的自然序号

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

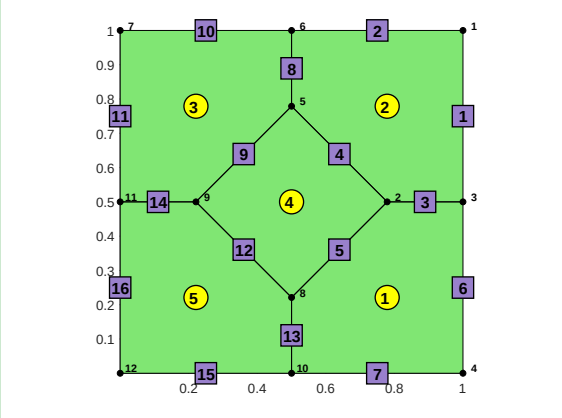
8x3 double

	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

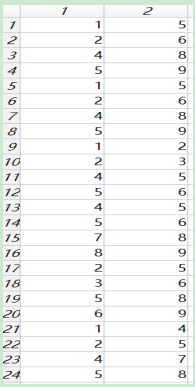
	1	2
1	1	3
2	1	6
3	3	4
4	4	10
5	6	7
6	7	11
7	10	12
8	11	12

(c) bdEdge

图 6. Auxiliary mesh data structure

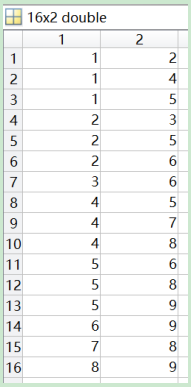
1.4.2 edge2elem 的生成

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

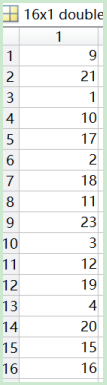
(a) totalEdge



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

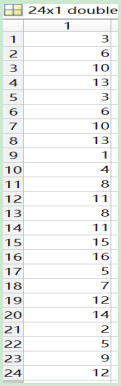
(b) edge



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

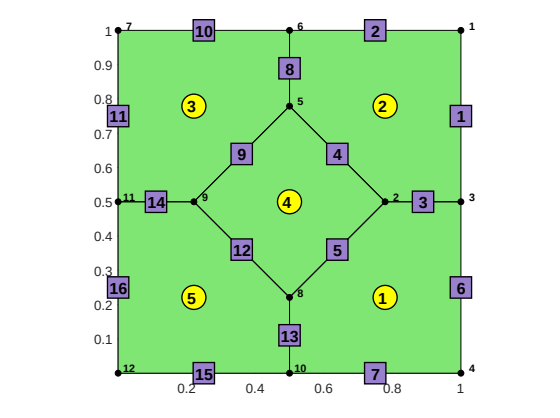


图 7. `edge2elem`

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);
```

```
17
18 else % Triangulation
19     totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
20     [¬, i1, totalJ] = unique(totalEdge,'rows');
21     elem2edge = reshape(totalJ,NT,3);
22 end
23
24 % -------- edge, bdEdge --------
25 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
26 edge = [j,i];
27 bdEdge = edge(s==1,:);
28
29 % ------- edge2elem --------
30 if iscell(elem)
31     Num = num2cell((1:NT)');    Len = num2cell(elemLen);
32     totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', ...
           false);
33     totalJelem = vertcat(totalJelem{:});
34 else
35     totalJelem = repmat((1:NT)',3,1);
36 end
37 [¬, i2] = unique(totalJ(end:-1:1),'rows');
38 i2 = length(totalEdge)+1-i2;
39 edge2elem = totalJelem([i1,i2]);
40
41 % --------- neighbor ---------
42 NE = size(edge,1);
43 ii1 = edge2elem(:,1); jj1 = (1:NE)'; ss1 = edge2elem(:,2);
44 ii2 = edge2elem(:,2); jj2 = (1:NE)'; ss2 = edge2elem(:,1);
45 label = (ii2≠ss2);
46 ii2 = ii2(label); jj2 = jj2(label); ss2 = ss2(label);
47 ii = [ii1;ii2]; jj = [jj1;jj2]; ss = [ss1;ss2];
48 neighbor = sparse(ii,jj,ss,NT,NE);
49
50
51 if ¬iscell(elem) % transform to cell
52     elem = mat2cell(elem,ones(NT,1),3);
53     elem2edge = mat2cell(elem2edge,ones(NT,1),3);
54 end
55
56 aux.node = node; aux.elem = elem;
57 aux.elem2edge = elem2edge;
58 aux.edge = edge; aux.bdEdge = bdEdge;
59 aux.edge2elem = edge2elem;
60 aux.neighbor = neighbor;
```

auxgeometry.m 函数如下

CODE 9. auxgeometry.m

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

```
2
3 % ------ elemCentroid, area, diameter -------
4 NT = size(elem,1);
5 elemCentroid = zeros(NT,2); area = zeros(NT,1); diameter = zeros(NT,1);
6 s = 1;
7 for iel = 1:NT
8     if iscell(elem)
9         index = elem{iel};
10    else
11        index = elem(iel,:);
12    end
13    verts = node(index, :); verts1 = verts([2:end,1],:);
14    area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
15    ar = 0.5*abs(sum(area_components));
16    area(iel) = ar;
17    elemCentroid(s,:) = ...
          sum((verts+verts1).*repmat(area_components,1,2))/(6*ar);
18    diameter(s) = max(pdist(verts));
19    s = s+1;
20 end
21
22 if ¬iscell(elem) % transform to cell
23    elem = mat2cell(elem,ones(NT,1),3);
24 end
25
26 aux.node = node; aux.elem = elem;
27 aux.elemCentroid = elemCentroid;
28 aux.area = area;
29 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)

```
1 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
2 edge = [j,i];
3 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

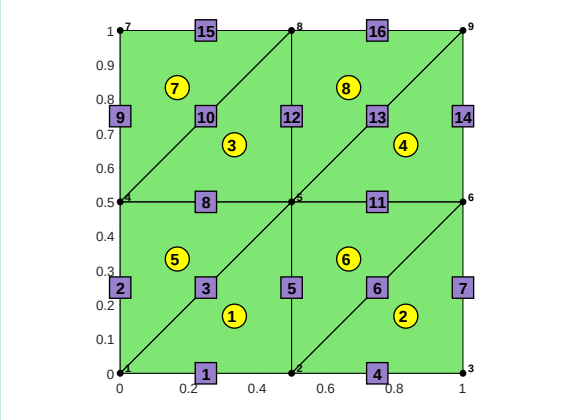


图 8. 边的自然序号

- 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 PolyMesher: 二维多角形剖分的生成

2.1 Voronoi 图

2.1.1 Voronoi 图的定义

Voronoi 图是一种空间分割算法, 它根据平面上给定的 n 个相异点将平面划分成 n 个区域, 且每个点位于一个区域. 这些点通常称为种子, 每个种子所在区域内的点到该种子的距离总是比到其他种子要近.

定义 2.1 设 $P=\{x_1,\cdots,x_n\}$ 是 $\mathbb{R}^2$ 或某个区域 Δ 内的 n 个相异点的集合, 对任意的 $y\in P$, 它的 Voronoi 单元定义为

$$V_y=\left\{x\in\mathbb{R}^2:\ \ |x-y|<|x-z|,\quad \forall z\in P\backslash\{y\}\right\}.$$

区域 Δ 的 Voronoi 划分定义为

$$\mathcal{T}(P;\Delta)=\{V_y\cap\Delta:\ \ y\in P\}.$$

显然区域 Δ 的 Voronoi 划分就是将平面的 Voronoi 划分附加上边界 $\partial\Delta$ 所得. 对平面两点, 易知 Voronoi 划分由两点之间的垂直平分线确定. 对多个点的 Voronoi 图, 我们只需要画出相邻点的垂直平分线, 这些平分线交出的若干个多边形图形就是 Voronoi 划分.

易知 Voronoi 单元若有界则必是凸的, 正因为如此它在网格剖分中很有用.

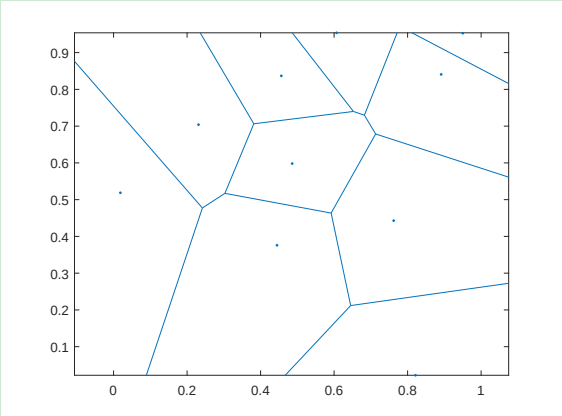
2.1.2 Voronoi 图的生成

有许多算法可以生成 Voronoi 图, 利用 Delaunay 三角剖分生成 Voronoi 图的算法是最快的. MATLAB 自带相关函数用以生成 Voronoi 图.

例 2.1 给定一些种子, 坐标分别为 `x,y`, 在 MATLAB 中可如下绘制 Voronoi 图

```
x = gallery('uniformdata',[1 10],0);
y = gallery('uniformdata',[1 10],1);
voronoi(x,y);
```

结果如下



该图是在坐标的一定范围内绘图. 从图中可以观察到垂直平分的特点.

Voronoi 函数可以有输出, 如下

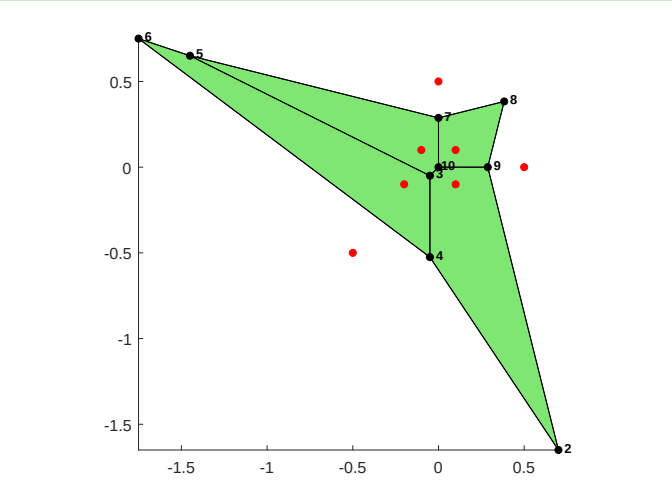
```
x = gallery('uniformdata',[1 10],0);
y = gallery('uniformdata',[1 10],1);
[vx,vy] = voronoi(x,y);
plot(x,y,'r.','MarkerSize',15), hold on
plot(vx,vy,'b-','LineWidth',1)
axis equal
xlim([min(x) max(x)])
ylim([min(y) max(y)])
```

这里, vx 有两行, 第一行是边的起点横坐标, 第二行是边的终点横坐标; 而 vy 则是相应的纵坐标.

Voronoi 命令不常用, 因为它没有给出多边形的表示, 常用如下命令

```
P = [0.5 0; 0 0.5; -0.5 -0.5; -0.2 -0.1; -0.1 0.1; 0.1 -0.1; 0.1 0.1];
[node,elem] = voronoin(P);
showmesh(node,elem); findnode(node); hold on
plot(P(:,1),P(:,2),'r.','MarkerSize',15)
```

图形如下



返回的 node 的第一个点的坐标一般都是 (Inf,Inf), 图中白色区域的种子所在的区域实际上就是与无穷远点构成的“多边形”. 例如, 左下角的种子所在的多边形为 6-1-2-4, 这里 1 是无穷原点的编号; 右侧种子的多边形为 9-2-1-8, 要注意这里的 1 对应的无穷远点应是右侧的, 只不过用一个记号标记罢了. 这样, 平面恰好被分成了 7 部分, 每个部分对应一个种子.

注 2.1 对一个凸区域, 对每一个种子关于最近的边界做对称点, 称该点为反射点, 所有反射点的集合记为 $R(P)$. 现在视 $\tilde{P} = P \cup R(P)$ 为新的种子, 考虑它们生成的 Voronoi 图.

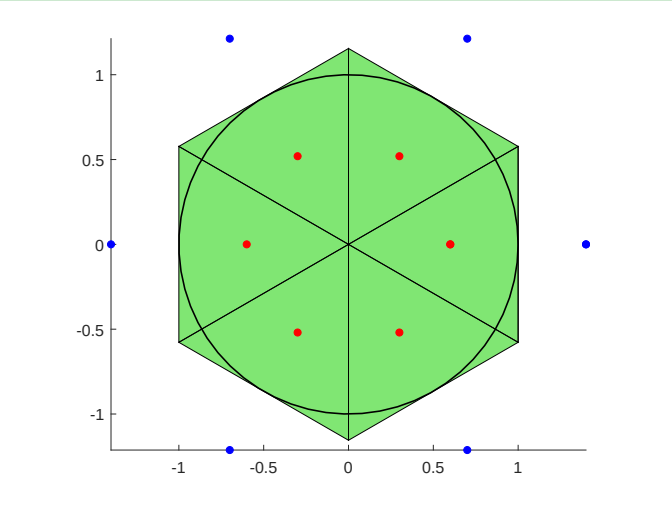
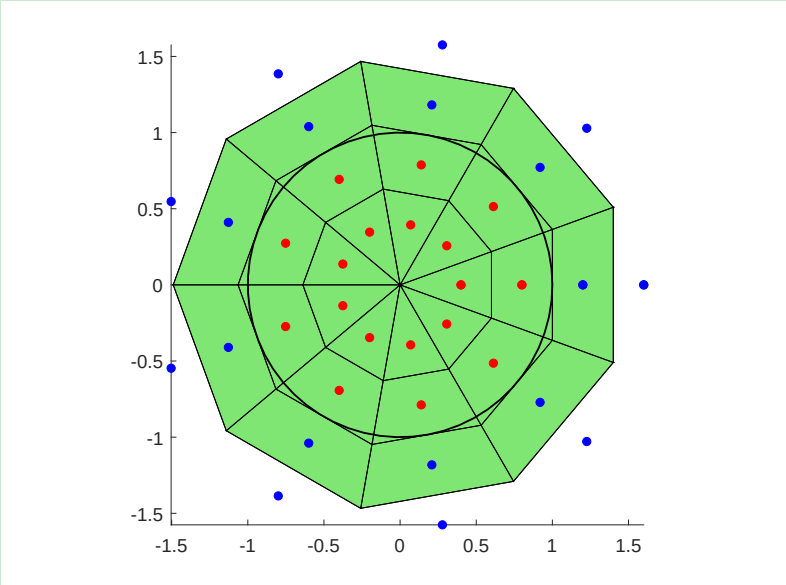


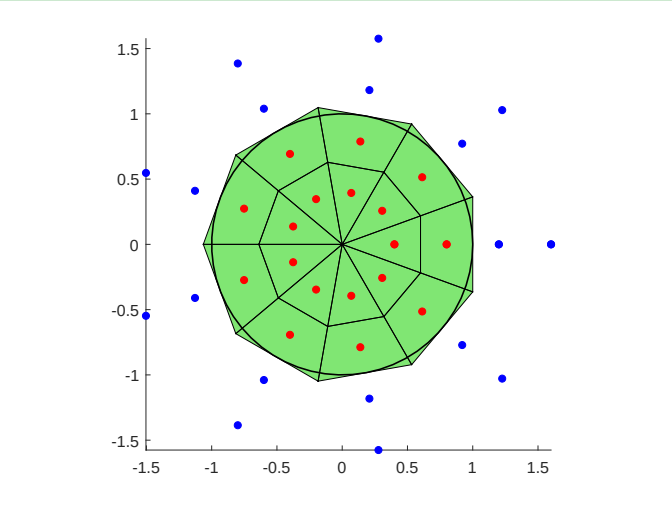
图 9. $\mathcal{T}(\tilde{P};\Delta)$

如图 9 所示, 圆是考虑的凸区域 Δ, 其内有 6 个种子, 标记为红色点, 相应的反射点为蓝色点. 注意该图给出的就是最终的剖分图. $\mathcal{T}(\tilde{P};\Delta)$ 有如下特点:

- 反射点 $R(y)$ 的 Voronoi 单元在区域外, 它与种子 y 的单元有公共边界, 且与凸区域的边界相切.
- $\mathcal{T}(\tilde{P};\Delta)$ 有 `2*NT` 个单元, 前 `NT` 个对应区域内的种子, 因而是该区域的一个覆盖.
- 当凸区域内点充分多时, 这个覆盖就是一个较好的剖分图.
- 注意上面给出的图外部的单元恰好是空白的, 它们与无穷原点构成“多角形”. 但外部也可能存在可以直接观察到的剖分, 如下图

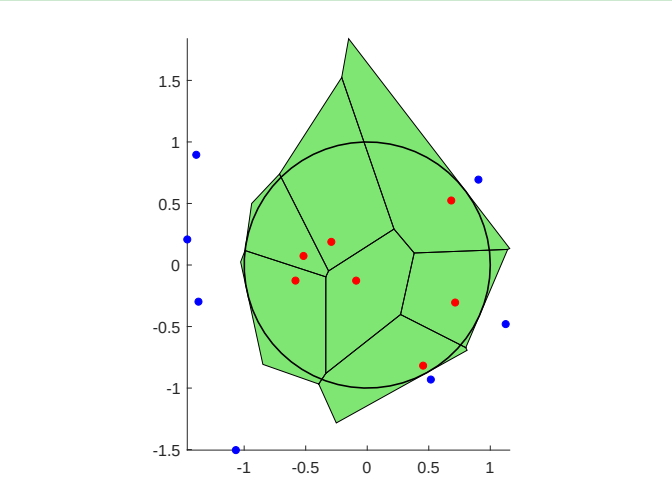


取前 `NT` 个的结果如下

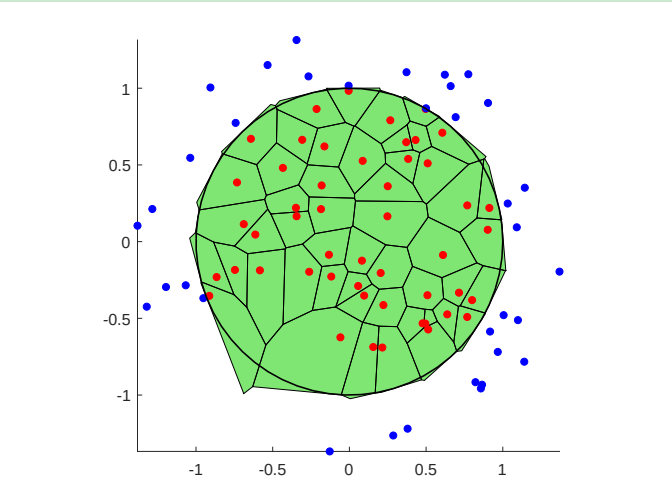


2.1.3 CVTs 及其变分描述

Voronoi 图的质量与种子的分布有关. 对前面的圆形区域, 若随机生成较少的点作为种子, 则生成的剖分图可能比较糟糕, 如下图所示



增加种子数, 例如 50 个随机种子, 结果如下



此时边界部分仍然较差 (事实上对四五百个随机种子也经常出现这种情况). 但若用均匀分布的种子, 则质量会好很多, 这在前面已经看到.

为了获得更高质量的剖分, 常采用形心 Voronoi 划分 (the centroid Voronoi tessellation, CVT). 给定密度函数 $\mu(x)$, Voronoi 单元的形心定义为

$$y_c = \frac{\int_{V_y\cap\Delta} x\mu(x)\mathrm{d}x}{\int_{V_y\cap\Delta} \mu(x)\mathrm{d}x}.$$

定义 2.2 $\mathcal{T}(P;\Delta)$ 称为一个 CVT, 如果对每个 $y\in P$, 都有

$$y = y_c = \frac{\int_{V_y\cap\Delta} x\mu(x)\mathrm{d}x}{\int_{V_y\cap\Delta} \mu(x)\mathrm{d}x}.$$

注 2.2 特别地, 当 $\mu(x)\equiv 1$ 时, y_c 就是多角形单元 $V_y\cap\Delta$ 的形心 (此时就不需要计算积分).

定义能量泛函

$$\mathcal{E}(P;\Delta) = \sum_{y\in P}\int_{V_y(P)\cap\Delta} \mu(x)|x-y|^2\mathrm{d}x,$$

注意它依赖于 P 中的点 (通过 y 的出现) 以及 Voronoi 单元. 可以证明, $\mathcal{E}(P;\Delta)$ 的极值点就是生成 CVT 的种子集合. 这是因为对任意的 $y\in P$, 能量泛函关于 y 的梯度为

$$\nabla_y\mathcal{E} = 2m_y(y-y_c), \qquad m_y = \int_{V_y\cap\Delta}\mu(x)\mathrm{d}x.$$

2.1.4 计算 CVTs 的 Lloyd 算法

定义 2.3 定义

$$L = (L_y)_{y\in P}^T:\ \mathbb{R}^{n\times 2}\to\mathbb{R}^{n\times 2},\ \ P\mapsto L(P) = (L_y(P))_{y\in P}^T,$$

其中

$$L_y(P) = \frac{\int_{V_y\cap\Delta} x\mu(x)\mathrm{d}x}{\int_{V_y\cap\Delta}\mu(x)\mathrm{d}x},$$

称其为 Lloyd 映射.

显然, 若 P 就是 CVTs 对应的种子集合, 则有 $P = L(P)$. 一般地, 给定一个初始种子集合 P_0, 考虑如下迭代

$$P_{k+1} = L(P_k),$$

则当 $k\to\infty$ 时, P_∞ 就是一个 CVT 的种子集合, 满足 $P_\infty = L(P_\infty)$, 因而这是一个不动点迭代, 称为 Lloyd 算法. 在迭代过程中, 能量泛函是下降的, 即

$$\mathcal{E}(P_{k+1};\Delta)\le\mathcal{E}(P_k;\Delta),$$

因而 Lloyd 算法可视为求能量泛函极值点的下降算法.

2.2 反射集的计算

2.2.1 符号距离函数

定义 2.4 设 $\Omega \subset \mathbb{R}^2$, 定义 $d_\Omega:\ \mathbb{R}^2 \to \mathbb{R}$ 为

$$d_\Omega(x) = s_\Omega(x) \min_{y\in\partial\Omega} |x-y|,$$

其中,

$$s_\Omega(x) = \begin{cases} -1, & x \in \Omega, \\ +1, & x \in \mathbb{R}^2\backslash\Omega. \end{cases}$$

称 d_Ω 为符号距离函数, 获得最小值的边界点称为关于 x 的最近边界点.

显然有

$$\overline{\Omega} = \left\{x \in \mathbb{R}^2:\quad d_\Omega(x) \le 0\right\}, \qquad \partial\Omega = \left\{x \in \mathbb{R}^2:\quad d_\Omega(x) = 0\right\}.$$

对圆心在 x_0, 半径为 r 的球, 易知

$$d_\Omega(x) = |x - x_0| - r.$$

当 Ω 光滑时, $\nabla d_\Omega(x)$ 是最近边界点处的单位法向量, 例如对圆, 有 $\nabla d_\Omega(x) = \mathrm{sgn}(x - x_0)$. 一般地, 对几乎所有的 $x \in \mathbb{R}^2$, 有 $|\nabla d_\Omega(x)| = 1$.

通过图形观察可知, x 关于最近边界点的反射点为

$$R_\Omega(x) = x - 2d_\Omega(x)\nabla d_\Omega(x). \tag{2.1}$$

对复合区域, 通常有

$$d_{\Omega_1\cup\Omega_2}(\mathbf{x}) = \min\left(d_{\Omega_1}(\mathbf{x}), d_{\Omega_2}(\mathbf{x})\right),$$

$$d_{\Omega_1\cap\Omega_2}(\mathbf{x}) = \max\left(d_{\Omega_1}(\mathbf{x}), d_{\Omega_2}(\mathbf{x})\right),$$

$$d_{\mathbb{R}^2\backslash\Omega_1}(\mathbf{x}) = -d_{\Omega_1}(\mathbf{x}).$$

2.2.2 初始种子的生成

设 $\Omega \subset \mathbb{R}^2$ 为有界区域, 且相应的符号距离函数 $d_\Omega(x)$. 为了方便, 以下考虑单位圆. 我们可如下生成随机种子:

1. 给定种子个数, 即剖分单元数 NT. 用一个合适的矩形将区域覆盖, 对单位圆可用正方形 $[-1,1]^2$ 覆盖.

2. 在矩形内随机生成种子, 并筛选在 Ω 内的, 可利用符号距离函数判断.

区域的基本定义如下

```
function x = Circle_Domain(Demand,P)
BdBox = [-1 1 -1 1];
switch(Demand)
    case('Dist');  x = DistFnc(P);
    case('BdBox'); x = BdBox;
end
% the signed distance function
```

```
function d = DistFnc(P)
xc = 0; yc = 0; r = 1;
d = sqrt((P(:,1)-xc).^2+(P(:,2)-yc).^2)-r;
```

随机种子如下生成

```
% ------------ Generate random pointset -----------
Domain = @Circle_Domain;
NT = 30;
P = zeros(NT,2); BdBox = Domain('BdBox'); s = 0;
while s < NT
    xy(:,1) = (BdBox(2)-BdBox(1))*rand(NT,1)+BdBox(1);
    xy(:,2) = (BdBox(4)-BdBox(3))*rand(NT,1)+BdBox(3);
    d = Domain('Dist',xy);
    I = find(d<0);        % index of seeds inside the domain
    NumAdded = min(NT-s,length(I));  % number of seeds that can be added
    P(s+1:s+NumAdded,:) = xy(I(1:NumAdded),:);
    s = s+NumAdded;
end
```

我们也可把矩形划分成 $N_x \times N_y$ 个小矩形, 然后在每个小矩形中选择满足条件的点. 为了方便, 取小矩形的中心, 对不满足要求的中心, 我们直接去掉. 称这种初始种子为均匀分布的, 如下获取

```
% ------------ Generate uniform pointset -----------
Nx = 5; Ny = 5;
x = linspace(BdBox(1),BdBox(2),Nx+1)';
y = linspace(BdBox(3),BdBox(4),Ny+1)';
xc = (x(1:end-1)+x(2:end))/2; yc = (y(1:end-1)+y(2:end))/2;
[X,Y] = ndgrid(xc,yc); P = [X(:),Y(:)];
d = Domain('Dist',P);  P = P(d<0,:);
NT = size(P,1);
```

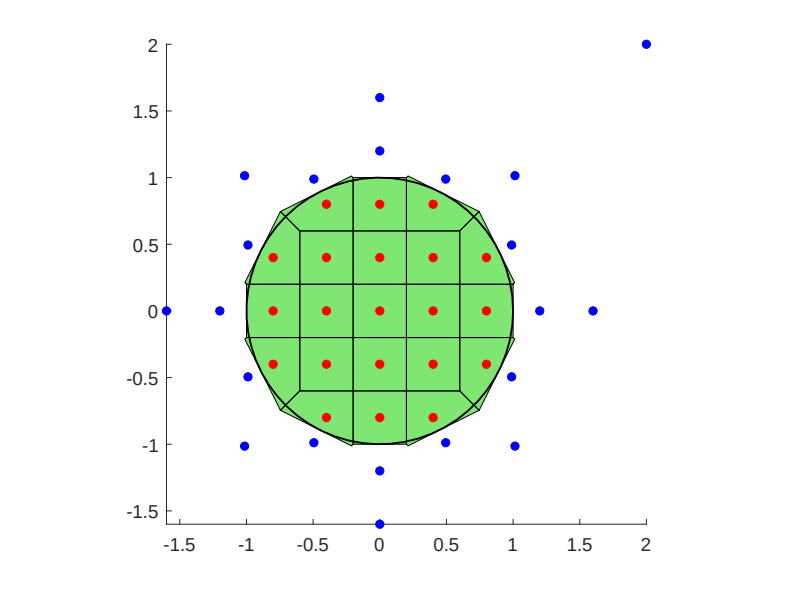
以上过程编写成函数文件

```
function P = PolyMesher_init_Pointset(Domain,varargin)

nar = length(varargin); BdBox = Domain('BdBox');
% ------------ Generate random pointset -----------
if nar==1
    NT = varargin{1};
    P = zeros(NT,2);  s = 0;
    while s < NT
        xy(:,1) = (BdBox(2)-BdBox(1))*rand(NT,1)+BdBox(1);
        xy(:,2) = (BdBox(4)-BdBox(3))*rand(NT,1)+BdBox(3);
        d = Domain('Dist',xy);
        I = find(d<0);        % index of seeds inside the domain
        NumAdded = min(NT-s,length(I));  % number of seeds that can be added
        P(s+1:s+NumAdded,:) = xy(I(1:NumAdded),:);
        s = s+NumAdded;
    end
end
% ------------ Generate uniform pointset -----------
```

```
if nar==2
    Nx = varargin{1}; Ny = varargin{2};
    x = linspace(BdBox(1),BdBox(2),Nx+1)';
    y = linspace(BdBox(3),BdBox(4),Ny+1)';
    xc = (x(1:end-1)+x(2:end))/2; yc = (y(1:end-1)+y(2:end))/2;
    [X,Y] = ndgrid(xc,yc); P = [X(:),Y(:)];
    d = Domain('Dist',P);  P = P(d<0,:);
end
```

2.2.3 光滑边界凸区域反射集的计算

光滑边界凸区域的典型代表是圆形区域, 它的符号距离函数比较容易获得. 反射点用公式 (2.1) 计算, 这里的梯度用导数的定义计算.



上图是直接对所有种子进行反射获得的剖分图, 该图正上方的两个反射点对应同一个边界切线, 其中一个可认为是无效反射. 注意, 我们考虑反射集的目的是给出区域边界的近似. 另外一种特殊的情况是, 某些种子的最近边界点不唯一, 例如图中圆心处的点. 考虑到内部大部分点的反射对边界剖分作用不大, 为此我们只对边界附近的点进行反射. 这种做法大大减少计算量, 而且也可避免以上情形的发生.

为此引入参数

$$\alpha(n,\Omega) := c\left(\frac{|\Omega|}{n}\right)^{1/2},$$

其中 n 是种子的个数, 且只对满足

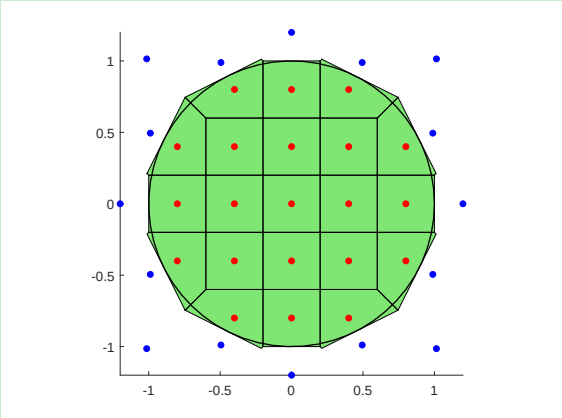
$$d_\Omega(y) < \alpha(n,\Omega) \tag{2.2}$$

的种子进行反射. 这里 c 为比例常数且大于 1, 从而 α 比单元的平均宽度大. 程序如下

```
clc;clear;close all
% ------- Generate intial pointset --------
Domain = @Circle_Domain;
% NT = 50;
% P = PolyMesher_init_Pointset(Domain,NT);
```

```
Nx = 5; Ny = 5;
P = PolyMesher_init_Pointset(Domain,Nx,Ny);
NT = size(P,1);

% ---------- Compute the reflection pointset ---------------
eps = 1e-8; c = 1.1;
BdBox = Domain('BdBox');
Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
Alpha = c*sqrt(Area/NT);

d = Domain('Dist',P);
I = abs(d)<Alpha;  % logical index of seeds near the bdry
n1 = (Domain('Dist',P+[eps,0])-d)/eps;
n2 = (Domain('Dist',P+[0,eps])-d)/eps;
R_P = P(I,:) - 2*[n1(I),n2(I)].*d(I);

% ------------ Voronoi -------------------
PP = [P; R_P];
[node,elem] = voronoin(PP);
elem = elem(1:NT);
showmesh(node,elem); %findnode(node);
hold on
plot(P(:,1),P(:,2),'r.',R_P(:,1),R_P(:,2),'b.','MarkerSize',15)
t = linspace(0,2*pi,60)';
plot(cos(t),sin(t),'k-','linewidth',1)
```

取 $c = 1.1$, 结果如下



2.2.4 分段光滑边界凸区域反射集的计算

典型代表是矩形区域, 此时 BdBox 就可选为该矩形区域. 设矩形为 $[a_1, b_1] \times [a_2, b_2]$, 对左侧直线, 它的右侧为内部, 符号距离函数为

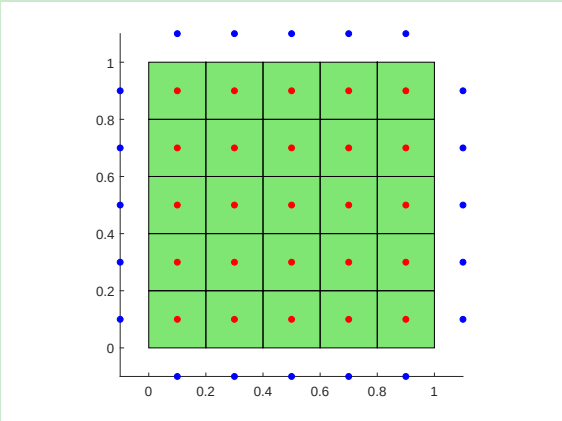
$$d_1(\boldsymbol{x}) = a_1 - \boldsymbol{x}(1),$$

类似下侧、右侧和上侧直线的符号距离函数分别为

$$d_2(\boldsymbol{x}) = b_1 - \boldsymbol{x}(2), \quad d_3(\boldsymbol{x}) = \boldsymbol{x}(1) - a_2, \quad d_4(\boldsymbol{x}) = \boldsymbol{x}(2) - a_2.$$

整体区域是这些半平面的交, 从而

$$d(x) = \max\left\{d_1(x), d_2(x), d_3(x), d_4(x)\right\}.$$



对分段光滑边界的区域, 我们将种子对每个边界做反射 (满足要求 (2.2)). 此时, 某些点可能反射多次, 例如上图中矩形角点附近的种子. 这种做法其实是合理的, 因为如果只反射一个边, 那么该角点处的拓扑信息可能很难捕捉. 通过反射两个边界, 上图获得的是标准的四边形剖分. 由于对每个边进行反射, 因此我们要保留每个边对应的符号距离函数. 这样, 矩形区域如下定义

```
function x = Rectangle_Domain(Demand,P)
BdBox = [0 1 0 1];
switch(Demand)
    case('Dist');  x = DistFnc(P,BdBox);
    case('BdBox'); x = BdBox;
end
% the signed distance function
function d = DistFnc(P,BdBox)
a1 = BdBox(1); a2 = BdBox(2); b1 = BdBox(3); b2 = BdBox(4);
d = [a1-P(:,1), b1-P(:,2), P(:,1)-a2,  P(:,2)-b2];
d = [d, max(d,[],2)];
```

这里, d 的前 4 列是分段边的符号距离函数, 而最后一个是整体符号距离函数, 用以判断点的内外.

如下生成初始剖分

```
clc;clear;close all
% ------- Generate intial pointset --------
Domain = @Rectangle_Domain;
Nx = 5; Ny = 5;
P = PolyMesher_init_Pointset(Domain,Nx,Ny);
NT = size(P,1);

% ---------- Compute the reflection pointset ---------------
eps = 1e-8; eta = 0.9; c = 1.5;
BdBox = Domain('BdBox');
Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
Alpha = c*sqrt(Area/NT);

d = Domain('Dist',P);
```

```
NBdrySegs = size(d,2)-1;          %Number of constituent bdry segments
n1 = (Domain('Dist',P+[eps,0])-d)/eps;
n2 = (Domain('Dist',P+[0,eps])-d)/eps;
I = abs(d(:,1:NBdrySegs))<Alpha;  %Logical index of seeds near the bdry
P1 = repmat(P(:,1),1,NBdrySegs);  %[NT x NBdrySegs] extension of P(:,1)
P2 = repmat(P(:,2),1,NBdrySegs);  %[NT x NBdrySegs] extension of P(:,2)
R_P = [P1(I), P2(I)] - 2*[n1(I), n2(I)].*d(I);

% ------------ Voronoi -------------------
PP = [P; R_P];
[node,elem] = voronoin(PP,{'Qbb','Qz'});
elem = elem(1:NT);
showmesh(node,elem); %findnode(node);
hold on
plot(P(:,1),P(:,2),'r.',R_P(:,1),R_P(:,2),'b.','MarkerSize',15)
```

这里,

- `d` 的第 1 列是所有种子相应于第 1 条边的符号距离, 其他列类似. `I` 则是近边条件的逻辑数组.
- P1 是横坐标, 复制 4 列以应用逻辑数组取值, 注意 `P1(I)` 取出值后排成一列.
- `d(I)` 是所有近边种子的符号距离.

2.2.5 非凸区域反射集的计算

对非凸区域, 若种子距离边界较远, 则反射点可能仍位于区域内, 或与其他种子的反射产生干扰. 可如下解决这个问题:

- 对反射点可能位于区域内, 我们只需要判断符号距离函数的符号, 即排除 $d_{\Omega_i}(\bar{y}) > 0$ 的反射点 $\bar{y} = R_{\Omega_i}(y)$.
- 对第二种情形, 可选取满足如下要求的反射点

$$|d_{\Omega}(\bar{y})| > \eta\,|d_{\Omega_i}(y)|\,, \quad 0 < \eta < 1.$$

为此, 前面的反射集合后面只需要添加语句

```
d_R_P = Domain('Dist',R_P);
J = d_R_P(:,end)>0 & abs(d_R_P(:,end))≥eta*abs(d(I));
R_P = R_P(J,:); R_P = unique(R_P,'rows');
```

2.2.6 反射集计算程序

为了一般性, 我们把圆形区域的 `d` 也按照分段区域一样设置, 即添加语句 `d = [d,d];` 这样, 计算反射集合的程序如下

```
function R_P = PolyMesher_Reflect(P,Domain)

% ---------- Compute the reflection pointset ---------------
eps = 1e-8; eta = 0.9; c = 1.5; NT = size(P,1);
```

```
BdBox = Domain('BdBox');
Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
Alpha = c*sqrt(Area/NT);

d = Domain('Dist',P);
NBdrySegs = size(d,2)-1;          %Number of constituent bdry segments
n1 = (Domain('Dist',P+[eps,0])-d)/eps;
n2 = (Domain('Dist',P+[0,eps])-d)/eps;
I = abs(d(:,1:NBdrySegs))<Alpha;  %Logical index of seeds near the bdry
P1 = repmat(P(:,1),1,NBdrySegs);  %[NT x NBdrySegs] extension of P(:,1)
P2 = repmat(P(:,2),1,NBdrySegs);  %[NT x NBdrySegs] extension of P(:,2)
R_P = [P1(I), P2(I)] - 2*[n1(I), n2(I)].*d(I);
d_R_P = Domain('Dist',R_P);
J = d_R_P(:,end)>0 & abs(d_R_P(:,end))≥eta*abs(d(I));
R_P = R_P(J,:); R_P = unique(R_P,'rows');
```

2.3 Lloyd 迭代

根据前面的说明, 在取 $\mu(x)=1$ 时, Voronoi 迭代就是计算单元的重心, 它的计算在辅助数据结构中已经说明. 如下获取 Voronoi 单元的数据 `node` 和 `elem`, 从而计算重心.

```
function [Pc,Err,node,elem] = PolyMesher_VoroCentroid(P,R_P)

% Compute the centroid of Voronoi cell
[node,elem] = voronoin([P;R_P],{'Qbb','Qz'});
NT = size(P,1);
Pc = zeros(NT,2); A = zeros(NT,1);
for iel = 1:NT
    vx = node(elem{iel},1);  vy = node(elem{iel},2); nv = length(elem{iel});
    vxS = vx([2:nv,1]); vyS = vy([2:nv,1]);
    temp = vx.*vyS - vy.*vxS;
    A(iel) = 0.5*sum(temp);
    Pc(iel,:) = 1/(6*A(iel,1))*[sum((vx+vxS).*temp),sum((vy+vyS).*temp)];
end
Area = sum(abs(A));
Err = sqrt(sum((A.^2).*sum((Pc-P).^2,2)))*NT/Area^1.5;
```

2.4 获取网格剖分的基本数据集

`elem` 的前 `NT` 个对应内部种子, 它就是连通性信息. 但 `node` 还有额外的信息, 它包含反射集对应的多边形节点 (包含无穷原点), 为此必须删除这些节点, 并将节点重新编号.

- 从 `elem = elem(1:NT)` 中可获取网格节点编号

  ```
  [id,¬,totalid] = unique(horzcat(elem{:})');
  ```

  这里, `id` 是去除重复的原始节点编号, 而 `totalid` 记录的是每个节点在 `id` 中的自然序号 (即索引), 该自然序号作为节点的新编号.

- 显然新的 `node` 为

```
node = node(id,:);
```

把 totalid 恢复为 elem 的模式就获得新的 elem

```
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
```

上面通过 elem 恢复自然序号的过程可总结为如下函数

```
function [node,elem] = PolyMesher_Reorder(NT,node,elem)
elem = elem(1:NT)';
[id,¬,totalid] = unique(horzcat(elem{:})');
node = node(id,:);
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
```

对随机初始种子, 若画出网格图并标上节点序号, 则会发现边界上的一些点出现多次编号, 这是由一些很短的边造成的. 现在考虑移除这些短边, 即把它退化为单个点.

- 根据讨论数据结构的过程, 我们可获得所有的一维边集合 edge

```
T1 = cellfun(@(verts) [verts(2:end),verts(1)], elem, 'UniformOutput', ...
    false);
v0 = horzcat(elem{:})'; v1 = horzcat(T1{:})';
totalEdge = sort([v0,v1],2);
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
```

- 我们将删除如下的边

$$h_e \le \eta h, \quad h = \max\{h_e\},$$

其中 η 是比例常数, 计算中取 $\eta = 0.1$, 即低于最大边长度 $1/10$ 的边视为很短的边.

```
z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
he = sqrt(sum((z1-z2).^2,2));
ir = find(he≤0.1*max(he));
nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices
```

这里 nv1, nv2 是要删除的边的起点和终点编号.

- 我们将把网格剖分中要删除边的终点编号替换为起点编号, 从而去除短边

```
[¬,¬,totalid] = unique(horzcat(elem{:})');
for i = 1:length(nv2)
    totalid(totalid==nv2(i)) = nv1(i);
end
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
```

要注意此时的 elem 的一些单元含有重复节点编号, 可如下去除一个并保留原顺序

```
for iel = 1:NT
```

```
    index = elem{iel};
    [¬,i1] = unique(index);
    elem{iel} = index(sort(i1));
end
```

再次恢复顺序即可.

上面的说明总结为如下程序

```
function [node,elem] = PolyMesher_rm_smalledge(NT,node,elem)
% remove small edges

[node,elem] = PolyMesher_Reorder(NT,node,elem);
T1 = cellfun(@(verts) [verts(2:end),verts(1)], elem, 'UniformOutput', false);
v0 = horzcat(elem{:})'; v1 = horzcat(T1{:})';
totalEdge = sort([v0,v1],2);
[i,j] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
he = sqrt(sum((z1-z2).^2,2));
ir = find(he≤0.1*max(he));
nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices
[¬,¬,totalid] = unique(horzcat(elem{:})');
for i = 1:length(nv2)
    totalid(totalid==nv2(i)) = nv1(i);
end
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
for iel = 1:NT
    index = elem{iel};
    [¬,i1] = unique(index);
    elem{iel} = index(sort(i1));
end
[node,elem] = PolyMesher_Reorder(NT,node,elem);
```

上面的过程暂时还未优化.

2.5 获取网格剖分

见 tool 中 meshfun.m 函数. 这部分内容后续进行优化. 一些其他区域的例子放在 PolyMesher-Modified 文件夹中.