Mesh Generalization and Mesh Optimization

Terence YUE Yu

MATLAB version: $\geq$ R2015a

目　录

第一章　网格的图示与标记

1.1　网格与解的图示

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

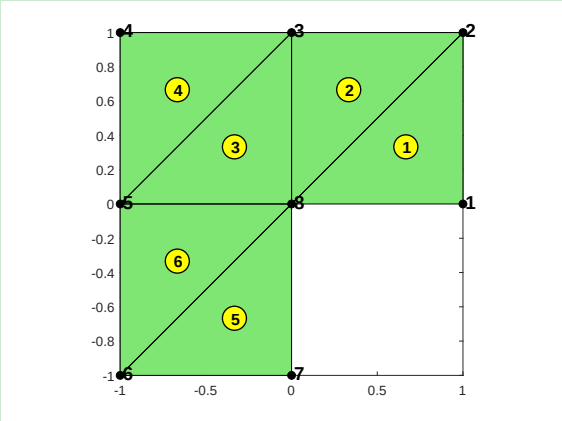


图 1.1. L 形区域的剖分

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.

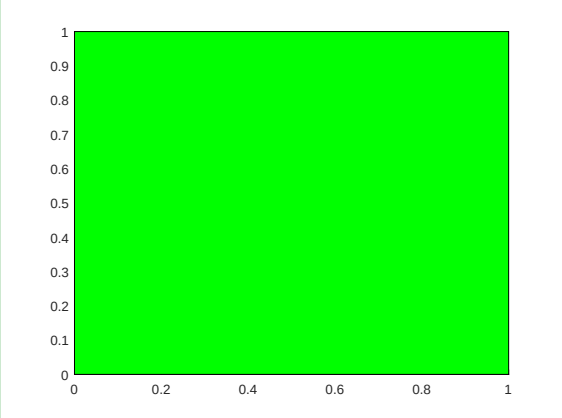
6x3 double

	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. 对多角形剖分, elem 为元胞数组, 每个元胞存储一个单元.

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, 为此本文只考虑 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',elem,'Vertices',node,'FaceColor','g')
```

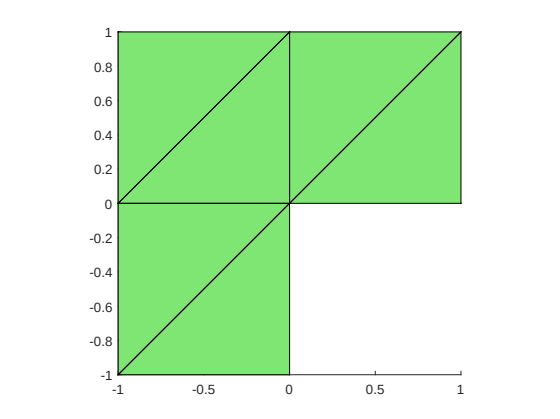
对多个相同类型的单元, 如下

```
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;
```

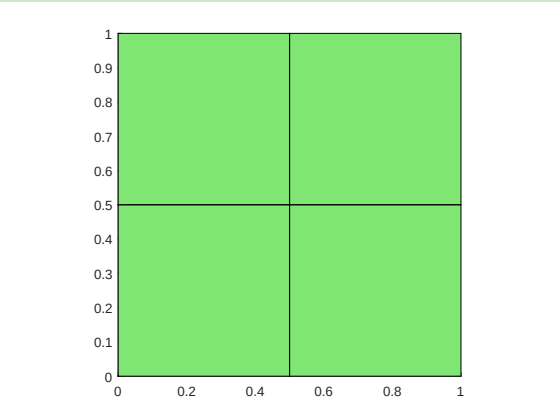
例 1.1 (三角剖分)　对前面的梯形区域, 如下调用 showmesh 函数

```
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, 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 (避免循环语句逐个), 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. 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, 如果希望输出的是多种类型的元素, 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

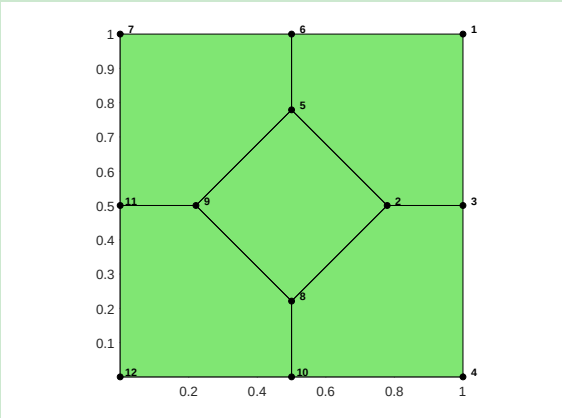


图 1.2. 多角形网格

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 1.1. 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

showsolution 函数绘制解的网格图, 它的程序如下

```
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');

```

```
16 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] 就是画图的三维点坐标.
- 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修改一下以适合多角形剖分, 如下

CODE 1.2. showsolution.m

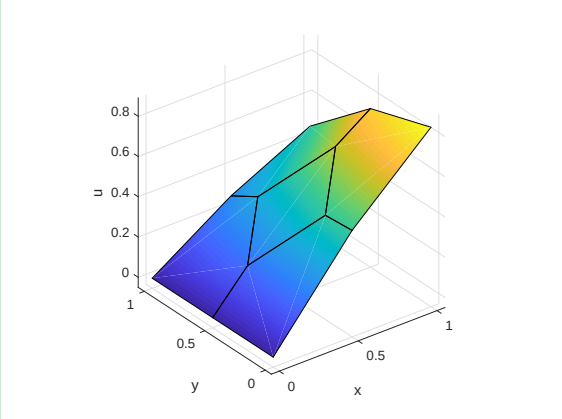
```
1  function showsolution(node,elem,u)
2  %Showsolution displays the solution corresponding to a mesh given by [node,elem] in 2-D.
3
4  data = [node,u];
5  if ¬iscell(elem)
6      patch('Faces', elem,...
7          'Vertices', data,...
8          'FaceColor', 'interp',...
9          'CData', u / max(abs(u)) );
10 else
11     max_n_vertices = max(cellfun(@length, elem));
12     padding_func = @(vertex_ind) [vertex_ind,...
13         NaN(1,max_n_vertices-length(vertex_ind))];  % function to pad the vacancies
14     tpad = cellfun(padding_func, elem, 'UniformOutput', false);
15     tpad = vertcat(tpad{:});
16     patch('Faces', tpad,...
17         'Vertices', data,...
18         'FaceColor', 'interp',...
19         '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, 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 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 1.3. 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 网格的标记

1.2.1 节点标记

可如下给出图 1.2 中的节点编号

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

函数文件如下

CODE 1.4. 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 当然也可简单改动以适用于三维情形, 这里略, 见 GitHub 上传程序 (tool 文件夹内).

1.2.2 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 (重心的计算说明略).

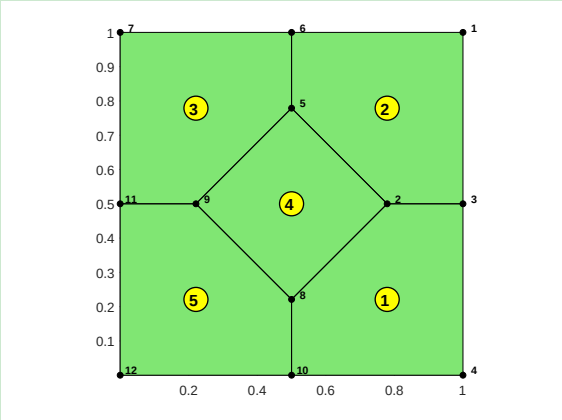


图 1.3. Polygonal mesh

主程序如下

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

单元标记函数如下

CODE 1.5. 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.

1.2.3 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.

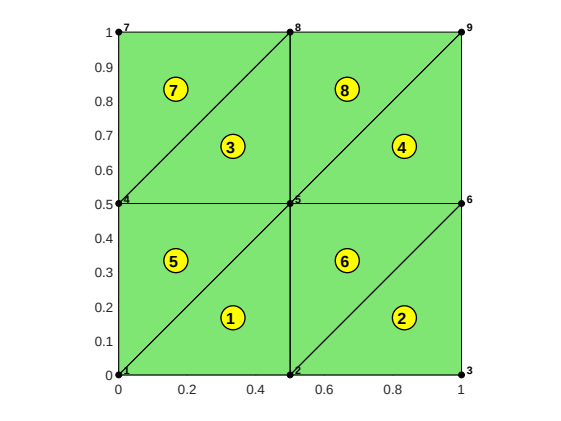
- 只要给出每条边两端的节点编号. 内部边在 elem 中会出现两次, 边界边只会出现一次, 我们可用 2 标记内部边, 1 标记边界边.
- 内部边在 elem 中会出现两次, 但它们是同一条边. 为了给定一致的标记, 我们规定每条边起点的顶点编号小于终点的顶点编号, 即 edge(k,1)<edge(k,2).
- 规定三角形的第 i 条边对应第 i 个顶点 (不是必须的, 这个规定有利于网格二分程序的实现), 例如, 设第 1 个三角形顶点顺序为 [1,4,5], 那么边的顺序应是 4-5, 5-1, 1-4. 在 MATLAB 中, 有如下对应

```
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)

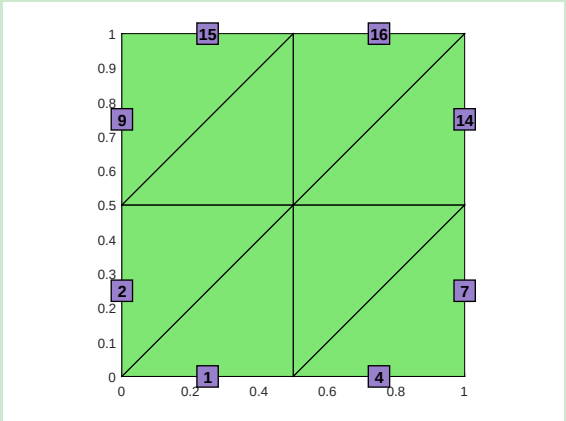
  ```
  sparse(totalEdge(:,1),totalEdge(:,2),1)
  ```

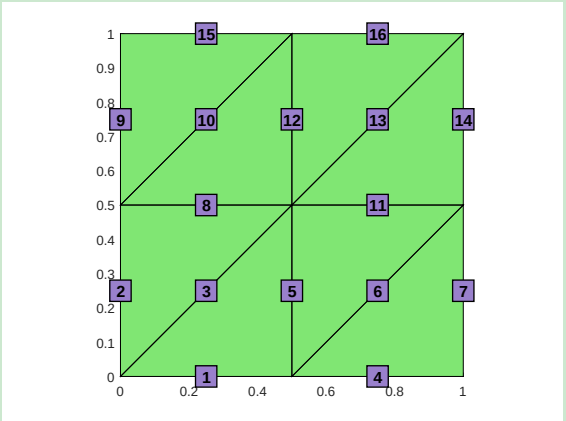
  1-5 边对应的位置 (1,5) 的值就是 2, 而 1-2 对应的位置 (1,2) 为 1, 即重复边的都是 2, 不重复的为 1. 用 `find` 可找到所有非零元素的位置及相应的值 (非零元素只有 1 和 2, 对应边界边和内部边). 显然, `sparse` 命令产生的矩阵, 第一行对应起点 1 的边, 第二行对应起点 2 的边, 等等.

- 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

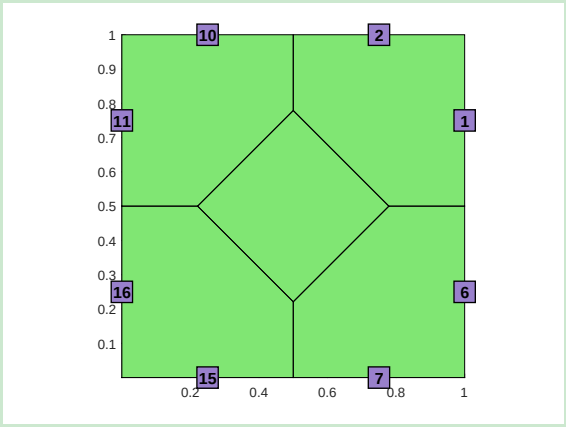
```
function findedgeTr(node,elem,bdInd)
%FindedgeTr highlights edges for triangulation
% bdEdge = 1; % boundary edge;
% other cases: all edges

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');
```

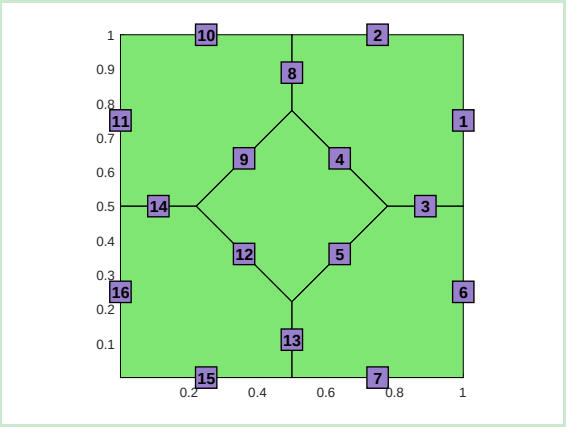
注意因为前面进行了转置, `edge = [j,i]`.

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 1.3 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 1.6. 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.

第二章　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

本章针对一般的多角形剖分给出需要的数据结构与几何量.

2.1　辅助数据结构

我们的数据结构包括

表 2.1. 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

2.1.1　elem2edge 的生成

`elem2edge` 按单元记录每条边的自然序号, 这里同时会给出 `edge`, `bdEdge`.

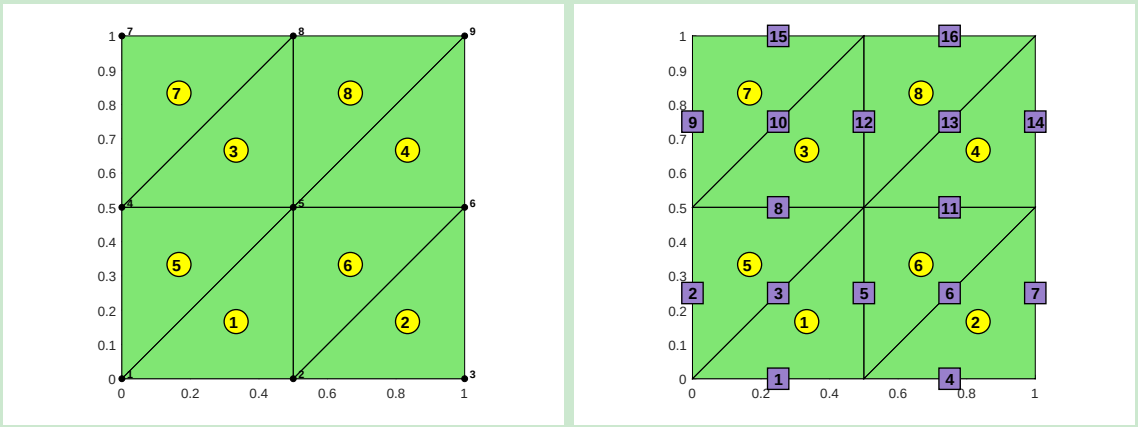
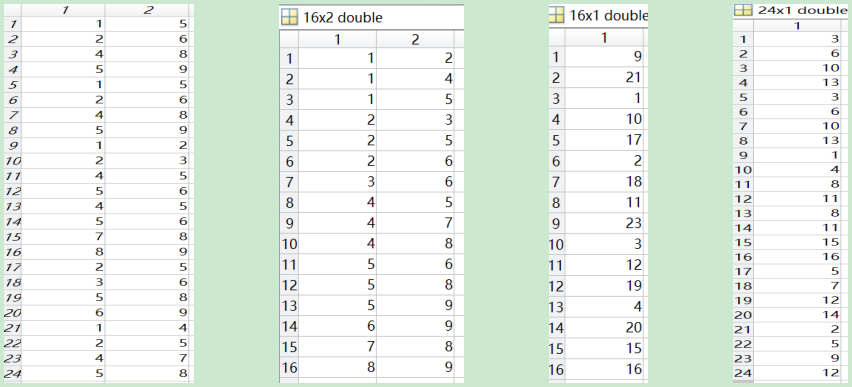


图 2.1. 三角剖分边的自然序号

考虑图 2.1 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 2.2(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 2.2. 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 2.2(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

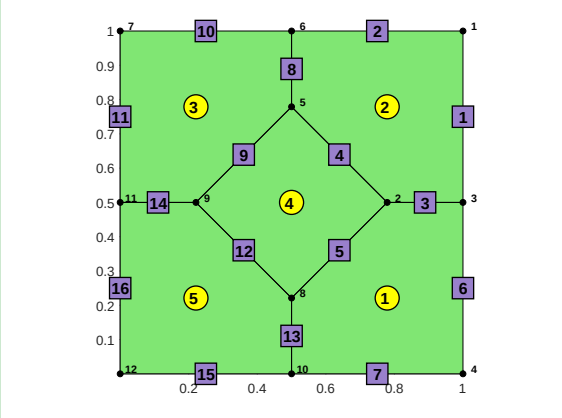
8x3 double

	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',1,elemLen)';

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
```

```
31
32 % -------- edge, bdEdge --------
33 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
34 edge = [j,i];
35 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

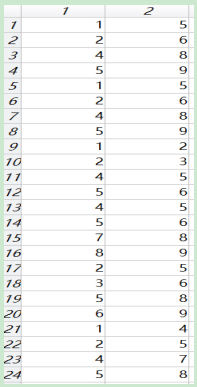
	1	2
1	1	3
2	1	6
3	3	4
4	4	10
5	6	7
6	7	11
7	10	12
8	11	12

(c) bdEdge

图 2.3. Auxiliary mesh data structure

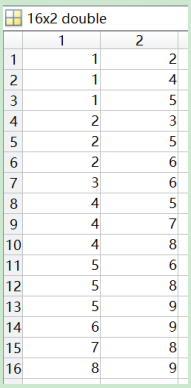
2.1.2 edge2elem 的生成

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

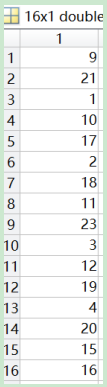
(a) totalEdge



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

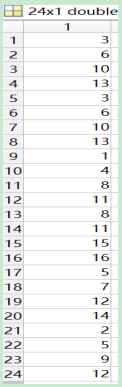
(b) edge



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` 记录了右单元边, 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

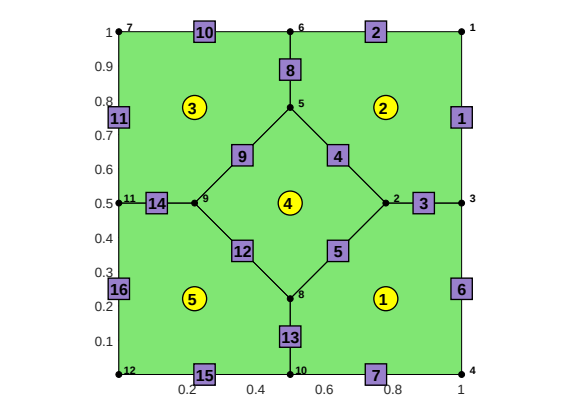


图 2.4. edge2elem

例如, 序号 12 的边连接的两个单元编号为 4 和 5.

注 2.1 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.

2.1.3 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 2.2 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.

2.2 网格相关的几何量

几何量包括

表 2.2. 几何量

`Centroid`	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 -------
Centroid = 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;
    Centroid(s,:) = sum((verts+verts1).*repmat(area_components,1,2))/(6*ar);
    diameter(s) = max(pdist(verts));
    s = s+1;
end
```

2.3 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 2.1. 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 2.2. auxgeometry.m

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

% ------ elemCentroid, area, diameter -------
NT = size(elem,1);
Centroid = 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;
    Centroid(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);
```

```
24 end
25
26 aux.node = node; aux.elem = elem;
27 aux.Centroid = Centroid;
28 aux.area = area;
29 aux.diameter = diameter;
```

2.4 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). 为了方便, 有时候需要 Dirichlet 边的信息, 为此我们用 `elemD` 存储 Dirichlet 边界的起点和终点编号.

2.4.1 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

```
1 [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

```
1 [node,elem] = squaremesh([0 1 0 1],0.5);
2 NT = size(elem,1);
3 % totalEdge
4 if iscell(elem)
5     shiftfun = @(verts) [verts(2:end),verts(1)];
6     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
```

注 2.3 边界边 bdEdge 在一维边集合 edge 中的自然序号为

```
bdIndex = find(s==1);
```

2.4.2 边界的设置

下面说明如何实现 eD, elemD 和 elemN. 以下图为例

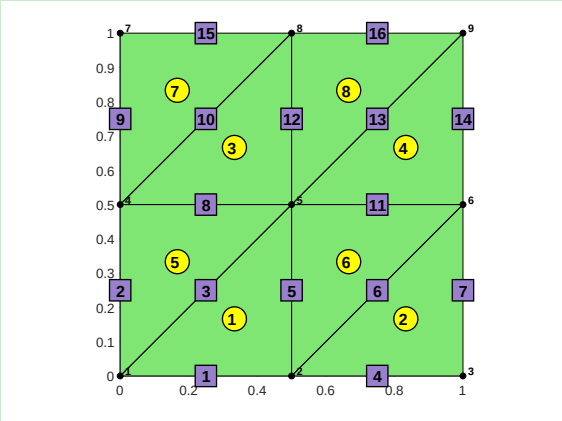


图 2.5. 边的自然序号

- 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 2.3. 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}))
```

```
30      for i = 1:nvar
31          bdNeumann = varargin{i};
32          id = eval(bdNeumann);
33          bdFlag(id) = false;
34      end
35  end
36  bdStruct.eD = unique(bdEdge(bdFlag,:));
37  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 2.4 以下两种写法等价

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

第三章　二维网格的生成

本文考虑 DistMesh (A Simple Mesh Generator in MATLAB) 工具箱, 如下网页给出了下载链接及简单的介绍

http://persson.berkeley.edu/distmesh/

3.1　Delaunay 三角剖分

3.1.1　Delaunay 三角剖分的定义

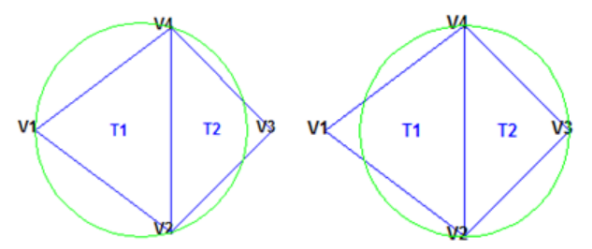
定义 3.1 (**三角剖分**)　假设 V 是二维实数域上的有限点集, 边 e 是由点集中的点作为端点构成的封闭线段, E 为 e 的集合. 称平面图 $\mathcal{T}=(V,E)$ 是点集 V 的三角剖分, 如果它满足条件:

1. 除了端点, 平面图中的边不包含点集中的任何点, 即边与点集无交点 (除端点);
2. 没有相交边, 即边和边没有交点 (除端点);
3. 所有的面都是三角面, 且所有三角面的合集是点集 V 的凸包.

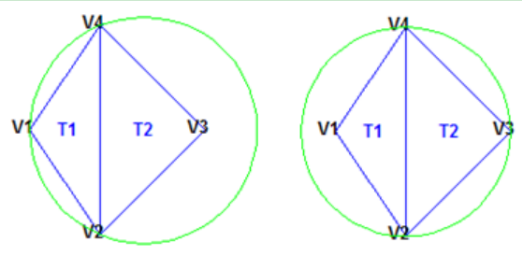
实际应用中, 最常用的是 Delaunay 三角剖分, 它是一种特殊的三角剖分. 为此, 我们先给出 Delaunay 边的定义.

定义 3.2 (Delaunay **边**)　任给边集合 E 中的一条边 e, 设其两个端点为 a,b. 称 e 为 Delaunay 边, 如果它满足空圆性质：存在一个圆经过 a,b 两点, 且该圆内不含点集 V 中任何其他的点.

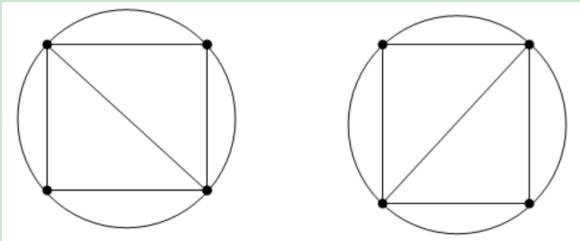
定义 3.3 (Delaunay **三角剖分**)　点集 V 的一个三角剖分 $\mathcal{T}$ 称为 Delaunay 三角剖分, 如果 $\mathcal{T}$ 的每条边都是 Delaunay 边.



如图, 设 $e=\overline{v_2v_4}$ 是三角剖分中的一条边, 以该边为公共边的三角形为 T_1 和 T_2. 可以证明, 若 T_1 的外接圆不包含 v_3, 且 T_2 的外接圆不包含 v_1, 则 e 就是 Delaunay 边. 下图中的 $e=\overline{v_2v_4}$ 就不是 Delaunay 边.



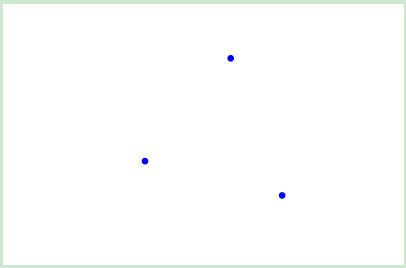
一般而言, Delaunay 三角剖分是不唯一的, 这是因为可能存在四点共圆, 如下图所示.



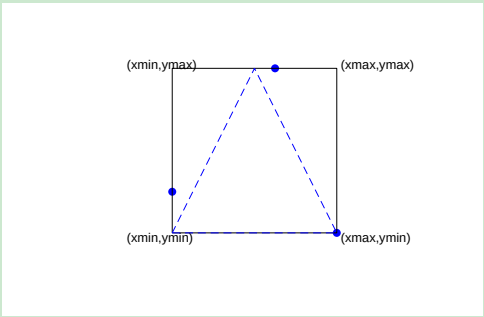
3.1.2 Bowyer 逐点插入法

Step 1: **构造超级三角形以包含所有点**

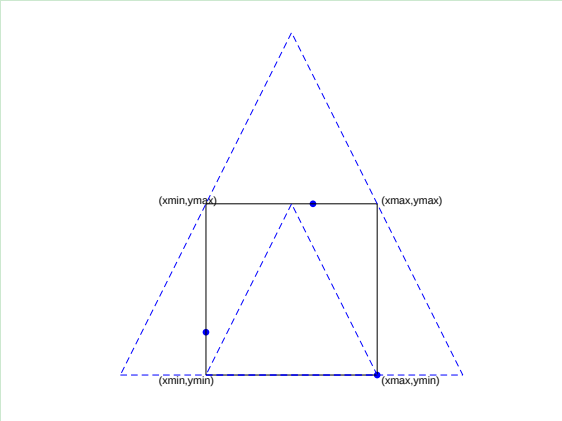
以如下的三个点进行说明.



先用坐标范围大小的矩形包含所有点, 并给定矩形内部的一个三角形, 如下图.



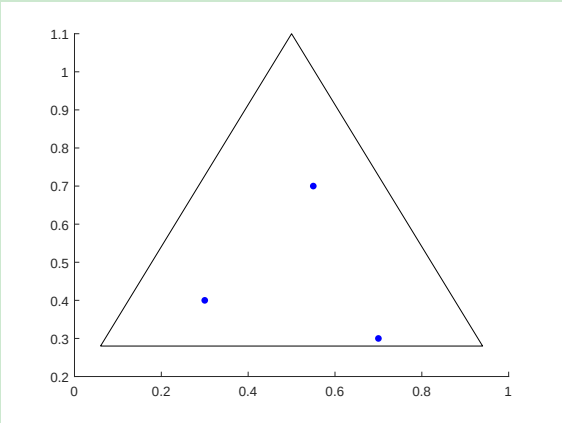
这里三角形的顶点为上侧边的中点. 做该三角形的相似三角形, 使其过矩形的两个上方顶点.



接着, 我们把三角形的底边往下移动一点并适当延长, 所得三角形一定包含所有散点 (都是内点).

```
% ----------------- vertex list --------------------
BdBox = [0 1 0 1]; h0 = 0.5;
[x,y] = meshgrid(BdBox(1):h0:BdBox(2),BdBox(3):h0*sqrt(3)/2:BdBox(4));
p = [x(:),y(:)];
np = size(p,1);

% ------------ determine the supertriangle ----------
xmin = min(p(:,1)); xmax = max(p(:,1));
ymin = min(p(:,2)); ymax = max(p(:,2));
hx = xmax-xmin; hy = ymax-ymin;

eps = 0.05*hy;
pts = [
    xmin+hx/2,        ymax+hy;
    xmin-hx/2-2*eps, ymin-eps;
    xmax+hx/2+2*eps, ymin-eps
    ];
tri = [1 2 3];
```

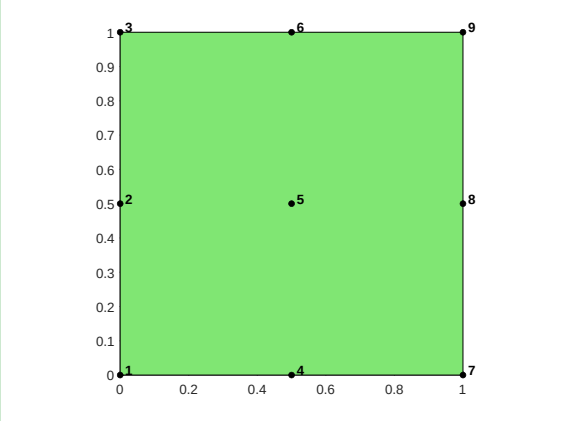
这里用 `pts` 和 `tri` 分别存储生成的三角形的顶点坐标和连通性.



注 3.1 超级三角形的顶点和预先给定的散点就是三角剖分的顶点, 因此最终的节点

以及初始的连通性信息为

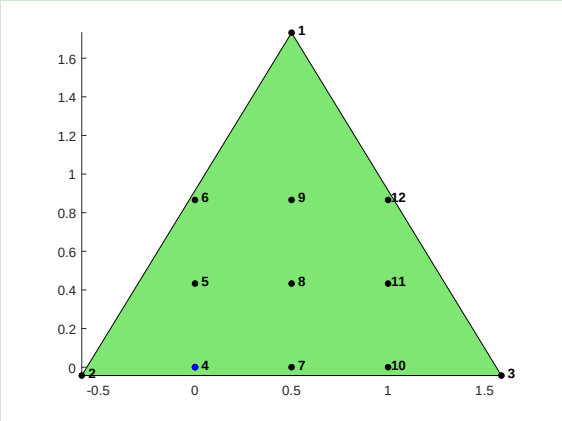
```
node = [pts; p];
elem = [1 2 3];
```



为了方便, 考虑上图中的 9 个散点. 注意到要加上超级三角形的三个顶点, 上面的顶点序号也要加上 3.

Step 2: **添加第 1 个点**

当前数据结构如下图



我们要找到第 1 个散点 (即图中的第 4 个点) 的影响三角形, 也就是外接圆包含该点的三角形. 函数 `inCircle(p0,p1,p2,p3)` 判断点 p_0 是否在以 p_1, p_2, p_3 为顶点的三角形的外接圆内. 这样, 可如下找到所有影响三角形.

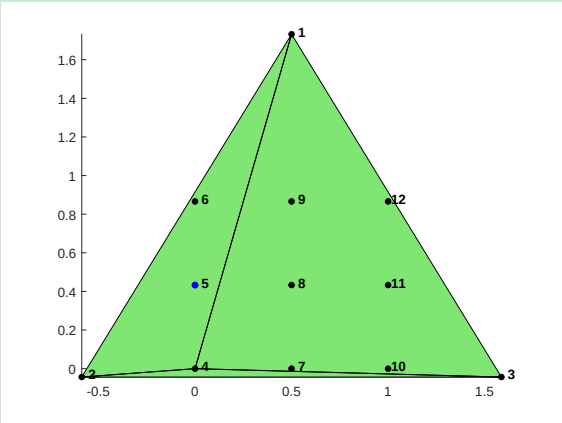
```
id_tri = false(NT,1); % extracting index
for iel = 1:NT
    index = elem(iel,:);
    z1 = node(elem(iel,1),:);
    z2 = node(elem(iel,2),:);
    z3 = node(elem(iel,3),:);
```

```
    if inCircle(pp,z1,z2,z3), id_tri(iel) = true; end
end
tri = elem(id_tri,:);
```

所有影响三角形构成一个多边形, 上面的情形只有一个影响三角形 (后面添加点时可看到多个三角形的情形). 连接散点和多边形的顶点得到当前的三角剖分.

```
tri = elem(id_tri,:);
edge_buffer = [tri(:,[2,3]); tri(:,[3,1]); tri(:,[1,2])];
N = 3+k; neb = size(edge_buffer,1);
tri = [edge_buffer, N*ones(neb,1)]; % new triangles
elem = [elem(¬id_tri,:); tri];
```

最后一步是把非影响三角形和新产生的三角形合并得到当前的三角剖分.



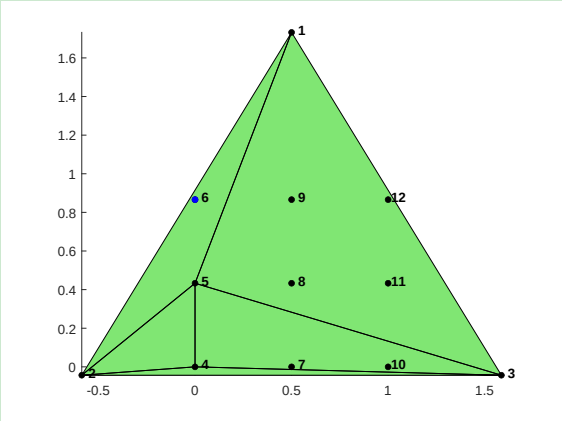
Step 3: **添加第 2 个点**

现在添加第 2 个散点, 即图中的点 5. 类似前面可找到影响三角形为 3-1-4 和 1-2-4, 即上图的左右两个三角形. 这两个三角形对应的多边形为 3-1-2-4, 显然只需要记录两个三角形的非公共边. 为此, 我们要删除 `edge_buffer` 中的所有重复边.

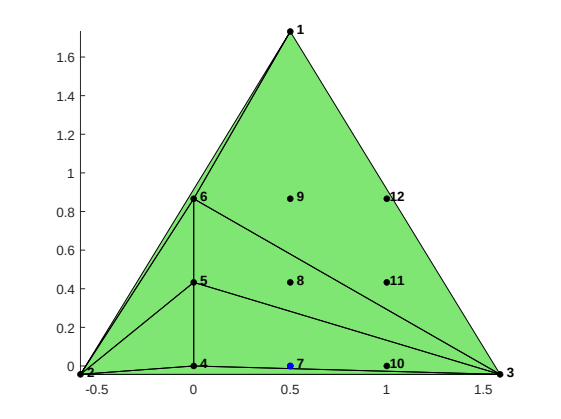
```
aa = full(sparse(edge_buffer(:,1),edge_buffer(:,2),1));
aa(aa+aa'≥2) = 0;
[i,j] = find(aa);
edge_poly = [i,j];
```

连接多边形的顶点与当前散点即得此时的三角剖分.

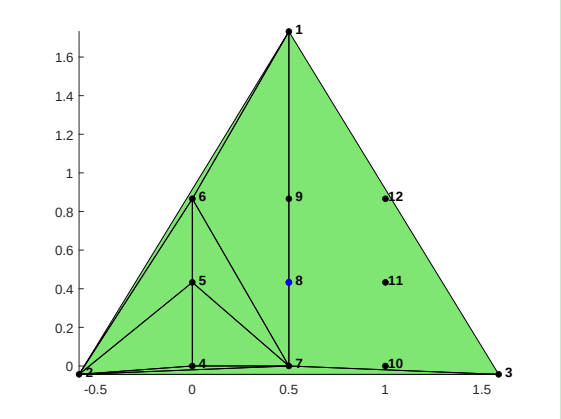
```
N = 3+k; n_poly = size(edge_poly,1);
tri_poly = [edge_poly, N*ones(n_poly,1)]; % new triangles
elem = [elem(¬id_tri,:); tri_poly];
```



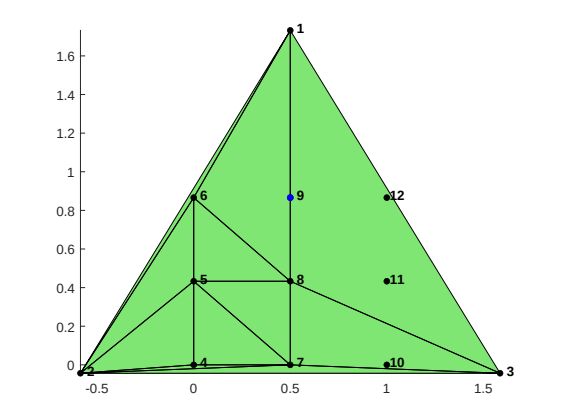
依次添加散点所得三角剖分图分别为



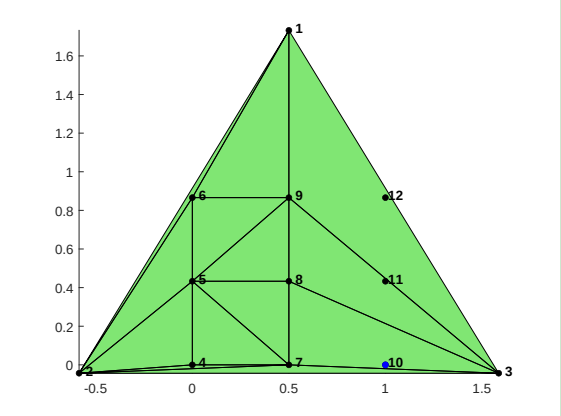
(a) $k = 3$



(b) $k = 4$



(c) $k = 5$



(d) $k = 6$

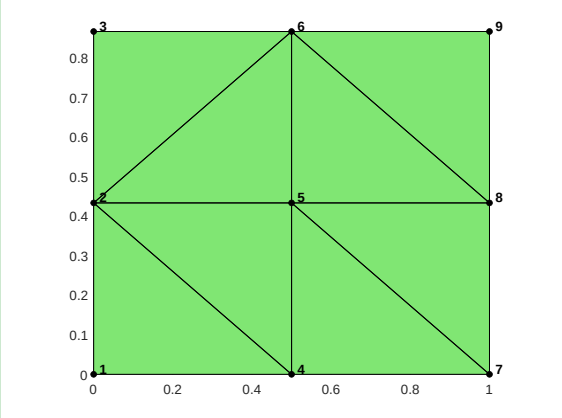
可以看到, $k = 6$ 时出现矩形对角线变方向的情形 (本程序不打算更改方向, 因为它一般实现的是无结构网格).

Step 4: **去掉超级三角形相关的三角形**

按照前面的过程遍历所有散点后, 我们只要去掉超级三角形顶点相关的三角形即得最终的三角剖分.

```
% remove any triangles that use the supertriangle vertices
NT = size(elem,1);
id = (elem(:,1)>3) & (elem(:,2)>3) & (elem(:,3)>3);
t = elem(id,:)-3;
```

剖分图如下



3.1.3 程序整理

最终的程序如下

```
function t = my_delaunay(p)

% determine the supertriangle
xmin = min(p(:,1)); xmax = max(p(:,1));
ymin = min(p(:,2)); ymax = max(p(:,2));
hx = xmax-xmin; hy = ymax-ymin;

eps = 0.05*hy;
pts = [
    xmin+hx/2,        ymax+hy;
    xmin-hx/2-2*eps, ymin-eps;
    xmax+hx/2+2*eps, ymin-eps
    ];
node = [pts; p];   elem = [1 2 3];

for k = 1:size(p,1)
    % find effected triangles assocaited with k-th point
    z1 = node(elem(:,1),:);
    z2 = node(elem(:,2),:);
    z3 = node(elem(:,3),:);
    id_tri = inCircle(p(k,:),z1,z2,z3);
    tri = elem(id_tri,:); % effected triangles
    edge_buffer = [tri(:,[2,3]); tri(:,[3,1]); tri(:,[1,2])];

    % obtain edges of the enclosing polygon
```

```
26      aa = full(sparse(edge_buffer(:,1),edge_buffer(:,2),1));
27      aa(aa+aa'≥2) = 0;
28      [i,j] = find(aa);
29      edge_poly = [i,j];
30
31      % update elem
32      N = 3+k; n_poly = size(edge_poly,1);
33      tri_poly = [edge_poly, N*ones(n_poly,1)]; % new triangles
34      elem = [elem(¬id_tri,:); tri_poly];
35  end
36
37  % remove any triangles that use the supertriangle vertices
38  id = (elem(:,1)>3) & (elem(:,2)>3) & (elem(:,3)>3);
39  t = elem(id,:)-3;
40  end % end of my_delaunay
```

注意, 我们修改了 inCircle.m 函数, 它可以判断一个点与多个三角形外接圆的位置关系.

3.2 DistMesh 的网格迭代思想

参考文献