MATLAB Programming for Virtual Element Methods

Part I

网格的生成与数据结构

1 网格的图示与标记

1.1 网格与解的图示

基本数据结构

采用陈龙编写的 iFEM 工具箱 [1] 中的数据结构, 用 `node` 表示节点坐标, `elem` 表示单元的连通性, 即单元顶点编号. 考虑下图中 L 形区域的一个简单剖分, iFEM 的网页说明链接如下:

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

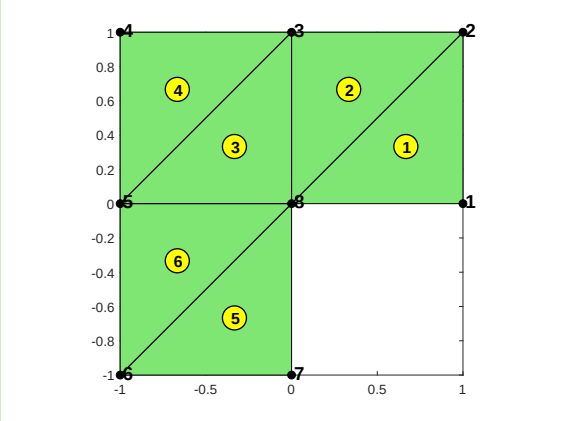


图 1. L 形区域的剖分

1. 矩阵 `node`: 节点坐标

   编程中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, 行的索引对应节点编号. 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. 矩阵 `elem`: 连通性

   `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.

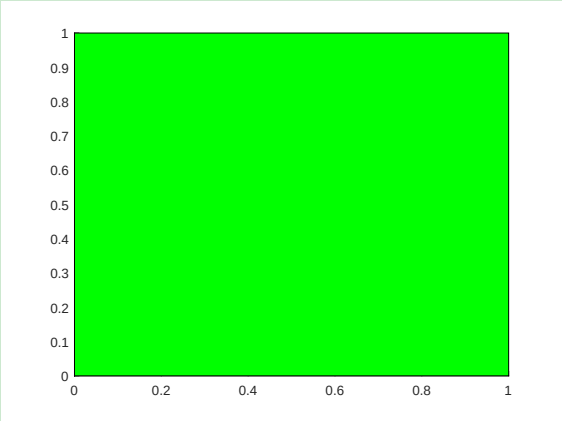
6x3 double

	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 i 列表示所有三角形的第 i 个点的编号, 其中 $i = 1, 2, 3$. 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 `P1`-元的局部整体对应.

补片函数 patch

MATLAB 中采用补片函数 `patch` 绘制多边形, 其内置的三角剖分画图函数 `trisurf` 也是如此. 以下考虑二维多角形剖分的图示, 命名为 showmesh.m. 一个简单的例子如下图



可如下编程

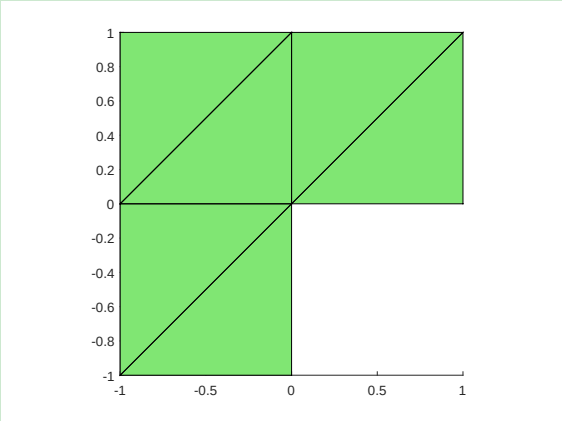
```
node = [0 0; 1 0; 1 1; 0 1];
elem = [1 2 3 4];
patch('Faces',elem,'Vertices',node,'FaceColor','g');
```

对多个相同类型的单元, showmesh.m 如下编写:

```
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;
```

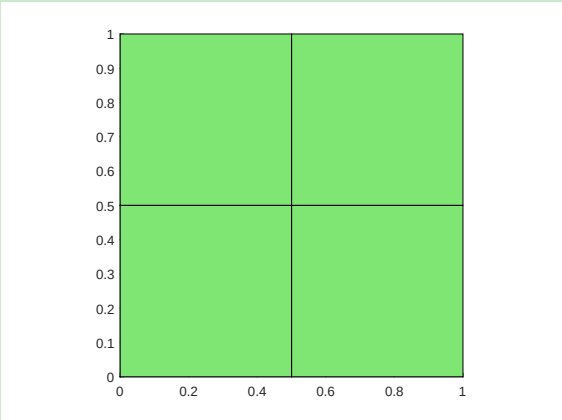
例 1.1 (三角剖分) 对前面的梯形区域, 如下调用 showmesh 函数

```
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)
```



操作 cell 数组的 cellfun 函数

对含有不同多角形单元的区域, 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), 需要将 `elem` 的每个 cell 用 `NaN` 填充成相同维数的向量, 该填充不会影响画图. 先介绍 MATLAB 中操作 cell 数组的函数 cellfun.m. 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'};
```

```
2 abbrev = cellfun(@(x) x(1:3), days, 'UniformOutput', false)
```

这里的 UniformOutput 也可简写为 un, 当然 false 也可写为 0. 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'}
```

showmesh 函数的建立

现在考虑图 2 所示的剖分

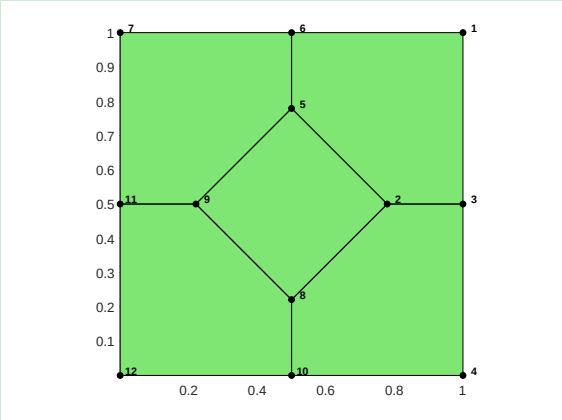


图 2. 多角形网格

相关的网格数据保存在 meshex1.mat 中. 程序如下

```
1 load('meshex1.mat'); % node, elem
2
3 max_n_vertices = max(cellfun(@length, elem));
4 % function to pad the vacancies ( 横向拼接 )
5 padding_func = @(vertex_ind) [vertex_ind,...
6       NaN(1,max_n_vertices-length(vertex_ind))];
7 tpad = cellfun(padding_func, elem, 'UniformOutput', false);
8 tpad = vertcat(tpad{:});
9 h = patch('Faces', tpad,'Vertices', node);
10 set(h,'facecolor',[0.5 0.9 0.45],'edgecolor','k');
11 axis equal; axis tight;
```

最终给出的 showmesh 函数如下

CODE 1. showmesh.m (2D 网格画图)

```
1 function showmesh(node,elem)
2
3 dim = size(node,2);
4
5 % ------------------- Triangulation --------------------------
6 % 2D
7 if ¬iscell(elem) && dim==2
8     h = patch('Faces', elem, 'Vertices', node);
9 end
10
```

```
% ------------------- Polygonal mesh -----------------------
if iscell(elem)
    if iscell(elem{1}), elem = vertcat(elem{:}); end
    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);
    face = vertcat(tpad{:});  % polygon
    h = patch('Faces', face, 'Vertices', node);
end

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

showsolution 函数的建立

showsolution 函数绘制解的网格图, 程序如下

```
function showsolution(node,elem,u)

dim = size(node,2);

% ------------------- Triangulation --------------------------
data = [node,u];
if ¬iscell(elem) && dim==2
    patch('Faces', elem,...
        'Vertices', data,...
        'FaceColor', 'interp',...
        'CData', u / max(abs(u)) );
end

% ------------------- Polygonal mesh -----------------------
if iscell(elem)
    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;
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);
```

简单说明一下.

- 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');
  ```

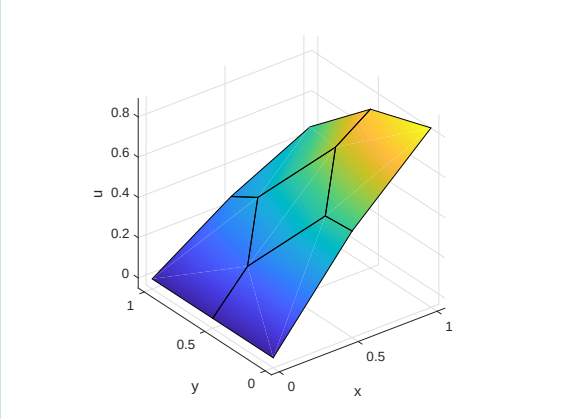
  此时显示的只是一种颜色, 但一般希望解有颜色的变化.
- 需要注意的是, 即便是三维数据, 若不加最后的

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

  给出的也是二维图 (投影, 即二维剖分图).

例 1.5 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);
```



1.2 网格的标记

节点标记

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 2. 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
```

当然也可简单改动以适用于三维情形.

单元标记

现在标记单元, 为此需给出单元重心, 从而标记序号. 重心使用 polycentroid.m 计算.

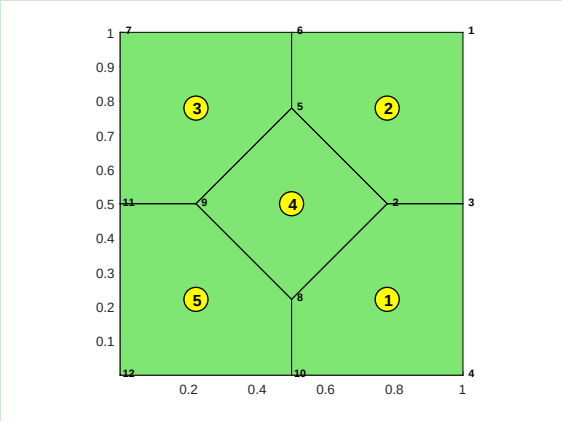


图 3. Polygonal mesh

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 3. findelem.m

```
function findelem(node,elem,varargin)

hold on

NT = size(elem,1);
if ~iscell(elem) % transform to cell
    elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
end

range = 1:NT;
if nargin==3, range = unique(varargin{1}); end
range = range(:);

center = zeros(length(range),2);
s = 1;
for iel = range(:)' % only valid for row vector
```

```
    index = elem{iel};
    V = node(index, :);
    center(s,:) = polycentroid(V);
    s = s+1;
end
plot(center(:,1),center(:,2),'o','LineWidth',1,'MarkerEdgeColor','k',...
    'MarkerFaceColor','y','MarkerSize',18);
text(center(:,1)-0.01,center(:,2),int2str(range),'FontSize',12,...
    'FontWeight','bold','Color','k');

hold off

end
```

注 1.1 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, 需要手动调整偏移量. 为了方便, 可直接用红色数字标记单元序号.

edge 的生成

为了标记边, 先要生成边的数据结构 `edge`. iFEM 对应的介绍见如下网页

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

该网页给出了一些辅助网格数据结构 (三角剖分), 其中的 `edge` 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), 并简单说明一下那里的思路.

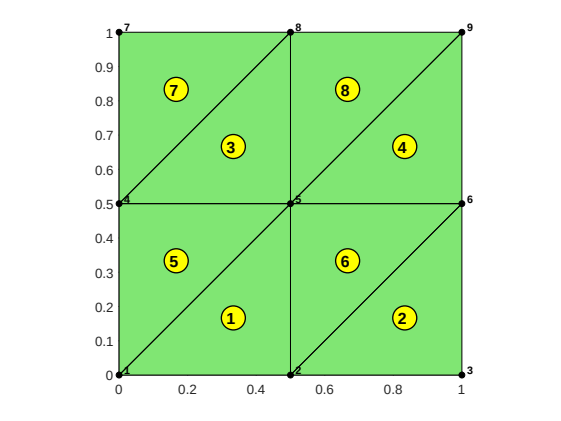
- 只要给出每条边两端的节点编号. 内部边在 `elem` 中会出现两次, 边界边只会出现一次, 为此可用 2 标记内部边, 1 标记边界边.
- 内部边在 `elem` 中会出现两次, 但它们是同一条边. 为了给定一致的标记, 规定每条边起点编号小于终点编号, 即 `edge(k,1)<edge(k,2)`.
- 对三角剖分, 通常规定三角形的第 i 条边对应第 i 个顶点. 例如, 设第 1 个三角形顶点顺序为 [1,4,5], 那么边的顺序应是 4-5, 5-1, 1-4. 在 MATLAB 中, 有如下对应

```
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)`, 只需将每行的两个元素排序. 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

- 设 e_{ij} $(i<j)$ 表示起点 i 终点 j 的边的重数, 对应的矩阵可用 sparse 函数如下生成

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

注意, 在 MATLAB 中, sparse 有一个特殊的性质 (summation property), 即当某个位置指标出现两次, 则相应的值会相加.

- 显然, sparse 命令产生的矩阵, 第一行对应起点 1 的边的重数, 第二行对应起点 2 的边的重数, 等等. 希望按下述方式排列边: 先排所有起点为 1 的边, 再排所有起点为 2 的边, 等等. 由于 find 是按列找非零元素, 因此要把矩阵进行转置, 即

```
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.

- edge 如下给出

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

注意因为前面进行了转置, 所以 `edge = [j,i]`. 边界边为

```
bdEdge = edge(s==1,:);
```

- 也可直接去重复实现, 即

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

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], 按顺序 8-10, 10-4, 4-3, 3-2, 2-8 给出单元的边的标记. 所有边的起点是 elem 中元素按列拉直给出的结果, 而终点是对 [10,4,3,2,8] 的拉直. 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.

边的标记

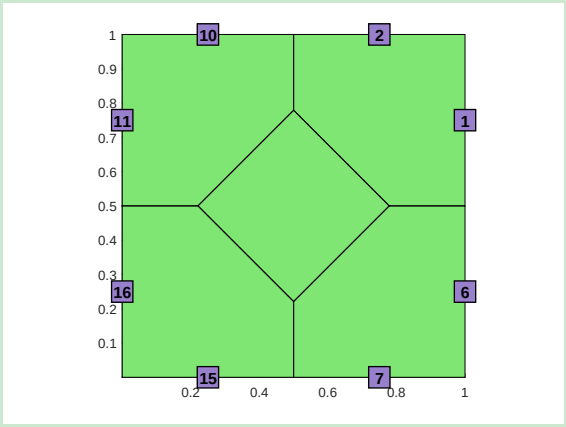
有了 edge, 边可用中点进行标记.

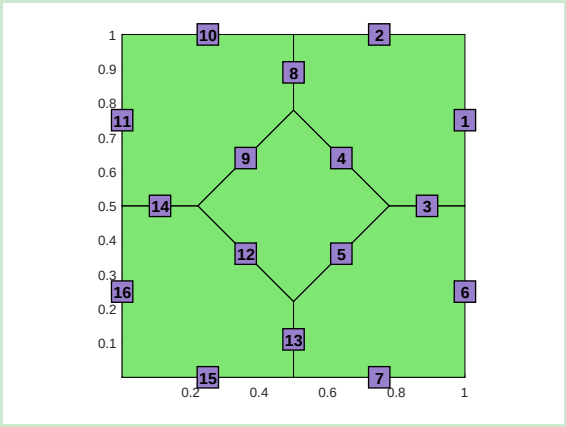
```
midEdge = (node(edge(range,1),:)+node(edge(range,2),:))/2;
plot(midEdge(:,1),midEdge(:,2),'s','LineWidth',1,'MarkerEdgeColor','k',...
        'MarkerFaceColor',[0.6 0.5 0.8],'MarkerSize',20);
    text(midEdge(:,1)-0.015,midEdge(:,2),int2str(range), ...
        'FontSize',12,'FontWeight','bold','Color','k');
```

上面的过程用函数 findedge.m 实现, 如

```
load('meshex1.mat');
figure,
showmesh(node,elem);
bdInd = 1;
findedge(node,elem,bdInd);% only show boundary edges
figure,
showmesh(node,elem);
findedge(node,elem); % show all edges
```

注 1.2 为了统一虚拟元的编号规则, 三角形的第一条边就是第一个顶点连接的“右侧边”.





2 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. 本文参考 iFEM 给出一些辅助数据结构, 相关说明见如下网页

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.

2.1 辅助数据结构

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
`node2elem`	顶点周围的单元

注 2.1 某些问题可能需要其他数据结构, 需要时再补充, 这里则不再说明.

elem2edge 的生成

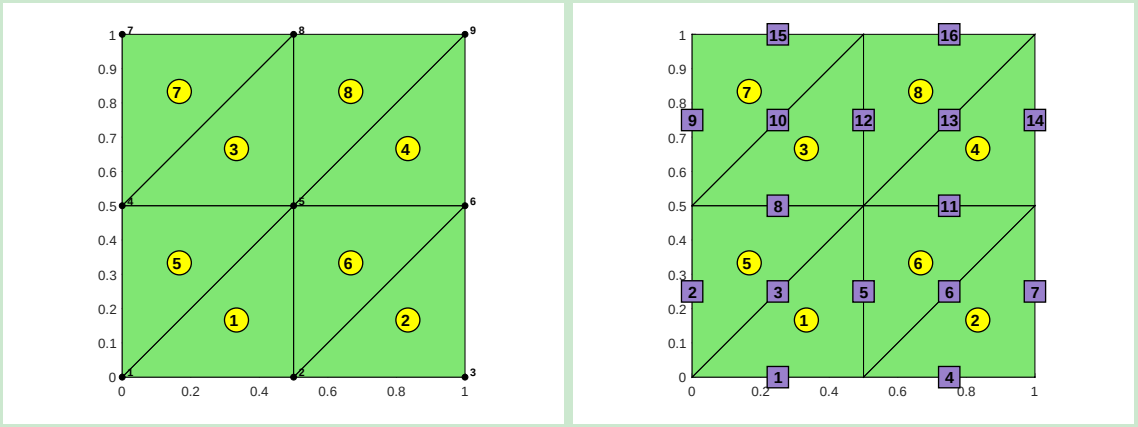
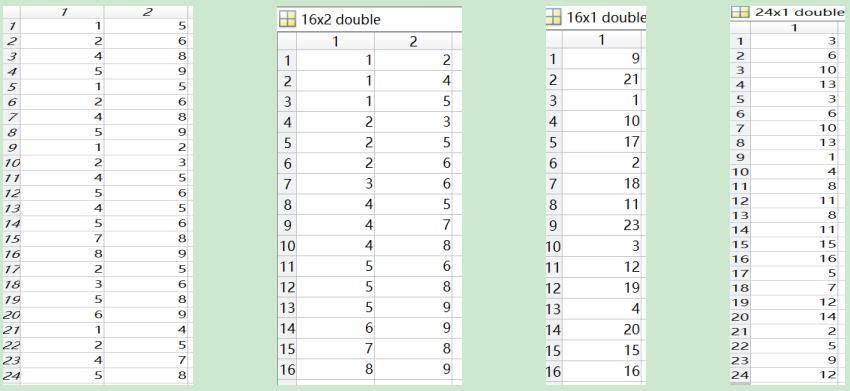


图 4. 三角剖分边的自然序号

用图 4 中给出的三角剖分说明一下 iFEM 中的思路 (有所改动).

- 根据前面的说明, 可给出含重复边的数组 `totalEdge` 见图 6(a).



(a) totalEdge (b) edge (c) i1 (d) totalJ

图 5. elem2edge 的说明

- 下面使用 unique 函数去除重复.
  - 在 MATLAB 中, `[C, iCA, iAC] = unique(A,'rows')` 按矩阵的行进行比较得到不重复的 `C`.

  - iCA 与 C 行数相同, 记录 C 在原矩阵 A 中的位置 (重复的按第一次出现记录, 但许早期版本可能按第二次记录).
  - iAC 与 A 行数相同, 记录 A 在矩阵 C 中的位置.

- 去除重复的行, 即得边的数据结构 edge (重复边一致化才能使用)

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

  - 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 6 (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 (即索引). 比如, 1-5 在 edge 中是第 3 个, 则 totalEdge 的所有 1-5 边的序号为 3.

  显然有

```
edge = totalEdge(i1,:); totalEdge = edge(totalJ,:);
```

- 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, 要用元胞数组存储.

- 设 elem 各单元顶点个数为分别为 $N_1, \cdots, N_t$ (t 为单元个数), 则向量 totalJ 的前 N_1 个元素为单元 1 的边序号, 接着的 N_2 个元素为单元 2 的边序号, 等等.

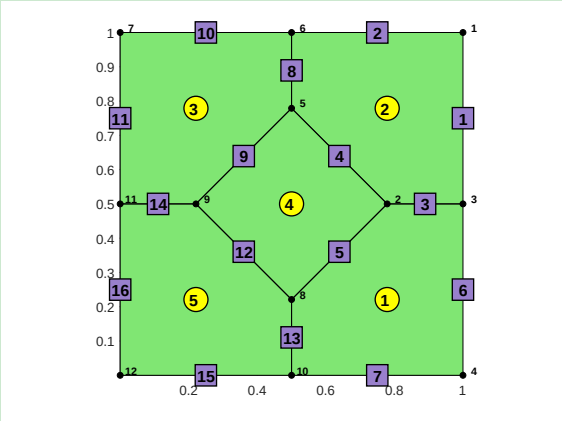
- 将 totalJ 变成 t 个元胞, 其中第 1 个元胞存储前 N_1 个元素, 第 2 个元胞存储接着的 N_2 个元素, 等等, 所得即为 elem2edge.

- 在 MATLAB 中可用 `mat2cell` 将一个矩阵变为元胞, 本质是对矩阵进行块的分割. 我们希望 `elem2edge` 的每个元胞是行向量, 为此在分割时, 先将 `totalJ` 变为行向量. 接着, 它将被分割为如下 t 个块:

$$J_1, J_2, \cdots, J_t, \quad J_i \in \mathbb{R}^{1\times N_i}.$$

MATLAB 中如下实现

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



对 meshex1.mat, 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

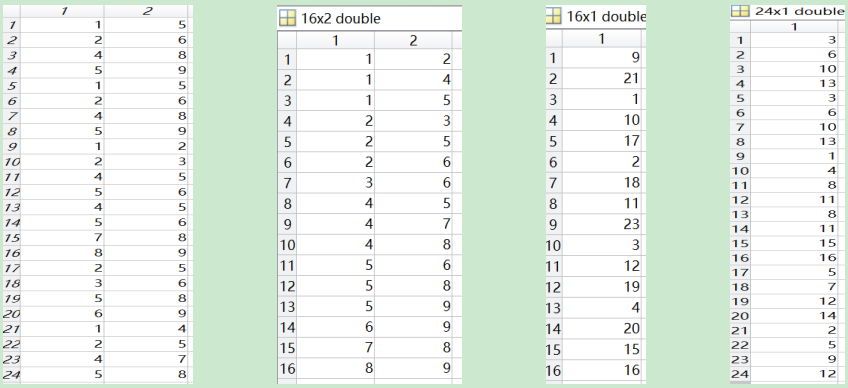
8x2 double

	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

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. 为此定义矩阵 `edge2elem`, 其维数为 `NE*2`, 而 `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

(d) 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

(e) 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

(f) 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

(g) totalJ

图 6. elem2edge 图示

- `totalEdge` 按单元排列每条边, 内部边因在相邻两个单元均出现, 会重复一次. 称第一次出现的边位于左单元, 第二次出现的边位于右单元.
- 第一次出现的行号如下获得

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

- 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` 对应右单元边, 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;
```

- 其实可以直接用 `i1` 和 `totalJ` 获得 `i2`.

```
i2(totalJ)= 1:length(totalJ); i2 = i2(:);
```

  这是简单的覆盖技巧.

  - 例如考察图 6 中 `totalJ` 的第 1 行和第 5 行, 它们都对应 `edge` 中的第 3 条边.
  - 上面获得 `i2(3)` 的过程为

```
i2(3)= 1; i2(3)= 5;
```

  - 即根据行索引与 3 的对应, 把第一次出现的索引 1 用第二次出现的索引 5 覆盖.

- 注意到 `totalEdge` 的行可分为 t 个块, 这些块的序号就是边对应的单元号. 为此, 可生成一个向量 `totalJElem`, 它的前 N_1 个元素均为 1, 接下来的 N_2 个元素均为 1, 依此类推.

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

- 这样, 设第一次出现的行号为 i1, totalJElem(i1) 给出的就是左单元编号.

综上, 如下实现 edge2elem

```
% ------- edge2elem --------
Num = num2cell((1:NT)');     Len = num2cell(elemLen);
totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
totalJelem = vertcat(totalJelem{:});
[¬, 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

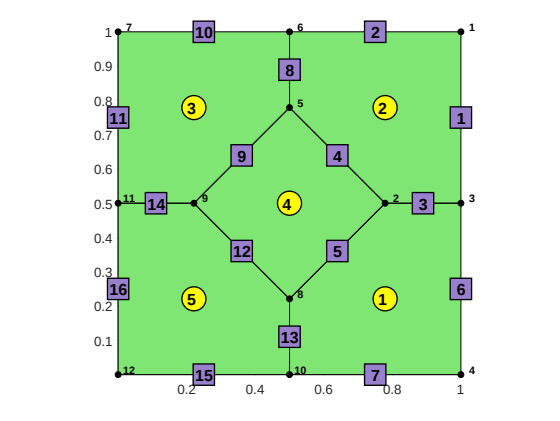


图 7. edge2elem

例如, 序号 12 的边连接的两个单元编号为 4 和 5.

注 2.2 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.

neighbor 的生成

neighbor 与 elem2edge 类似, 只不过记录的是每个单元各条边连接的单元序号 (对边界单元, 规定其边界边连接的单元为自身).

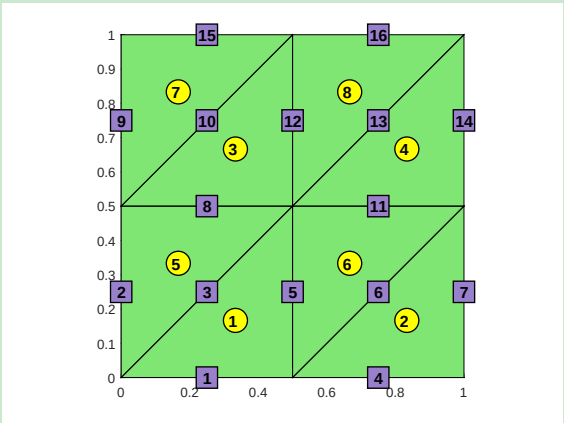


图 8. 三角剖分边的自然序号

- 以图 8 的单元 3 为例, 它的边的局部顺序为 10-8-12. 这些边对应的左右单元如下 (左右顺序其实无所谓, 但前面已经保证左侧序号小于右侧序号)

$$\begin{bmatrix} \text{左}: & 3 & 3 & 3 \\ \text{右}: & 7 & 5 & 8 \end{bmatrix}$$

- 将该矩阵按列拉直, 并去除当前单元的序号 3, 所得向量 $[7,5,8]$ 即为相邻单元的序号. 但对边界单元, 这种处理会丢失边界边. 例如, 单元 1 边的局部顺序为 3-1-5, 每条边对应的左右单元为

$$\begin{bmatrix} \text{左}: & 1 & 1 & 1 \\ \text{右}: & 5 & 1 & 6 \end{bmatrix}$$

  去除重复后为 $[5,6]$, 边界边没有保留序号.

我们换一种处理方法, 先给出程序.

```
% --------- neighbor ---------
neighbor = cell(NT,1);
for iel = 1:NT
    index = elem2edge{iel};
    ia = edge2elem(index,1); ib = edge2elem(index,2);
    ia(ia==iel) = ib(ia==iel);
    neighbor{iel} = ia;
end
```

- `ia` 和 `ib` 分别是左侧和右侧单元的序号.
- 左侧单元序号 `ia` 中有一部分是当前单元序号 `iel`, 它们应该替换成对应的右侧单元序号 `ib(ia==iel)`.
- 上面的替换保留了矩阵

$$\begin{bmatrix} \text{左}: & 1 & 1 & 1 \\ \text{右}: & 5 & 1 & 6 \end{bmatrix}$$

  中第 2 列的左侧单元序号 1, 因而保留了边界边对应的单元.

node2elem 的生成

我们将给出一个稀疏矩阵 `t2v`, 其尺寸为 `NT*N`, 它的行对应单元, 列对应顶点. 对固定单元, `t2v` 相应行的非零元素所在的列索引恰好为单元的顶点序号, 这样, 按列查找非零元素即可获得顶点周围的单元.

`t2v` 元素的行索引在前面生成 `edge2elem` 已经给出, 即 `totalJelem`, 它的前面若干个元素为 1, 长度为第 1 个单元的顶点数, 接着若干个元素为 2, 长度为第 2 个单元的顶点数, 依此类推.

```
% --------- node2elem ---------
ii = totalJelem; jj = v0; ss = ones(length(ii),1);
t2v = sparse(ii,jj,ss,NT,N);
node2elem = cell(N,1);
for i = 1:N
    node2elem{i} = find(t2v(:,i))';
end
```

上面的循环可用 cellfun 函数实现.

```
node2elem = cellfun(@(x) find(x), num2cell(t2v,1) ,'UniformOutput', false);
node2elem = node2elem';
```

这里的 `num2cell(A,1)` 是将矩阵按列转化为元胞数组.

2.2 网格相关的几何量

几何量包括

表 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

```
% ------ centroid, 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
```

注: MATLAB 自带函数 polyarea 用以计算多边形的面积.

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 4. auxstructure.m

```
function aux = auxstructure(node,elem)
%auxstructure gets auxiliary mesh data structure

NT = size(elem,1); N = size(node,1);
if ¬iscell(elem) % transform to cell
    elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
end

% 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',1,elemLen)';

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

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

% --------- neighbor ---------
neighbor = cell(NT,1);
for iel = 1:NT
    index = elem2edge{iel};
    ia = edge2elem(index,1); ib = edge2elem(index,2);
    ia(ia==iel) = ib(ia==iel);
    neighbor{iel} = ia;
end

% --------- node2elem ---------
ii = totalJelem; jj = v0; ss = ones(length(ii),1);
t2v = sparse(ii,jj,ss,NT,N);
node2elem = cellfun(@(x) find(x), num2cell(t2v,1) ,'UniformOutput', false);
node2elem = node2elem';
```

```
48
49 aux.node = node; aux.elem = elem;
50 aux.elem2edge = elem2edge;
51 aux.edge = edge; aux.bdEdge = bdEdge;
52 aux.edge2elem = edge2elem;
53 aux.neighbor = neighbor; aux.node2elem = node2elem;
```

auxgeometry.m 函数如下

CODE 5. auxgeometry.m

```
1  function aux = auxgeometry(node,elem)
2
3  % ------ centroid, area, diameter -------
4  NT = size(elem,1);
5  centroid = 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     centroid(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.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 `bdNodeIdx` 存储 Dirichlet 节点的编号, 后者用 `bdEdgeN` 存储 Neumann 边界的起点和终点编号 (即一维问题的连通性信息).

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 $\partial_{\boldsymbol{n}} u$, 这就需要我们恢复边界边的定向以确定外法向量 (边的旋转获得).

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

- 一是累计重复的次数 (1 是边界, 2 是内部)

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

- 二是直接去掉重复的边

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

这里, `i1` 记录的是 `edge` 在重复边 `totalEdge` 中的位置.

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

```
%% 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
allEdge = [v0,v1];
totalEdge = sort(allEdge,2);

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

2.4.2 边界的设置

下面说明如何实现 `bdNodeIdx` 和 `bdEdgeN`. 以下图为例

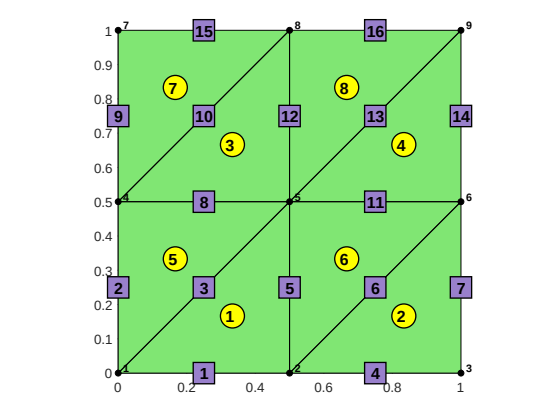


图 9. 边的自然序号

- 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';
```

```
midbdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
x = midbdEdge(:,1); y = midbdEdge(:,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 6. setboundary.m

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

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

%% 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
allEdge = [v0,v1];
totalEdge = sort(allEdge,2);

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

%% set up boundary
nE = size(bdEdge,1);
% initial as Dirichlet (true for Dirichlet, false for Neumann)
Idx = true(nE,1);
midbdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
x = midbdEdge(:,1); y = midbdEdge(:,2); %#ok<NASGU>
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);
```

```
        Idx(id) = false;
    end
end
bdStruct.bdEdge = bdEdge; % all boundary edges
bdStruct.bdEdgeD = bdEdge(Idx,:); % Dirichlet boundary edges
bdStruct.bdEdgeN = bdEdge(¬Idx,:); % Neumann boundary edges
bdStruct.bdNodeIdx = unique(bdEdge(Idx,:)); % index of Dirichlet boundary nodes
bdEdgeIdx = find(s==1);      % index of all boundary edges
bdStruct.bdEdgeIdx = bdEdgeIdx;
bdStruct.bdEdgeIdxD = bdEdgeIdx(Idx); % index of Dirichelt boundary edges
bdStruct.bdEdgeIdxN = bdEdgeIdx(¬Idx); % index of Neumann boundary edges
```

这里我们还增加了 Dirichlet 边, 以方便后面使用. 相关信息保存在结构体 `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 2.3 以下两种写法等价

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