

## 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>

本节针对一般的多三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

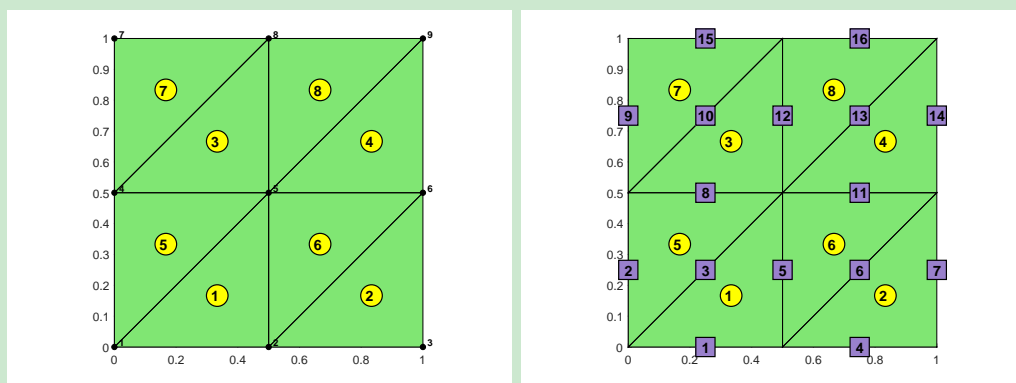


图 4. 三角剖分边的自然序号

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).

	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

	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

	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

	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 的说明

- 下面使用 unique 函数去除重复.

- 在 MATLAB 中,  $[C, iCA, iAC] = \text{unique}(A, 'rows')$  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.

---

```
1 % ----- elem2edge (triangulation) -----
2 [node,elem] = squaremesh([0 1 0 1],0.5);
3 totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
4 [edge, i1, totalJ] = unique(totalEdge, 'rows');
5 NT = size(elem,1);
6 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

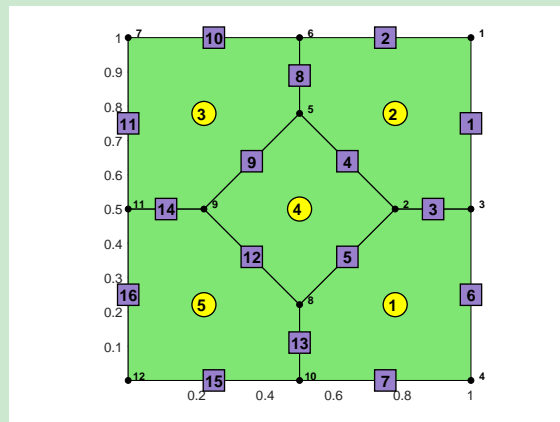
上面的思路适用于多边形剖分, 只不过此时因边数不同, 要用元胞数组存储.

- 设 `elem` 各单元顶点个数分别为  $N_1, \dots, N_t$  ( $t$  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:

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

MATLAB 中如下实现

```
1 % ----- elem2edge: elementwise edges -----
2 [~, i1, totalJ] = unique(totalEdge, 'rows');
3 elemLen = cellfun('length', elem); % length of each elem
4 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  $NE \times 2$ , 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  $3 \times NT+1$ ). 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.

```
1 Num = num2cell((1:NT)');
2 Len = num2cell(cellLen);
3 totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
4 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  $edge2elem$

```
1 % ----- edge2elem -----
2 Num = num2cell((1:NT)'); Len = num2cell(elemLen);
3 totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
4 totalJelem = vertcat(totalJelem{:});
5 [~, i2] = unique(totalJ(end:-1:1), 'rows');
6 i2 = length(totalEdge)+1-i2;
7 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

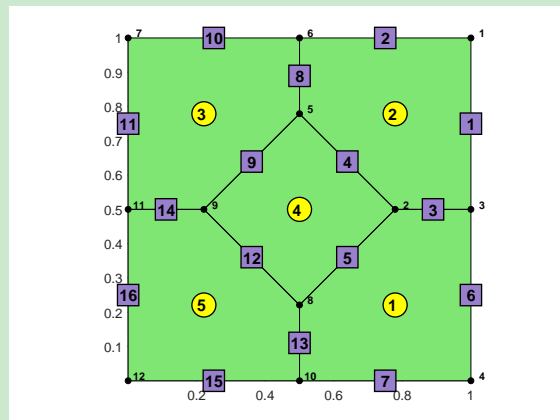
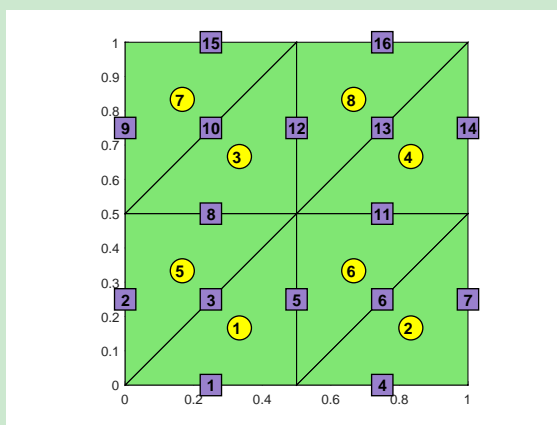


图 7.  $edge2elem$

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` 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).



- 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.

```

1 % ----- neighbor -----
2 neighbor = cell(NT,1);
3 for iel = 1:NT
4     index = elem2edge{iel};
5     ia = edge2elem(index,1); ib = edge2elem(index,2);
6     ia(ia==iel) = ib(ia==iel);
7     neighbor{iel} = ia;
8 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  $t_{2v}$ , 其尺寸为  $N_T \times N$ , 它的行对应单元, 列对应顶点. 对固定单元,  $t_{2v}$  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.

$t_{2v}$  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.

---

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

---

上面的循环可用 `cellfun` 函数实现.

---

```
1 node2elem = cellfun(@(x) find(x), num2cell(t2v,1), 'UniformOutput', false);
2 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

---

```

1 % ----- centroid, area, diameter -----
2 centroid = zeros(NT,2); area = zeros(NT,1); diameter = zeros(NT,1);
3 s = 1;
4 for iel = 1:NT
5     if iscell(elem)
6         index = elem{iel};
7     else
8         index = elem(iel,:);
9     end
10    verts = node(index, :); verts1 = verts([2:end,1],:);
11    area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
12    ar = 0.5*abs(sum(area_components));
13    area(iel) = ar;
14    centroid(s,:) = sum((verts+verts1).*repmat(area_components,1,2))/(6*ar);
15    diameter(s) = max(pdist(verts));
16    s = s+1;
17 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

```

1 function aux = auxstructure(node,elem)
2 %auxstructure gets auxiliary mesh data structure
3
4 NT = size(elem,1); N = size(node,1);
5 if ~iscell(elem) % transform to cell
6     elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
7 end
8
9 % totalEdge
10 shiftfun = @(verts) [verts(2:end),verts(1)]; % or shiftfun = @(verts) ...
    circshift(verts,-1);
11 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
12 v0 = horzcat(elem{:})'; % the starting points of edges
13 v1 = horzcat(T1{:})'; % the ending points of edges
14 totalEdge = sort([v0,v1],2);
15
16 % ----- elem2edge: elementwise edges -----
17 [i, i1, totalJ] = unique(totalEdge,'rows');
18 elemLen = cellfun('length',elem); % length of each elem
19 elem2edge = mat2cell(totalJ',1,elemLen)';
20
21 % ----- edge, bdEdge -----

```



```

22 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
23 edge = [j,i];
24 bdEdge = edge(s==1,:);
25
26 % ----- edge2elem -----
27 Num = num2cell((1:NT)'); Len = num2cell(elemLen);
28 totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
29 totalJelem = vertcat(totalJelem{:});
30 [~, i2] = unique(totalJ(end:-1:1),'rows');
31 i2 = length(totalEdge)+1-i2;
32 edge2elem = totalJelem([i1,i2]);
33
34 % ----- neighbor -----
35 neighbor = cell(NT,1);
36 for iel = 1:NT
37     index = elem2edge{iel};
38     ia = edge2elem(index,1); ib = edge2elem(index,2);
39     ia(ia==iel) = ib(ia==iel);
40     neighbor{iel} = ia;
41 end
42
43 % ----- node2elem -----
44 ii = totalJelem; jj = v0; ss = ones(length(ii),1);
45 t2v = sparse(ii,jj,ss,NT,N);
46 node2elem = cellfun(@(x) find(x), num2cell(t2v,1), 'UniformOutput', false);
47 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  $\text{edge}(k,1) < \text{edge}(k,2)$ . 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 `bdEdge` 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 %% totalEdge
2 shiftfun = @(verts) [verts(2:end),verts(1)]; % or shiftfun = @(verts) ...
    circshift(verts,-1);
3 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
4 v0 = horzcat(elem{:})'; % the starting points of edges
5 v1 = horzcat(T1{:})'; % the ending points of edges
6 allEdge = [v0,v1];
7 totalEdge = sort(allEdge,2);
8
9 %% counterclockwise bdEdge
10 [~,~,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
11 [~, i1, ~] = unique(totalEdge, 'rows');
12 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

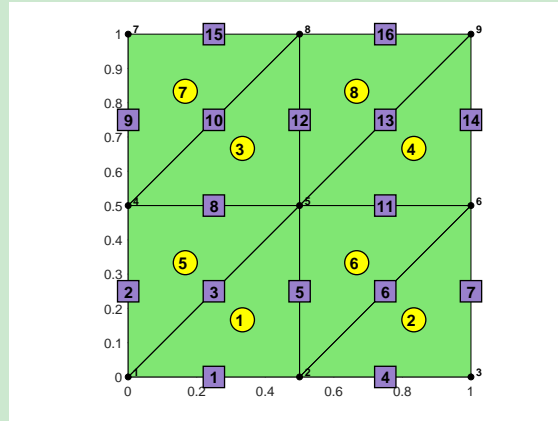


图 9. 边的自然序号

- 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` 对应的边的中点在该线段上. 如下

```
1 bdFun = 'x==1';
2 midbdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
3 x = midbdEdge(:,1); y = midbdEdge(:,2);
4 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, 可以给出函数 `setboundary.m`

### CODE 6. `setboundary.m`

```
1 function bdStruct= setboundary(node,elem,varargin)
2 % varargin: string for Neumann boundary
3
4 NT = size(elem,1);
5 if ~iscell(elem) % transform to cell
6     elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
7 end
8
```

```

9 %% totalEdge
10 shiftfun = @(verts) [verts(2:end),verts(1)]; % or shiftfun = @(verts) ...
    circshift(verts,-1);
11 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
12 v0 = horzcat(elem{:})'; % the starting points of edges
13 v1 = horzcat(T1{:})'; % the ending points of edges
14 allEdge = [v0,v1];
15 totalEdge = sort(allEdge,2);
16
17 %% counterclockwise bdEdge
18 [¬,¬,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
19 [¬, i1, ¬] = unique(totalEdge,'rows');
20 bdEdge = allEdge(i1(s==1),:);
21
22 %% set up boundary
23 nE = size(bdEdge,1);
24 % initial as Dirichlet (true for Dirichlet, false for Neumann)
25 Idx = true(nE,1);
26 midbdEdge = (node(bdEdge(:,1),:) + node(bdEdge(:,2),:))/2;
27 x = midbdEdge(:,1); y = midbdEdge(:,2); %#ok<NASGU>
28 nvar = length(varargin); % 1 * size(varargin,2)
29 % note that length(varargin) = 1 for bdNeumann = [] or ''
30 if (nargin==2) || (¬isempty(varargin{1}))
31     for i = 1:nvar
32         bdNeumann = varargin{i};
33         id = eval(bdNeumann);
34         Idx(id) = false;
35     end
36 end
37 bdStruct.bdEdge = bdEdge; % all boundary edges
38 bdStruct.bdEdgeD = bdEdge(Idx,:); % Dirichlet boundary edges
39 bdStruct.bdEdgeN = bdEdge(¬Idx,:); % Neumann boundary edges
40 bdStruct.bdNodeIdx = unique(bdEdge(Idx,:)); % index of Dirichlet boundary nodes
41 bdEdgeIdx = find(s==1); % index of all boundary edges
42 bdStruct.bdEdgeIdx = bdEdgeIdx;
43 bdStruct.bdEdgeIdxD = bdEdgeIdx(Idx); % index of Dirichelt boundary edges
44 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');

```