

MATLAB Programming for Virtual Element Methods

| | |
|--|----|
| I 网格的生成与数据结构 | 7 |
| 1 二维网格的预处理 | 8 |
| 1.1 网格的数据结构及图示 | 8 |
| 1.1.1 基本数据结构 | 8 |
| 1.1.2 补片函数 patch | 9 |
| 1.1.3 操作 cell 数组的 cellfun 函数 | 10 |
| 1.1.4 showmesh 函数的建立 | 11 |
| 1.1.5 showsolution 函数的建立 | 12 |
| 1.2 Generation of the polygonal meshes | 14 |
| 1.3 网格的标记 | 15 |
| 1.3.1 节点标记 | 15 |
| 1.3.2 单元标记 | 15 |
| 1.3.3 边的标记 | 16 |
| 1.4 VEM 空间的辅助数据结构与几何量 | 20 |
| 1.4.1 elem2edge 的生成 | 21 |
| 1.4.2 edge2elem 的生成 | 24 |
| 1.4.3 neighbor 的生成 | 25 |
| 1.4.4 node2elem 的生成 | 26 |
| 1.5 网格相关的几何量 | 26 |
| 1.6 auxstructure 与 auxgeometry 函数 | 27 |
| 1.7 边界设置 | 29 |
| 1.7.1 边界边的定向 | 29 |
| 1.7.2 边界的设置 | 29 |
| 2 MPT: 多角形与多面体区域的网格剖分 | 32 |
| 2.1 Voronoi 图 | 32 |
| 2.1.1 Voronoi 图的定义 | 32 |
| 2.1.2 MPT 中多面体的表示 | 32 |
| 2.1.3 带边界的 Voronoi 图的生成 | 34 |
| 2.2 Lloyd 迭代 | 35 |

| | | |
|-----------|---------------------------------|-----------|
| 2.2.1 | CVTs 及其变分描述 | 35 |
| 2.2.2 | 计算 CVTs 的 Lloyd 算法 | 36 |
| 2.2.3 | Lloyd 迭代的实现 | 36 |
| 2.3 | 2D 基本数据结构的获取 | 37 |
| 2.3.1 | 恢复单元节点的逆序 | 37 |
| 2.3.2 | 获得 node 和 elem | 38 |
| 2.3.3 | 去除短边 | 39 |
| 2.3.4 | 程序整理 | 42 |
| 2.4 | 3D 基本数据结构的获取 | 44 |
| 2.4.1 | 单元面的提取 | 44 |
| 2.4.2 | 节点的整体编号 | 47 |
| 2.4.3 | 短边的去除 | 50 |
| 2.4.4 | 程序整理 | 52 |
| 3 | PolyMesher: 二维多角形剖分的生成 | 53 |
| 3.1 | Voronoi 图 | 53 |
| 3.2 | 反射集的计算 | 56 |
| 3.2.1 | 符号距离函数 | 56 |
| 3.2.2 | 初始种子的生成 | 56 |
| 3.2.3 | 光滑边界凸区域反射集的计算 | 58 |
| 3.2.4 | 分段光滑边界凸区域反射集的计算 | 59 |
| 3.2.5 | 非凸区域反射集的计算 | 61 |
| 3.2.6 | 反射集计算程序 | 61 |
| 3.3 | Lloyd 迭代 | 62 |
| 3.4 | 获取网格剖分的基本数据集 | 62 |
| 3.5 | 程序整理 | 65 |
| 4 | PolyMesher3D: 三维多面体剖分的生成 | 67 |
| 4.1 | 网格的生成 | 67 |
| 4.1.1 | 三维 Voronoi 图 | 67 |
| 4.1.2 | Lloyd 迭代 | 69 |
| 4.1.3 | 去除短边 | 70 |
| 4.2 | 基本数据结构的获取 | 73 |
| 4.2.1 | 获得共面三角剖分 | 73 |
| 4.2.2 | 提取基本数据结构 | 74 |
| 4.3 | 程序整理 | 75 |
| II | 经典问题的虚拟元方法 | 78 |

| | |
|------------------------------------|------------|
| 5 Poisson 方程的协调 VEM | 79 |
| 5.1 VEM 简介 | 79 |
| 5.2 椭圆投影的计算 | 80 |
| 5.2.1 过渡矩阵 D | 80 |
| 5.2.2 椭圆投影在基下的矩阵 | 81 |
| 5.2.3 投影限制条件 | 84 |
| 5.3 L^2 投影与强化技巧 | 85 |
| 5.3.1 提升空间中的椭圆投影 | 85 |
| 5.3.2 L^2 投影的计算 | 86 |
| 5.4 Poisson 方程一阶 VEM 程序 | 88 |
| 5.4.1 问题描述 | 88 |
| 5.4.2 单元刚度矩阵的计算 | 89 |
| 5.4.3 单元载荷向量的计算 | 90 |
| 5.4.4 刚度矩阵与载荷向量的装配 | 91 |
| 5.4.5 边界条件的处理 | 94 |
| 5.4.6 L^2 和 H^1 误差分析 | 95 |
| 5.4.7 能量范数误差 | 97 |
| 5.4.8 程序整理 | 98 |
| 5.5 Poisson 方程的二阶虚拟元方法 | 103 |
| 5.5.1 装配指标 | 103 |
| 5.5.2 投影矩阵的计算 | 104 |
| 5.5.3 单元刚度矩阵和载荷向量的计算 | 106 |
| 5.5.4 刚度矩阵和载荷向量的装配 | 107 |
| 5.5.5 边界条件的处理 | 107 |
| 5.5.6 误差分析 | 108 |
| 5.6 Poisson 方程的三阶虚拟元方法 | 110 |
| 5.6.1 椭圆投影的计算 | 110 |
| 5.6.2 L^2 投影的计算 | 113 |
| 5.6.3 单元刚度矩阵和载荷向量的计算 | 116 |
| 5.6.4 刚度矩阵和载荷向量的装配 | 116 |
| 5.6.5 边界条件的处理 | 118 |
| 5.6.6 误差分析 | 118 |
| 6 三维 Poisson 方程的虚拟元方法 | 121 |
| 6.1 变分问题 | 121 |
| 6.1.1 虚拟元空间 | 121 |
| 6.1.2 虚拟元方法的变分问题 | 121 |
| 7 线弹性边值问题的虚拟元方法 | 122 |
| 7.1 线弹性边值问题简介 | 122 |
| 7.1.1 问题说明 | 122 |

| | | |
|----------|-------------------------------|------------|
| 7.1.2 | 连续变分问题 | 122 |
| 7.1.3 | 近似变分形式 I—Navier 形式 | 124 |
| 7.1.4 | 近似变分形式 II—tensor 形式 | 125 |
| 7.2 | 刚度矩阵和载荷向量的装配 | 125 |
| 7.2.1 | 矩阵分析法 | 125 |
| 7.2.2 | sparse 装配指标 | 126 |
| 7.3 | 变分形式 I: Navier 型 | 128 |
| 7.3.1 | 过渡矩阵 | 128 |
| 7.3.2 | 椭圆投影 | 129 |
| 7.3.3 | L^2 投影 | 131 |
| 7.3.4 | 单元刚度矩阵和载荷向量的计算 | 132 |
| 7.3.5 | 边界条件的处理 | 133 |
| 7.3.6 | 程序整理 | 134 |
| 7.4 | 变分形式 II: tensor 型 | 135 |
| 7.4.1 | 椭圆投影的定义 | 135 |
| 7.4.2 | 椭圆投影的计算 | 138 |
| 7.4.3 | 边界条件的处理 | 142 |
| 7.4.4 | 程序整理 | 143 |
| 8 | Kirchhoff 板弯问题的虚拟元方法 | 145 |
| 8.1 | Kirchhoff 板弯问题 | 145 |
| 8.1.1 | 模型假设 | 145 |
| 8.1.2 | 薄板弯曲的 Hooke 定律 | 147 |
| 8.1.3 | 变分原理 | 148 |
| 8.1.4 | 平衡方程 | 149 |
| 8.1.5 | 分部积分公式 | 151 |
| 8.2 | C^0 连续的非协调 VEM | 153 |
| 8.2.1 | C^0 连续的非协调虚拟元空间 | 153 |
| 8.2.2 | 过渡矩阵 D | 156 |
| 8.2.3 | 椭圆投影 | 157 |
| 8.2.4 | 单元刚度矩阵与载荷向量的计算 | 160 |
| 8.2.5 | 符号刚度矩阵和载荷向量 | 161 |
| 8.2.6 | 边界条件的处理 | 161 |
| 8.2.7 | 程序整理 | 162 |
| 8.2.8 | C^0 -强化处理 | 164 |
| 8.3 | Morley 型非协调 VEM | 165 |
| 8.3.1 | 过渡矩阵 D | 166 |
| 8.3.2 | 椭圆投影的计算 | 167 |
| 8.3.3 | 单元刚度矩阵与载荷向量的计算 | 170 |
| 8.3.4 | 符号刚度矩阵和载荷向量 | 171 |
| 8.3.5 | 边界条件的处理 | 172 |

| | |
|---|------------|
| 8.3.6 程序整理 | 173 |
| 8.4 C^1 协调 VEM | 174 |
| 8.4.1 虚拟元空间 | 174 |
| 8.4.2 计算说明 | 177 |
| III 奇异摄动问题的虚拟元方法 | 180 |
| 9 四阶奇异摄动问题的 C^0 元方法 | 181 |
| 9.1 一些说明 | 181 |
| 9.1.1 数学模型 | 181 |
| 9.1.2 分部积分公式 | 181 |
| 9.2 C^0 连续的非协调虚拟元空间 | 182 |
| 9.3 Poisson 项的处理 | 183 |
| 9.3.1 过渡矩阵 | 183 |
| 9.3.2 椭圆投影 Π_{Δ}^K | 184 |
| 9.3.3 a_{∇}^K 的椭圆投影 | 185 |
| 9.4 板弯项的处理 | 187 |
| 9.5 刚度矩阵与载荷向量的计算 | 189 |
| 9.6 边界条件的处理 | 191 |
| 9.7 误差分析 | 191 |

Part I

网格的生成与数据结构

1 二维网格的预处理

持续更新

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>

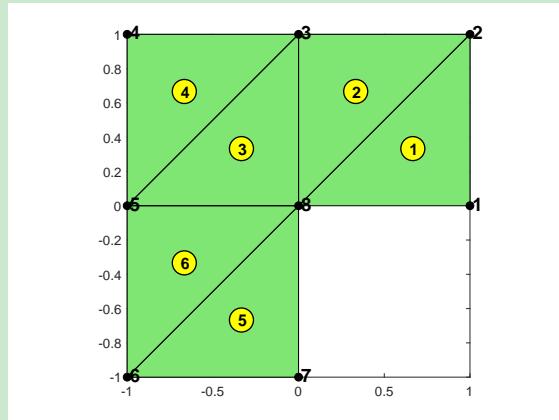


图 1. L 形区域的剖分

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)

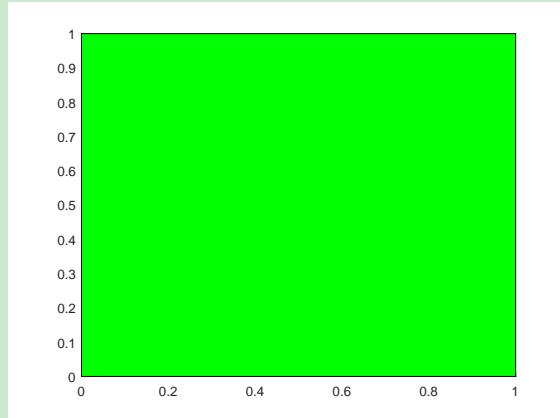
数组 elem 给出每个三角形的顶点编号, 它给出的是单元的连通性信息, 每行对应一个单元.

| 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

```

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

```

对多个相同类型的单元, 如下

```

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

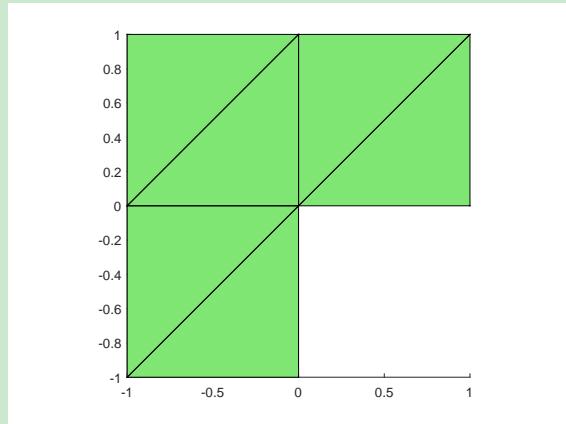
```

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

```

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

```

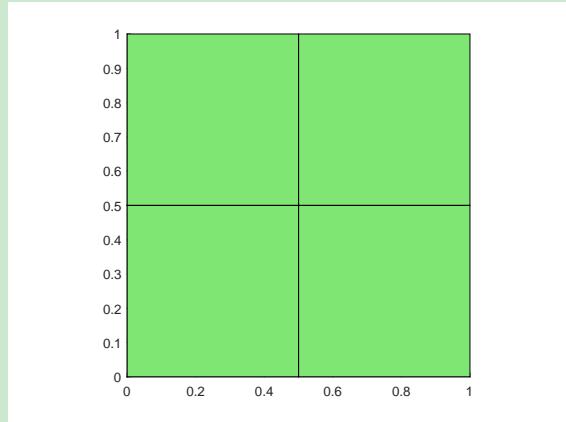


例 1.2 (四边形剖分) 对矩形区域的四边形剖分可如下调用 showmesh 函数

```

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

```

1 C = {1:10, [2; 4; 6], []};
2 averages = cellfun(@mean, C)
3 [nrows, ncols] = cellfun(@size, C)
4
5 % 结果为 averages = 5.5000    4.0000      NaN
6 %      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

```

1 days = {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'};

```

```
2 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

现在考虑图 2 所示的剖分

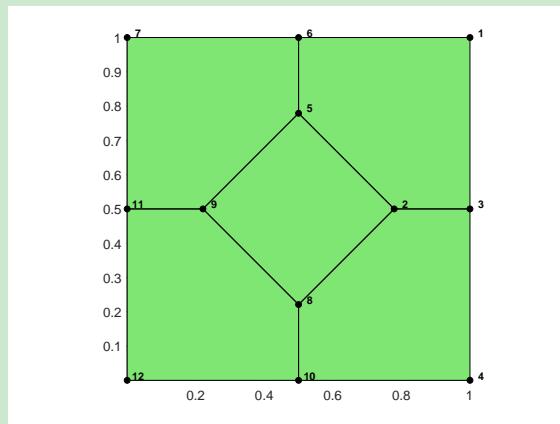


图 2. 多角形网格

相关的网格数据保持在 meshhex1.mat 中. 程序如下

```
1 load('meshhex1.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 %Showmesh displays a mesh in 2-D.
3
4 if ~iscell(elem)
5     h = patch('Faces', elem, 'Vertices', node);
6
7 else
8     max_n_vertices = max(cellfun(@length, elem));
9     padding_func = @(vertex_ind) [vertex_ind, ...
10         NaN(1,max_n_vertices-length(vertex_ind))]; % function to pad the vacancies
11     tpad = cellfun(padding_func, elem, 'UniformOutput', false);
```

```

12     tpad = vertcat(tpad{:});
13     h = patch('Faces', tpad, 'Vertices', node);
14 end
15
16 set(h, 'facecolor', [0.5 0.9 0.45], 'edgecolor', 'k');
17 axis equal; axis tight;

```

1.1.5 showsolution 函数的建立

程序如下

```

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 patch('Faces', elem, ...
6       'Vertices', data, ...
7       'FaceColor', 'interp', ...
8       'CData', u / max(abs(u)));
9 axis('square');
10 sh = 0.05;
11 xlim([min(node(:,1)) - sh, max(node(:,1)) + sh])
12 ylim([min(node(:,2)) - sh, max(node(:,2)) + sh])
13 zlim([min(u) - sh, max(u) + sh])
14 xlabel('x'); ylabel('y'); zlabel('u');
15
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 $\text{data} = [\text{node}, \text{u}]$ 就是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

```

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)) );
20 end
21 axis('square');
22 sh = 0.05;
23 xlim([min(node(:,1)) - sh, max(node(:,1)) + sh])
24 ylim([min(node(:,2)) - sh, max(node(:,2)) + sh])
25 zlim([min(u) - sh, max(u) + sh])
26 xlabel('x'); ylabel('y'); zlabel('u');
27
28 view(3); grid on; % view(150,30);

```

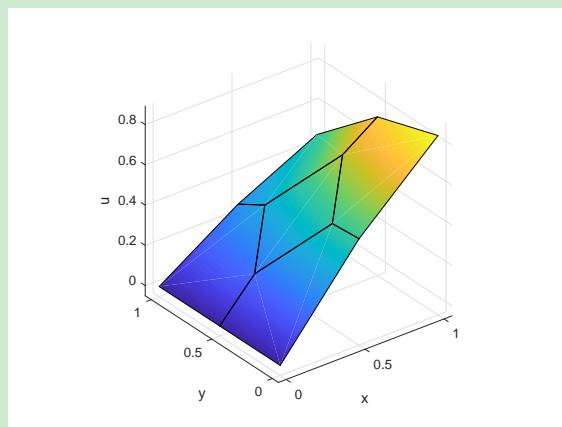
例 1.5 例如, 可如下画 $u(x,y) = \sin x \cos y$ 的图像

```

1 load('meshhex1.mat');
2 x = node(:,1); y = node(:,2); u = sin(x).*cos(y);
3 showsolution(node,elem,u);

```

结果如下



注 1.1 可以看到, showmesh 与 showsolution 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

```

1 function showmesh(node,elem)
2 %Showmesh displays a mesh in 2-D and 3-D.
3
4 if ~iscell(elem)
5     h = patch('Faces', elem, 'Vertices', node);
6 else
7     max_n_vertices = max(cellfun(@length, elem));
8     padding_func = @(vertex_ind) [vertex_ind, ...
9         NaN(1,max_n_vertices-length(vertex_ind))]; % function to pad the vacancies
10    tpad = cellfun(padding_func, elem, 'UniformOutput', false);
11    tpad = vertcat(tpad{:});
12    h = patch('Faces', tpad, 'Vertices', node);
13 end
14
15 dim = size(node,2);
16 if dim==3
17     view(3); set(h,'FaceAlpha',0.4); % 透明度
18 end
19
20 set(h,'facecolor',[0.5 0.9 0.45], 'edgecolor','k');
21 axis equal; axis tight;

```

显然, 用该函数也可画解的图像

```

1 load('meshhex1.mat');
2 x = node(:,1); y = node(:,2); u = sin(x).*cos(y);
3 % show the solution by using showmesh
4 data = [node,u];
5 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`

```

1 %Meshfun: genearte basic data structure (node, elem) representing meshes
2
3 clc;clear;close all
4 % ----- Choices of the domain -----
5 % Options: Rectangle_Domain, Circle_Domain, Upper_Circle_Domain,
6 % Upper_Circle_Circle_Domain, Rectangle_Circle_Domain, Circle_Circle_Domain
7 Domain = @Rectangle_Domain;
8 NT = 5; MaxIter = 300;
9 [node,elem]= PolyMesher(Domain,NT,MaxIter);
10
11 % ----- Plot the mesh -----
12 showmesh(node,elem);
13 findnode(node);
14
15 % ----- Save the mesh data -----

```

```
16 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

```
1 load('meshhex1.mat');
2 showmesh(node,elem);
3 findnode(node);
```

函数文件如下

CODE 5. findnode.m

```
1 function findnode(node,range)
2 %Findnode highlights nodes in certain range.
3
4 hold on
5 dotColor = 'k.';
6 if nargin==1
7     range = (1:size(node,1))';
8 end
9 plot(node(range,1),node(range,2),dotColor, 'MarkerSize', 15);
10 shift = [0.015 0.015];
11 text(node(range,1)+shift(1),node(range,2)+shift(2),int2str(range), ...
12      'FontSize',8,'FontWeight','bold'); % show index number
13 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. 需要给出单元的重心, 从而标记序号. 重心的计算后面说明.

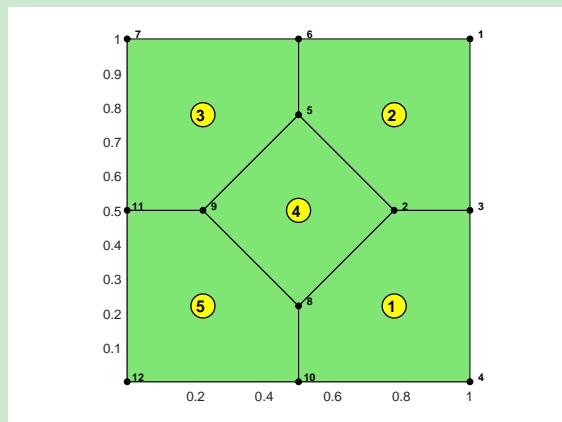


图 3. Polygonal mesh

主程序如下

```
1 load('meshhex1.mat');
2 showmesh(node,elem);
```

```
3 findnode(node);
4 findelem(node, elem);
```

标记单元的函数如下

CODE 6. findelem.m

```
1 function findelem(node, elem, range)
2 %Findelem highlights some elements
3
4 hold on
5
6 if nargin==2
7     range = (1:size(elem,1))';
8 end
9
10 center = zeros(length(range),2);
11 s = 1;
12 for iel = range(1):range(end)
13     if iscell(elem)
14         index = elem{iel};
15     else
16         index = elem(iel,:);
17     end
18     verts = node(index, :); verts1 = verts([2:end,1],:);
19     area_components = verts(:,1).*verts1(:,2)-verts1(:,1).*verts(:,2);
20     area = 0.5*abs(sum(area_components));
21     center(s,:) = sum((verts+verts1).*repmat(area_components,1,2))/(6*area);
22     s = s+1;
23 end
24
25 plot(center(:,1),center(:,2), 'o', 'LineWidth', 1, 'MarkerEdgeColor', 'k',...
26       'MarkerFaceColor', 'y', 'MarkerSize', 18);
27 text(center(:,1)-0.02,center(:,2),int2str(range), 'FontSize', 12, ...
28      'FontWeight', 'bold', 'Color', 'k');
29
30 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 $\text{edge}(k, 1) < \text{edge}(k, 2)$.

- 在有限元中, 通常规定三角形的第 i 条边对应第 i 个顶点 (该规定有利于网格二分程序的实现). 但对多边形区域, 通常规定第一个顶点逆时针右侧的边为第一个边 (若三角形仍采用以前的规定, 则后面编程时可能会出错). 仍以前面的规定考虑三角形边的标记.
- 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`

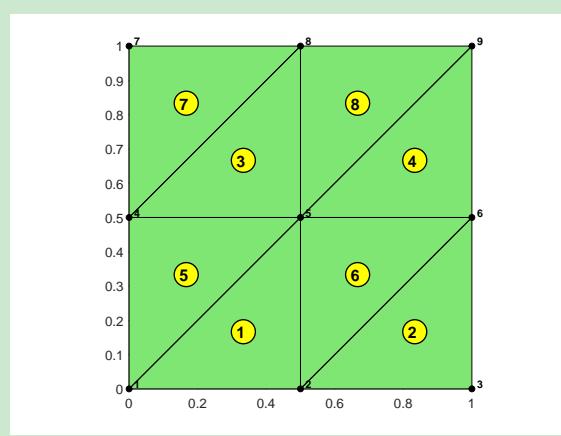
所有单元的第 2 条边: `elem(:,[3,1]); % NT * 2`

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

为了满足 $\text{edge}(k, 1) < \text{edge}(k, 2)$, 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)

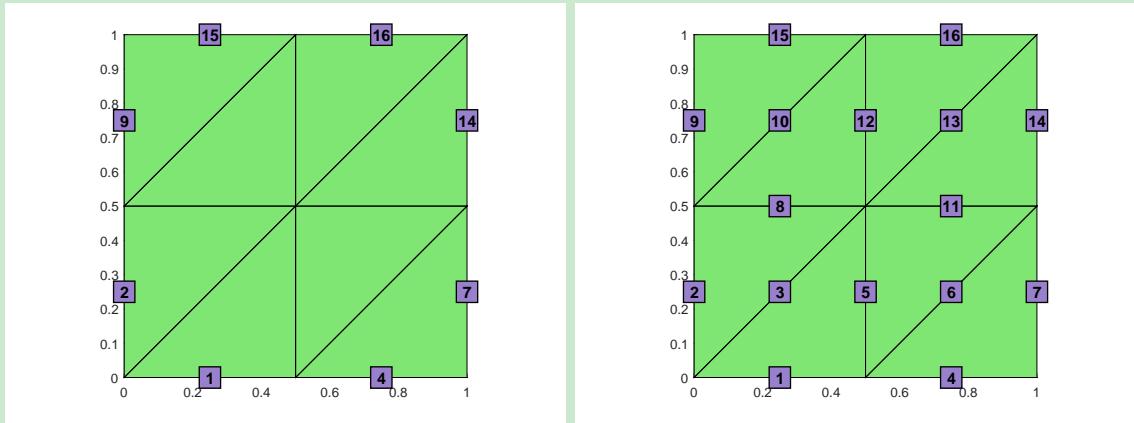
`sparse(totalEdge(:,1), totalEdge(:,2), 1)`

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, 如下标记边界边或所有的边

```

1 % ----- edge -----
2 [node,elem] = squaremesh([0 1 0 1],0.5);
3 figure, % boundary edges
4 showmesh(node,elem);
5 bdInd = 1;
6 findedgeTr(node,elem,bdInd);
7 figure, % all edges
8 showmesh(node,elem);
9 findedgeTr(node,elem);

```

函数文件如下

```

1 function fidedgeTr(node,elem,bdInd)
2 %FidedgeTr highlights edges for triangulation
3 % bdInd = 1; % boundary edge;
4 % other cases: all edges
5
6 hold on
7 % ----- edge matrix -----
8 totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
9 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
10 edge = [j,i];
11 % bdEdge = edge(s==1,:);
12
13 % ----- range -----
14 if nargin==2 || bdInd!=1
15     range = (1:size(edge,1))'; % all edges
16 else
17     range = find(s==1); % boundary edges
18 end
19
20 % ----- edge index -----
21 midEdge = (node(edge(range,1),:)+node(edge(range,2),:))/2;
22 plot(midEdge(:,1),midEdge(:,2),'s','LineWidth',1,'MarkerEdgeColor','k',...
23      'MarkerFaceColor',[0.6 0.5 0.8],'MarkerSize',20);
24 text(midEdge(:,1)-0.025,midEdge(:,2),int2str(range),...
25      'FontSize',12,'FontWeight','bold','Color','k');

```

注意因为前面进行了转置, $\text{edge} = [j, i]$.

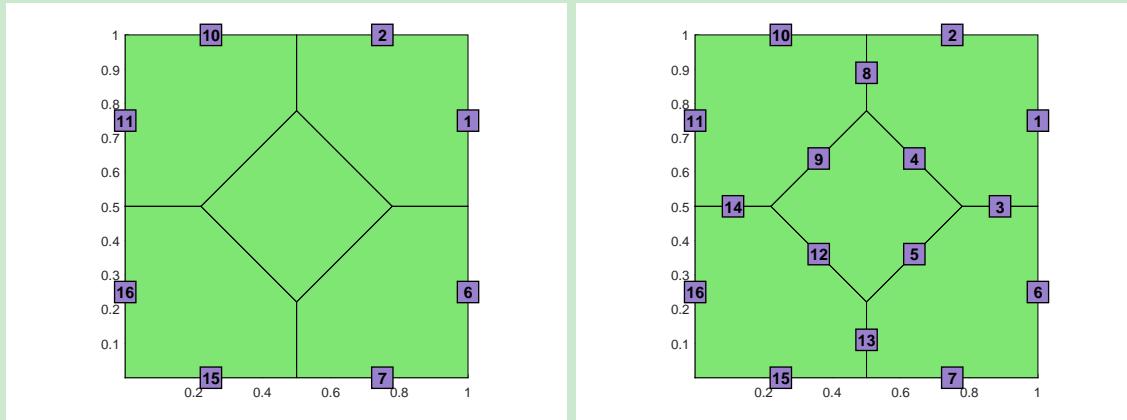
上面的思想也可用于多角形剖分, 只不过因边数不同, 要逐个单元存储每条边. 图 3 中第 1 个单元的顶点顺序为 [8,10,4,3,2], 我们按顺序 8-10, 10-4, 4-3, 3-2, 2-8 给出单元的边的标记. 所有边的起

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

```

1 % the starting points of edges
2 v0 = horzcat(elem{:})';
3
4 % the ending points of edges
5 shiftfun = @(verts) [verts(2:end),verts(1)];
6 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
7 v1 = horzcat(elem{:})';

```



其他过程与三角剖分一致. 如下运行

```

1 % ----- edge (polygonal meshes) -----
2 load('meshhex1.mat');
3 figure, % boundary edges
4 showmesh(node, elem);
5 bdInd = 1;
6 findedge(node, elem, bdInd)
7 figure, % all edges
8 showmesh(node, elem);
9 findedge(node, elem)

```

注 1.4 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”.

函数文件如下

CODE 7. `findedge.m`

```

1 function findedge(node, elem, bdInd)
2 %Finedge highlights edges
3 % bdInd = 1; % boundary edge;
4 % other cases: all edges
5
6 hold on
7
8 NT = size(elem,1);
9 if ~iscell(elem) % transform to cell
10     elem = mat2cell(elem, ones(NT,1), length(elem(1,:)));
11 end
12
13 % ----- edge matrix -----

```

```

14 shiftfun = @(verts) [verts(2:end),verts(1)];
15 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
16 v0 = horzcat(elem{:}); % the starting points of edges
17 v1 = horzcat(T1{:}); % the ending points of edges
18 totalEdge = sort([v0,v1],2);
19 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
20 edge = [j,i];
21 % bdEdge = edge(s==1,:);
22
23 % ----- range -----
24 if nargin==2 || bdInd!=1
25     range = (1:size(edge,1))'; % all edges
26 else
27     range = find(s==1); % boundary edges
28 end
29
30 % ----- edge index -----
31 midEdge = (node(edge(range,1),:)+node(edge(range,2),:))/2;
32 plot(midEdge(:,1),midEdge(:,2),'s','LineWidth',1,'MarkerEdgeColor','k',...
33 'MarkerFaceColor',[0.6 0.5 0.8],'MarkerSize',20);
34 text(midEdge(:,1)-0.025,midEdge(:,2),int2str(range),...
35 'FontSize',12,'FontWeight','bold','Color','k');

```

注 1.5 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. 我们更常用这种方式, 而且三维问题无法使用 `sparse` 的方式.

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>

本节针对一般的多边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

| | |
|-------------------------|---------------|
| <code>node, elem</code> | 基本数据结构 |
| <code>elem2edge</code> | 边的自然序号 (单元存储) |
| <code>edge</code> | 一维边的端点标记 |
| <code>bdEdge</code> | 边界边的端点标记 |
| <code>edge2elem</code> | 边的左右单元 |
| <code>neighbor</code> | 目标单元边的相邻单元 |
| <code>node2elem</code> | 顶点周围的单元 |

注 1.6 某些问题可能需要其他数据结构, 需要时我们再补充, 这里则不再说明.

1.4.1 elem2edge 的生成

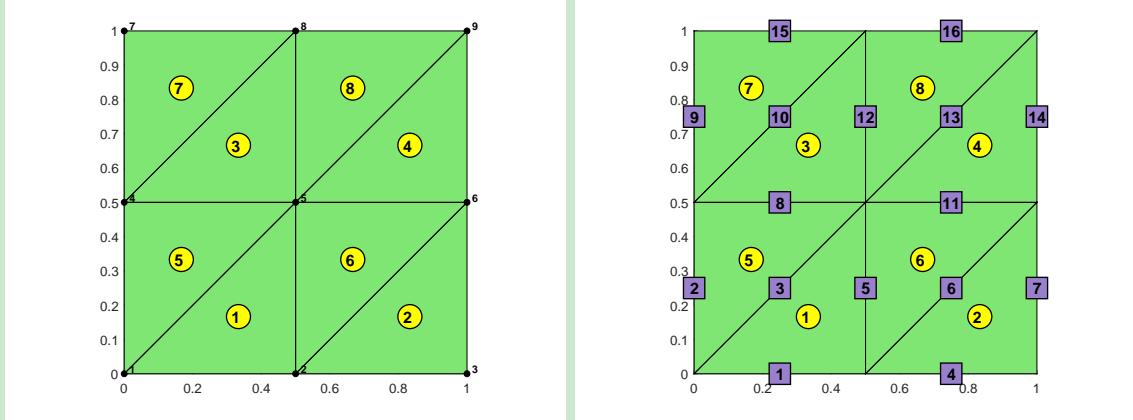


图 4. 三角剖分边的自然序号

考虑图 4 中给出的三角剖分, 下面说明一下 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).

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

图 5. elem2edge 图示

- 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).
- iAC 与 A 行数相同, 记录 A 在矩阵 C 中的位置.

- 这样, 如下获得边集合:

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

- edge 是 $NE \times 2$ 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 $NE \times 1$ 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.
- 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, \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)';

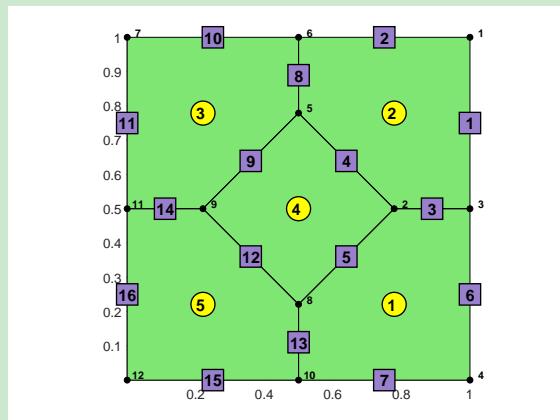
```

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.

```

1 NT = size(elem,1); N = size(node,1);
2 if ~iscell(elem) % transform to cell
3     elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
4 end
5
6 % totalEdge
7 shiftfun = @(verts) [verts(2:end),verts(1)]; % or shiftfun = @(verts) ...
    circshift(verts,-1);
8 T1 = cellfun(shiftfun, elem, 'UniformOutput', false);
9 v0 = horzcat(elem{:}); % the starting points of edges
10 v1 = horzcat(T1{:}); % the ending points of edges
11 totalEdge = sort([v0,v1],2);
12
13 % ----- elem2edge: elementwise edges -----
14 [~, i1, totalJ] = unique(totalEdge,'rows');
15 elemLen = cellfun('length',elem); % length of each elem
16 elem2edge = mat2cell(totalJ',1,elemLen)';
17
18 % ----- edge, bdEdge -----
19 [i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
20 edge = [j,i];
21 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 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 $\text{NE} \times 2$, 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, 对边界边规定两个编号一致.

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

| 16x2 double | |
|-------------|----|
| 1 | 2 |
| 1 | 2 |
| 2 | 4 |
| 3 | 5 |
| 4 | 3 |
| 5 | 2 |
| 6 | 6 |
| 7 | 3 |
| 8 | 5 |
| 9 | 4 |
| 10 | 7 |
| 11 | 5 |
| 12 | 12 |
| 13 | 8 |
| 14 | 14 |
| 15 | 10 |
| 16 | 16 |
| 17 | 11 |
| 18 | 19 |
| 19 | 4 |
| 20 | 20 |
| 21 | 21 |
| 22 | 5 |
| 23 | 9 |
| 24 | 12 |

| 16x1 double | |
|-------------|----|
| 1 | 2 |
| 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 |
| 17 | 5 |
| 18 | 7 |
| 19 | 12 |
| 20 | 14 |
| 21 | 2 |
| 22 | 5 |
| 23 | 9 |
| 24 | 12 |

| 24x1 double | |
|-------------|----|
| 1 | 2 |
| 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 |

(a) totalEdge

(b) edge

(c) i1

(d) totalJ

- 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 i_2 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 $\text{length}(\text{totalEdge})+1$ (三角形为 $3 \times NT + 1$). 这样, 还原后的为

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

- 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 totalJElement , 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 totalJElement = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
4 totalJElement = vertcat(totalJElement{:});
```

- 这样, 设第一次出现的行号为 i_1 , $\text{totalJElement}(i_1)$ 给出的就是左单元编号.

综上, 可如下实现 edge2elem

```
1 % ----- edge2elem -----
2 Num = num2cell((1:NT)');
3 Len = num2cell(elemLen);
```

```

3 totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
4 totalJelem = vertcat(totalJelem{:});
5 [~, i2] = unique(totalJ elem(end:-1:1), 'rows');
6 i2 = length(totalEdge)+1:i2;
7 edge2elem = totalJ elem([i1, i2]);

```

多角形剖分结果如下

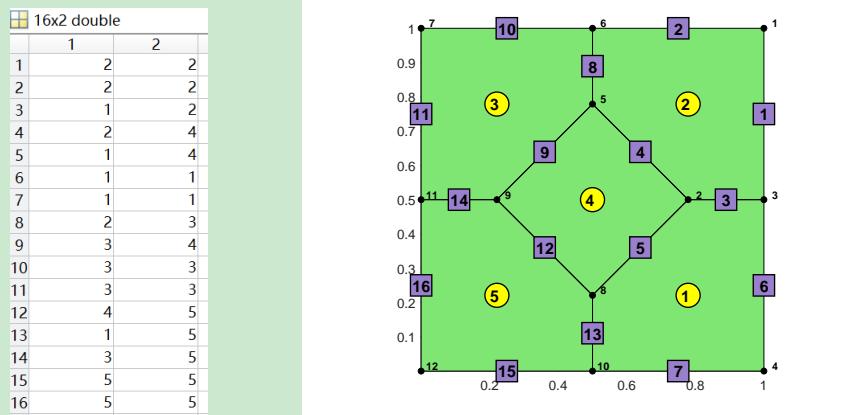


图 7. edge2elem

例如, 序号 12 的边连接的两个单元编号为 4 和 5.

注 1.7 totalEdge 是按单元顺序排列的, i_1 对应 e 第一次出现的单元, i_2 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 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, 规定其边界边为自身).

- 以图 7 的单元 4 为例, 它的边的局部顺序为 12-5-4-9. 这些边对应的左右单元如下

$$\begin{bmatrix} \text{左: } & 4 & 1 & 2 & 3 \\ \text{右: } & 5 & 4 & 4 & 4 \end{bmatrix}$$

- 将该矩阵按列拉直, 并去除当前单元的序号 4, 所得向量 $[5, 1, 2, 3]$ 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.
- 我们换一种处理方法, 即把上述矩阵每列的当前序号换成相邻单元序号. 例如, 第 1 列的 4 换成 5, 第 2 列的 4 换成 1, 等等. 这样处理后, 左右对应的行向量都是相邻单元序号.

上面的说明总结为如下程序.

```

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

```

1.4.4 node2elem 的生成

我们将给出一个稀疏矩阵 t_{2v} , 其尺寸为 $NT \times N$, 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 $totalJele$, 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 = totalJele; 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.

1.5 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.

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

```
1 function aux = auxstructure(node,elem)
2
3 NT = size(elem,1); N = size(node,1);
4 if ~iscell(elem) % transform to cell
5     elem = mat2cell(elem,ones(NT,1),length(elem(1,:)));
6 end
7
8 % totalEdge
9 shiftfun = @(verts) [verts(2:end),verts(1)]; % or shiftfun = @(verts) ...
10 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 [~, 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 -----
```

```

26 Num = num2cell((1:NT)'); Len = num2cell(elemLen);
27 totalJelem = cellfun(@(n1,n2) n1*ones(n2,1), Num, Len, 'UniformOutput', false);
28 totalJelem = vertcat(totalJelem{:});
29 [~, i2] = unique(totalJ(end:-1:1), 'rows');
30 i2 = length(totalEdge)+1-i2;
31 edge2elem = totalJelem([i1,i2]);
32
33 % ----- neighbor -----
34 NE = size(edge,1);
35 ii1 = edge2elem(:,1); jj1 = (1:NE)'; ss1 = edge2elem(:,2);
36 ii2 = edge2elem(:,2); jj2 = (1:NE)'; ss2 = edge2elem(:,1);
37 label = (ii2≠ss2);
38 ii2 = ii2(label); jj2 = jj2(label); ss2 = ss2(label);
39 ii = [ii1;ii2]; jj = [jj1;jj2]; ss = [ss1;ss2];
40 neighbor = sparse(ii,jj,ss,NT,NE);
41
42 % ----- node2elem -----
43 ii = totalJelem; jj = v0; ss = ones(length(ii),1);
44 t2v = sparse(ii,jj,ss,NT,N);
45 node2elem = cellfun(@(x) find(x), num2cell(t2v,1), 'UniformOutput', false);
46 node2elem = node2elem';
47
48 aux.node = node; aux.elem = elem;
49 aux.elem2edge = elem2edge;
50 aux.edge = edge; aux.bdEdge = bdEdge;
51 aux.edge2elem = edge2elem;
52 aux.neighbor = neighbor; aux.node2elem = node2elem;

```

auxgeometry.m 函数如下

CODE 9. 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;

```

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 `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).

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 $\text{edge}(k, 1) < \text{edge}(k, 2)$. 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),:);

```

1.7.2 边界的设置

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

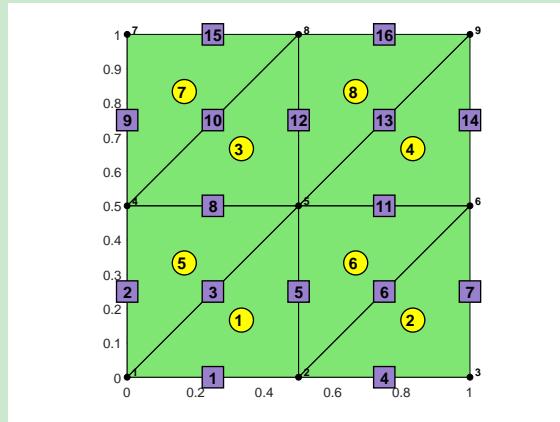


图 8. 边的自然序号

- 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

```

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 $\text{id} = [0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0]'$, 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 $\text{bdEdge}(\text{id},:)$. 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

```

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.bdEdgeD = bdEdge(Idx,:); % Dirichlet boundary edges
38 bdStruct.bdEdgeN = bdEdge(~Idx,:); % Neumann boundary edges
39 bdStruct.bdNodeIdx = unique(bdEdge(Idx,:)); % index of Dirichlet boundary nodes
40 bdEdgeIdx = find(s==1); % index of all boundary edges
41 bdStruct.bdEdgeIdx = bdEdgeIdx;
42 bdStruct.bdEdgeIdxD = bdEdgeIdx(Idx); % index of Dirichelt boundary edges
43 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 1.8 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 MPT: 多角形与多面体区域的网格剖分

待整理

本文使用 MPT2, 现在已经有 MPT3, 但后者是采用面向对象的形式组织程序的, 不太容易理解和修改程序. MPT 只能处理多角形或多面体区域.

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 $P = \{x_1, \dots, x_n\}$ 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

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

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.

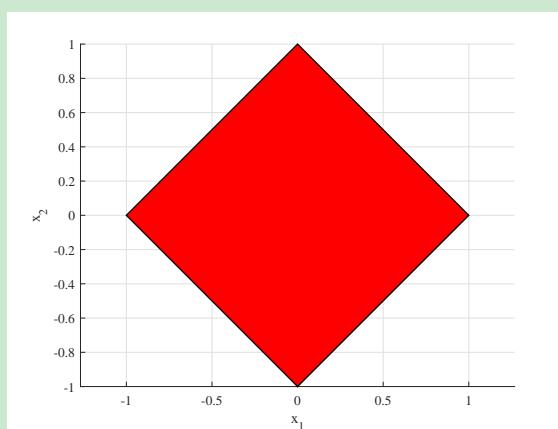
2.1.2 MPT 中多面体的表示

多面体是由若干个空间半平面相交所得, 而平面的方程可写为 $a_1x_1 + \dots + a_nx_n = b$, 这里, $x = (x_1, \dots, x_n)^T$ 是平面上的一点, $a = (a_1, \dots, a_n)^T$ 是平面的单位法向量, $|b|$ 是平面到原点的距离. 一般地, 多面体可写为

$$\mathcal{P} = \{x \in \mathbb{R}^n : Hx \leq K\}, \tag{2.1}$$

这里, K 是列向量, H 的每行对应多面体的一个面.

式 (2.1) 这种表述法可容易判断给定点在多面体区域的内外, MPT 中调用函数 `isinside.m` 确定.



例 2.1 (多角形) 绘制连接 Ox_1x_2 轴上距离原点为 1 的顶点组成的多角形.

通过判断原点满足的不等式, 可知多角形内的点满足

$$\frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ -1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \frac{\sqrt{2}}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

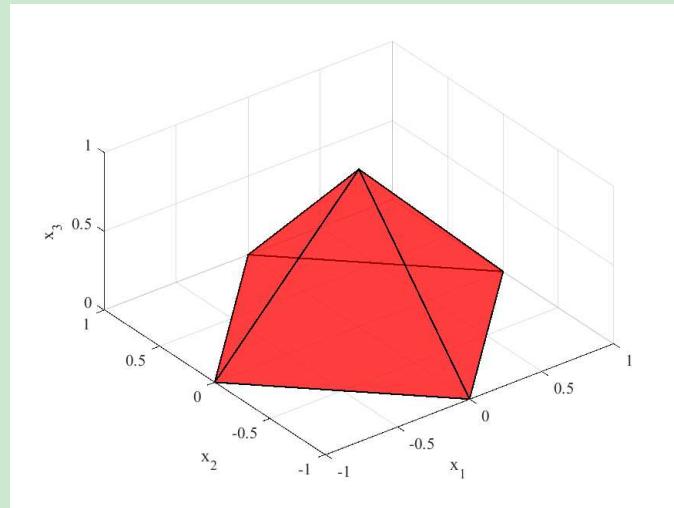
绘图程序如下 (可以不是单位向量)

```
1 H = sqrt(2)/2 * [1 1; -1 1; -1 -1; 1 -1];
2 K = sqrt(2)/2 * ones(4,1);
3 P = polytope(H,K);
4 figure, plot(P)
```

这里的 plot 函数是 MPT 中的.

也可直接画顶点确定的凸包, 程序如下

```
1 V = [1 0; 0 1; -1 0; 0 -1];
2 P = polytope(V);
3 figure, plot(P); axis equal
```



例 2.2 (多面体) 绘制连接 $Ox_1x_2x_3$ 轴上距离原点为 1 的顶点组成的多面体.

```
1 H = 1/sqrt(3) * [1 1 1; -1 1 1; -1 -1 1; 1 -1 1; 0 0 -1];
2 K = 1/sqrt(3) * [1; 1; 1; 1; 0];
3 P = polytope(H,K);
4 figure, plot(P); axis equal
```

或直接画凸包

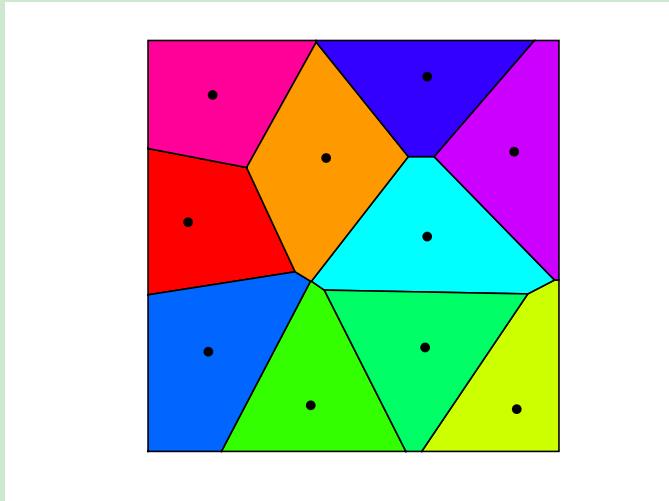
```
1 V = [1 0 0; 0 1 0; -1 0 0; 0 -1 0; 0 0 1];
2 P = polytope(V);
3 figure, plot(P); axis equal
```

2.1.3 带边界的 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, 这里暂时不考虑, 而使用 MPT.

对给定的凸区域 Ω , 可事先在区域边界上给定若干个点, 以生成区域边界的多面体近似, 之后再获取 Voronoi 剖分. MPT 中将多面体边界视为优化问题的不等式约束, 而 Voronoi 划分等价于一个参数优化问题. 具体说明略. 注意, MPT 这种求法相对 Delaunay 三角剖分算法要慢很多.

先考虑二维区域.



- 给定矩形区域的 4 个顶点, 其对应的凸包就是矩形区域.

```

1 % ----- 区域边界的凸包近似 -----
2 Vb = [0 0; 0 1; 1 0; 1 1];
3 Pb = polytope(Vb);
4 Options.plot = 1;
5 Options.pbound = Pb;

```

- 接着我们要在该区域内给定若干个种子, 可如下随机给出

```

1 % ----- 随机生成 Voronoi 种子 -----
2 NT = 10; P = zeros(NT,2);
3 s = 1; BdBox = [0 1 0 1]; x0 = zeros(2,1); % 列向量
4 while s<=NT
5     x0(1) = (BdBox(2)-BdBox(1))*rand + BdBox(1);
6     x0(2) = (BdBox(4)-BdBox(3))*rand + BdBox(3);
7     if isinside(Pb,x0)
8         P(s,:) = x0'; s = s+1;
9     end
10 end

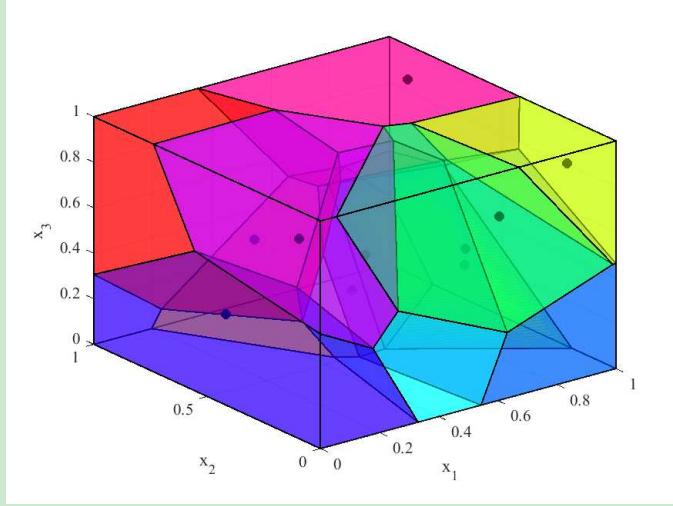
```

- 带边界的 Voronoi 划分如下给出

```

1 % ----- 带边界的 Voronoi 划分 -----
2 Pn = mpt_voronoi(P,Options);
3 axis([0 1 0 1])

```



类似的, 三维正方体可如下给出 Voronoi 划分.

```

1 % ----- 区域边界的凸包近似 -----
2 Vb = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
3 Pb = polytope(Vb);
4 Options.plot = 1;
5 Options.pbound = Pb; figure.plot(Pb)
6 % ----- 随机生成 Voronoi 种子 -----
7 NT = 10; P = zeros(NT,3);
8 s = 1; BdBox = [0 1 0 1 0 1]; x0 = zeros(3,1); % 列向量
9 while s≤NT
10     x0(1) = (BdBox(2)-BdBox(1))*rand(1) + BdBox(1);
11     x0(2) = (BdBox(4)-BdBox(3))*rand(1) + BdBox(3);
12     x0(3) = (BdBox(6)-BdBox(5))*rand(1) + BdBox(5);
13     if isinside(Pb,x0)
14         P(s,:) = x0'; s = s+1;
15     end
16 end
17 % ----- 带边界的 Voronoi 划分 -----
18 Pn = mpt_voronoi(P,Options);
19 axis([0 1 0 1 0 1])

```

2.2 Lloyd 迭代

2.2.1 CVTs 及其变分描述

Voronoi 图的质量与种子的分布有关. 为了获得更高质量的剖分, 常采用重心 Voronoi 划分 (the centroid Voronoi tessellation, CVT). 给定密度函数 $\mu(x)$, Voronoi 单元的重心定义为

$$y_c = \frac{\int_{V_y \cap \Delta} x \mu(x) dx}{\int_{V_y \cap \Delta} \mu(x) dx}. \quad (2.2)$$

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

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

注 2.1 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

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

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

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

2.2.2 计算 CVTs 的 Lloyd 算法

定义 2.3 定义

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

其中

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

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 $k \rightarrow \infty$ 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

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

因而 Lloyd 算法可视为求能量泛函极值点的下降算法.

2.2.3 Lloyd 迭代的实现

Lloyd 迭代非常简单, 只需要新剖分的单元重心作为新的种子即可. MPT 中有两个函数涉及到形心, 即 analyticCenter.m 和 chebyball.m. 后者是包含在区域中球的球心, 前者似乎不是单元的重心, 因为迭代的结果很差 (使用 chebyball.m 好点). 下面使用前面数据结构中给出的计算方法, 为此要把单元顶点循环排列, 程序如下

```

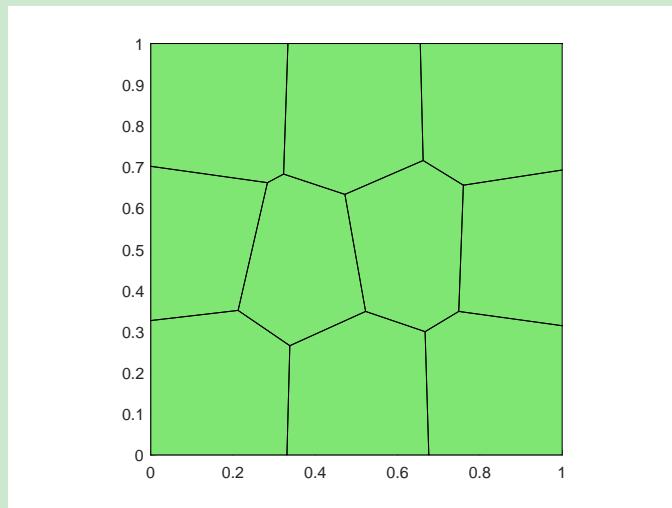
1 % ----- Lloyd's iteration -----
2 Pn = mpt_voronoi(P, Options); nodeElem = cell(NT, 1); iter = 1;
3 while (iter <= MaxIter)
4     for iel = 1:NT
5         % cyclic order
6         [V, -, -, adjV] = extreme(Pn(iel));
7         nv = size(V, 1); order = zeros(nv, 1);
8         previous = 1; adj = adjV{previous};
9         current = adj(1);
10        order(1) = previous; order(2) = current;
11        for p = 2:nv-1
12            adj = adjV{current}; next = adj(adj > previous);
13            order(p+1) = next;

```

```

14         previous = current; current = next;
15     end
16     Vnew = V(order,:); nodeElem{iel} = Vnew;
17     % centroid
18     vx = Vnew(:,1); vy = Vnew(:,2);
19     vxS = vx([2:nv,1]); vyS = vy([2:nv,1]);
20     temp = vx.*vyS - vy.*vxS; A = 0.5*sum(temp);
21     P(iel,:) = 1/(6*A)*[sum((vx+vxS).*temp),sum((vy+vyS).*temp)];
22 end
23 Pn = mpt_voronoi(P,Options); iter = iter+1;
24 end

```



注 2.2 这里循环序号的实现后面说明. 可以看到, 用 MPT 实现 CVTs 相当容易, 但基本数据结构 `node`, `elem` 不能直接获得.

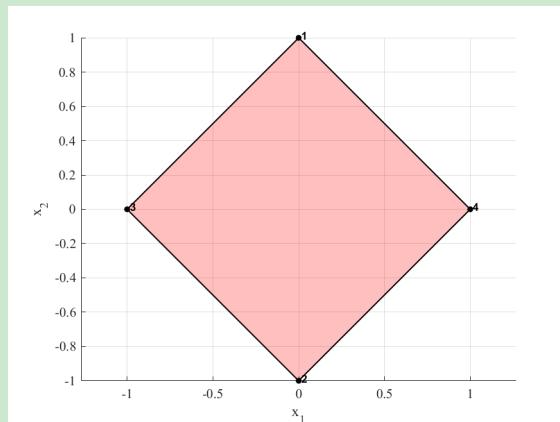
2.3 2D 基本数据结构的获取

2.3.1 恢复单元节点的逆序

可用如下命令获取具体单元的顶点坐标

```
1 V = extreme(Pn(1));
```

图示可以发现, 单元顶点按自然序号给的, 而且没有固定的顺序. 为此, 首先需要将单元恢复成逆时针的自然序号.



以上图为例, 我们来说明做法.

- `extreme` 可输出每个顶点相邻的两个顶点序号, 如下

```
1 [V,~,~,adjV] = extreme(Pn(1));
```

单元的第 1 个顶点连接的顶点为 4, 3, 则 `adjV` 的第一行就为数组 [4,3]. 实际上 `adjV` 是用元胞数组存储的, 二维连接的顶点个数都为 2, 除非是无界的单元. 三维大多数是连接 3 个点.

- 先生成循环的点, 即逆时针或顺时针排列的. 从第 1 个顶点开始, 由 `adjV{1}` 可知它连接的点序号为 [4,3], 则循环的第 2 个点可取为点 4. 查看点 4 连接的点为 [1,2], 显然接下来取不是循环中前面的那个点, 即点 2.

```
1 % cyclic order
2 [V,~,~,adjV] = extreme(Pn(iel));
3 nv = size(V,1); order = zeros(nv,1);
4 previous = 1; adj = adjV{previous};
5 current = adj(1);
6 order(1) = previous; order(2) = current;
7 for p = 2:nv-1
8     adj = adjV{current}; next = adj(adj~=previous);
9     order(p+1) = next;
10    previous = current; current = next;
11 end
12 V = V(order,:);
```

- 逆时针顺序可用前三个点确定的三角形的有向面积来获得. 在 Lloyd 迭代中已经实现顶点的循环排序, 为此在计算过程中把单元记录下来 (即 `nodeElem`), 再逆时针排序.

```
1 % ----- Pn 单元逆序 -----
2 elemLen = cellfun('length',double(Pn)); totalN = sum(elemLen);
3 node = zeros(totalN,2); s = 1;
4 for iel = 1:NT
5     Vnew = nodeElem{iel}; nV = size(Vnew,1);
6     z1 = Vnew(1,:); z2 = Vnew(2,:); z3 = Vnew(3,:);
7     e2 = z3-z1; e3 = z1-z2;
8     area = 0.5*(-e3(:,1).*e2(:,2)+e3(:,2).*e2(:,1));
9     if area<0, Vnew = Vnew(end:-1:1,:); end
10    node(s:s+nV-1,:) = Vnew; s = s+nV;
11 end
```

2.3.2 获得 `node` 和 `elem`

前面将所有单元的顶点逐行排列, 其中有些行是重复的. 为此必须删除重复的, 规定其顺序, 相应地要调整 `elem`.

- 对第 1 行的顶点, 可将其与后面的点比较, 若相同, 则将重复点的序号改为 1. 对第 2 行的点, 若它不属于之前改过序号的点 (自然序号比前面查看后的点序号大, 因为改过序号的都是前面处理过的行的序号), 则可重复前面的操作.

```
1 s = 1; totalid = (1:totalN)';
2 for i = 1:totalN
3     for j = i+1:totalN
4         p1 = node(i,:); p2 = node(j,:);
```

```

5         if norm(p1-p2)≤1e-4 && totalid(j)>i
6             totalid(j) = i;
7         end
8     end
9 end

```

totalid 记录的就是更改后的序号.

- 类似以前数据结构中的操作, 利用 unique 函数可获得基本数据结构

```

1 [id, ~, totalJ] = unique(totalid, 'rows');
2 node = node(id,:);
3 elem = mat2cell(totalJ', 1, elemLen)';

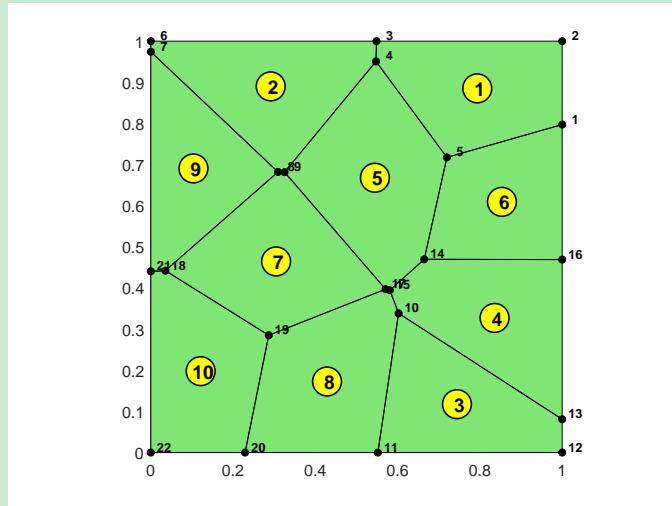
```

这里,

- id 是去除重复后的网格节点的原始编号, 其行索引作为新的编号, 称为自然序号.
- totalJ 按 elem 的拉直模式记录每个节点的自然序号 (即 id 的行索引).

2.3.3 去除短边

Voronoi 划分容易产生短边, 如下图所示.



注意, 短边是相对于单元来说的, 总体短的边不一定是短边.

下面考虑去除短边.

- 根据辅助数据结构的讨论过程, 我们可获得所有的一维边集合 edge

```

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

```

- 我们将删除如下的边

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

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.

```

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

```

这里 $nv1, nv2$ 是要删除的边的起点和终点编号.

- 短边可如下删除, 即把要删除边的终点编号替换为起点编号 (称保留起点编号)

```

1 [id,~,totalid] = unique(horzcat(elem{:})');
2 for i = 1:length(nv2)
3     v1 = nv1(i); v2 = nv2(i);
4     totalid(totalid==v2) = v1;
5 end
6 elemLen = cellfun('length',elem);
7 elem = mat2cell(totalid', 1, elemLen)';

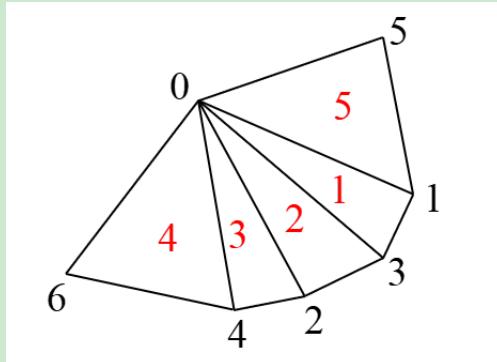
```

注意, 尽管 $totalid$ 对应 id 的行索引, 但因前面已经是自然序, 从而 id 本身就是自然序排列的. 对一般情形, 应该将 $totalid$ 改为

```

1 totalid = id(totalid);

```



替换做法存在一个问题, 即错过某些短边或进行了假处理. 为了方便, 以上图为例 (多角形为 0-6-4-2-3-1-5), 假设三角形 1,2,3 的顶点 0 所对边为短边.

- 边的序号规定: 起点编号小于终点编号.
- 去除短边 1-3 时, $totalid$ 中所有序号 3 都会替换成 1.
- 去除短边 2-3 时, 按照处理方法, $totalid$ 中所有序号 3 都会替换成 2. 但上一步操作时, $totalid$ 中已不存在序号 3, 因而这条短边并没有处理.
- 解决的办法是把 $nv1, nv2$ 的序号, 即边的起点和终点序号同时替换. 如下

```

1 for i = 1:length(nv2)
2     v1 = nv1(i); v2 = nv2(i);
3     totalid(totalid==v2) = v1;
4     nv1(nv1==v2) = v1;
5     nv2(nv2==v2) = v1;
6 end

```

经过上面的处理, 点 1,3,2,4 最终都会编号为 2, 即所有短边退化为点 2.

现在继续去除短边的操作.

- 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 (显然不会改变原来的节点顺序).

```

1 for iel = 1:NT
2     index = elem{iel};
3     [~,i1] = unique(index);
4     elem{iel} = index(sort(i1));
5 end

```

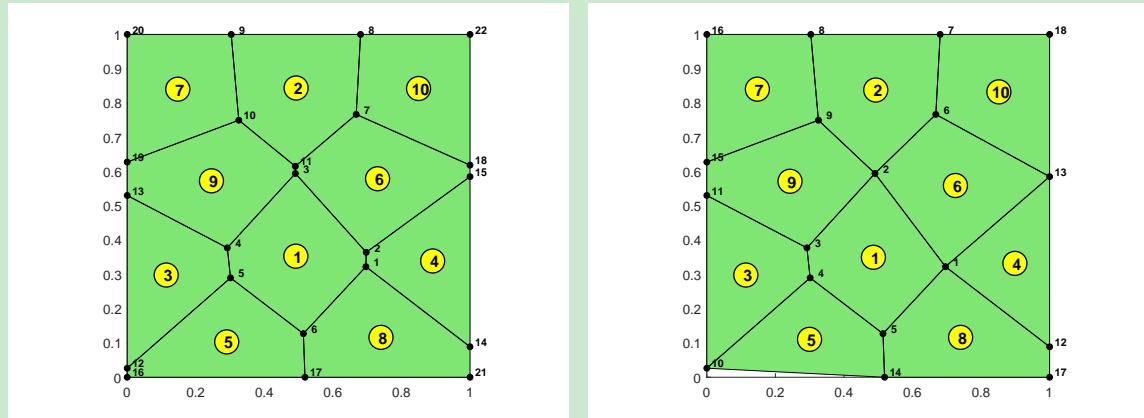
- 一般而言, NT 个单元中可能存在三角形单元, 而且该三角形单元含有短边. 此时, 短边顶点替换后, 该单元将退化为线, 成为无用单元. **这种情况其实很难出现, 为此程序中不处理这个问题.**
- 注意, 此时节点的编号可能不是连续的, 这可类似前面只有局部序号的方法处理. 现在的单元节点是逆序的, 可以不用考虑逆序问题.

```

1 [id,~,totalid] = unique(horzcat(elem{:})');
2 node = node(id,:);
3 elemLen = cellfun('length',elem);
4 elem = mat2cell(totalid', 1, elemLen)';

```

用起点编号替换终点编号的做法实际上还存在另一个问题, 如下图所示.



该图左侧是原始剖分图, 右侧为去除短边的. 显然, 在边界处, 图中的剖分应该保留终点编号. 容易发现, 对合适的矩形边框 BdBox, 应该保留横纵方向最小距离之和更小的点. 为此把前面替换的过程改为

```

1 for i = 1:length(nv2)
2     v1 = nv1(i); v2 = nv2(i); iv = ir(i);
3     ax = min(abs([z1(iv,1)-BdBox(1), BdBox(2)-z1(iv,1)]));
4     ay = min(abs([z1(iv,2)-BdBox(3), BdBox(4)-z1(iv,2)]));
5     bx = min(abs([z2(iv,1)-BdBox(1), BdBox(2)-z2(iv,1)]));
6     by = min(abs([z2(iv,2)-BdBox(3), BdBox(4)-z2(iv,2)]));
7     if (ax+ay)<(bx+by)
8         totalid(totalid==v2) = v1; % 保留起点编号
9         nv1(nv1==v2) = v1;
10        nv2(nv2==v2) = v1;
11    else

```

```

12     totalid(totalid==v1) = v2;
13     nv1(nv1==v1) = v2;
14     nv2(nv2==v1) = v2;
15   end
16 end

```

2.3.4 程序整理

在 tool/MPT 文件夹中给出了 mpt_meshfun2D.m 函数, 如下

```

1 % ----- Usage -----
2 % Preserved as meshdata.mat
3 % load('meshdata.mat') to load basic data structure: node, elem
4 %
5
6 clc;clear;close all
7
8 % ----- Choice of the domain -----
9 NT = 20; MaxIter = 30;
10 BdBox = [0 1 0 1];
11 Pb = mpt_rectangle_domain(BdBox);
12 [node,elem] = MPT_Mesher2D(Pb,NT,MaxIter,BdBox);
13
14 % ----- Visualization of the mesh -----
15 showmesh(node,elem);
16 %findnode(node); % findelem(node,elem)
17
18 % % ----- Save the mesh data -----
19 % save meshdata node elem

```

函数文件如下

CODE 11. MPT_Mesher2D.m

```

1 function [node,elem] = MPT_Mesher2D(Pb,NT,MaxIter,BdBox)
2
3 % ----- Convex hull approximatation of boundary -----
4 Options.pbound = Pb;
5
6 % ----- Generation of intial pointset -----
7 P = zeros(NT,2); Nbox = length(BdBox); s = 1;
8 while s<=NT
9     x0 = (BdBox(2:2:Nbox)-BdBox(1:2:Nbox-1)).*rand(1,2) + BdBox(1:2:Nbox);
10    if isinside(Pb,x0(:))
11        P(s,:) = x0; s = s+1;
12    end
13 end
14
15 % ----- Lloyd's iteration -----
16 Pn = mpt_voronoi(P,Options); nodeElem = cell(NT,1); iter = 1;
17 while (iter<=MaxIter)
18     for iel = 1:NT
19         % cyclic order
20         [V,~,~,adjV] = extreme(Pn(iel));
21         nv = size(V,1); order = zeros(nv,1);
22         previous = 1; adj = adjV{previous};
23         current = adj(1);

```

```

24     order(1) = previous; order(2) = current;
25     for p = 2: nv-1
26         adj = adjV{current}; next = adj(adj≠previous);
27         order(p+1) = next;
28         previous = current; current = next;
29     end
30     V = V(order,:); nodeElem{iel} = V;
31     % centroid
32     vx = V(:,1); vy = V(:,2);
33     vxS = vx([2:nv,1]); vyS = vy([2:nv,1]);
34     temp = vx.*vyS - vy.*vxS; A = 0.5*sum(temp);
35     P(iel,:) = 1/(6*A)*[sum((vx+vxS).*temp),sum((vy+vyS).*temp)];
36 end
37 Pn = mpt_voronoi(P,Options); iter = iter+1;
38 end
39
40 % ----- Counterclockwise order of elements in Pn -----
41 elemLen = cellfun('length',double(Pn)); totalN = sum(elemLen);
42 node = zeros(totalN,2); s = 1;
43 for iel = 1:NT
44     V = nodeElem{iel}; nV = size(V,1);
45     z1 = V(1,:); z2 = V(2,:); z3 = V(3,:);
46     e2 = z3-z1; e3 = z1-z2;
47     area = 0.5*(-e3(:,1).*e2(:,2)+e3(:,2).*e2(:,1));
48     if area<0, V = V(end:-1:1,:); end
49     node(s:s+nV-1,:) = V; s = s+nV;
50 end
51
52 % ----- node, elem -----
53 totalid = (1:totalN)';
54 for i = 1:totalN
55     for j = i+1:totalN
56         p1 = node(i,:); p2 = node(j,:);
57         if norm(p1-p2)≤1e-4 && totalid(j)>i
58             totalid(j) = i;
59         end
60     end
61 end
62 [id, ~, totalJ] = unique(totalid,'rows');
63 node = node(id,:);
64 elem = mat2cell(totalJ',1,elemLen)';
65
66 % ----- Remove small edges -----
67 T1 = cellfun(@(verts) [verts(2:end),verts(1)], elem, 'UniformOutput', false);
68 v0 = horzcat(elem{:})'; v1 = horzcat(T1{:})';
69 totalEdge = sort([v0,v1],2);
70 [i,j] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
71 edge = [j,i];
72 z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
73 he = sqrt(sum((z1-z2).^2,2));
74 ir = find(he < 0.1*max(he));
75 nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices
76 [id,~,totalid] = unique(horzcat(elem{:})');
77 totalid = id(totalid); % actually not needed since id = 1,2,...
78 for i = 1:length(nv2)
79     v1 = nv1(i); v2 = nv2(i); iv = ir(i);
80     ax = min(abs([z1(iv,1)-BdBox(1), BdBox(2)-z1(iv,1)]));
81     ay = min(abs([z1(iv,2)-BdBox(3), BdBox(4)-z1(iv,2)]));

```

```

82     bx = min(abs([z2(iv,1)-BdBox(1), BdBox(2)-z2(iv,1)]));
83     by = min(abs([z2(iv,2)-BdBox(3), BdBox(4)-z2(iv,2)]));
84     if (ax+ay)<(bx+by)
85         totalid(totalid==v2) = v1; % replaced by starting indices
86         nv1(nv1==v2) = v1;
87         nv2(nv2==v2) = v1;
88     else
89         totalid(totalid==v1) = v2;
90         nv1(nv1==v1) = v2;
91         nv2(nv2==v1) = v2;
92     end
93 end
94 elemLen = cellfun('length',elem);
95 elem = mat2cell(totalid', 1, elemLen)';
96 for iel = 1:NT
97     index = elem{iel};
98     [~,i1] = unique(index);
99     elem{iel} = index(sort(i1));
100 end
101 [id,~,totalid] = unique(horzcat(elem{:})');
102 node = node(id,:);
103 elemLen = cellfun('length',elem);
104 elem = mat2cell(totalid', 1, elemLen)';

```

注 2.3 MPT 生成网格的速度非常慢, 原因在于它把求 Voronoi 图的问题转化为一个多参数优化问题, 后者速度相对 Delaunay 三角剖分来说要慢得多.

2.4 3D 基本数据结构的获取

2.4.1 单元面的提取

生成三维剖分的过程与二维一样, 前面已经给出, 这里总结如下

```

1 clc;clear;close all
2
3 NT = 10; MaxIter = 10;
4 % ----- 区域边界的凸包近似 -----
5 BdBox = [0 1 0 1 0 1];
6 Pb = mpt_rectangle_domain(BdBox);
7 Options.plot = 0;
8 Options.pbound = Pb;
9
10 % ----- 随机生成 Voronoi 种子 -----
11 P = zeros(NT,3); Nbox = length(BdBox); s = 1;
12 while s≤NT
13     x0 = (BdBox(2:2:Nbox)-BdBox(1:2:Nbox-1)).*rand(1,3) + BdBox(1:2:Nbox);
14     if isinside(Pb,x0(:))
15         P(s,:) = x0; s = s+1;
16     end
17 end
18
19 % ----- Lloyd 迭代 -----
20 Pn = mpt_voronoi(P,Options);
21 for iter = 1:MaxIter
22     for iel = 1:NT
23         P(iel,:) = analyticCenter(Pn(iel));
24     end

```

```

25      Pn = mpt_voronoi(P,Options);
26 end
27 % Options.plot = 1;
28 % figure,Pn = mpt_voronoi(P,Options); axis equal
29 Options.shade = 0.5;
30 figure,mpt_plot(Pn,Options); axis equal

```

注 2.4 这里用的是 analyticCenter.m 计算重心, 实际上它给出的并不是重心. 多面体的重心可用数学分析中给出的公式计算, 即式 (2.2) ($\mu = 1$). 而其中的积分用 Gauss 公式可转化为面的积分. 在 MATLAB 中, 给定凸包的顶点 v, 命令 `Tri = convhulln(v)` 生成面的三角剖分, 从而面上的积分又由这些三角形上的积分之和获得. 重心用如下函数实现

```

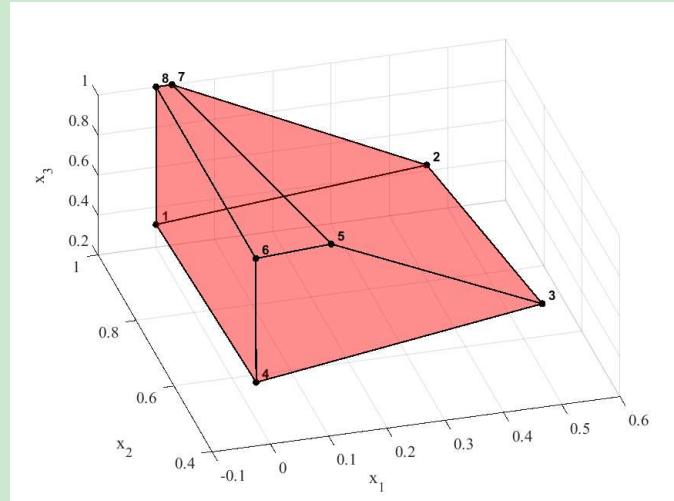
1 P(iel,:)=Polycentroid(V,Tri);

```

后面的 PolyMesher3D 中还会提及.

三维的数据结构相比二维要复杂一点, 基本数据结构可如下安排

- `node`: 按正常处理, 即存储剖分的顶点坐标.
- `elem`: 它的第 `iel` 个分量存储单元的面. 可把固定单元想象为展开的平面, 从而每个单元都类似二维问题存储.



- 类似二维问题, 下面的语句给出了单元顶点的坐标 `V` 以及顶点连通的顶点 `adjV`.

```

1 [V,~,~,adjV] = extreme(Pn{1});

```

考虑上面的图, 第 1 个顶点连接的是顶点 2,4,8, 则 `adjV` 的第 1 个元素就是数组 `[2, 4, 8]`. 它是以元胞存储的, 尽管很多时候连接的顶点个数都是 3.

- 确定单元的面, 即确定每个面包含的顶点. 在 MPT 中是比较容易的, 这得益于多面体的表示 (2.1), 即

$$\mathcal{P} = \{x \in \mathbb{R}^n : Hx \leq K\}.$$

该不等式的每行对应一个面, 为此可把顶点代入判断. `H` 和 `K` 如下获得

```

1 [H,K] = double(Pn{1});

```

面的个数为 $nf = \text{length}(K)$.

- 第 i 个面上的点的序号如下获得

```
1 indexf = find(abs(V*H(i,:)'-K(i))≤1e-6);
```

例如, 第 1 个面上的点为 [1, 4, 6, 8].

- 当然, 这些点的排列可能是乱的, 先进行循环排列. 二维问题已经给出了循环排列的做法, 只需要确定当前的面对应的 adjV , 为了区别记为 adjVf . 例如, 点 1 连接的点为 [2,4,8], 它与面上的交就是点 1 在该面上的连接点. 因 adjV 和 adjVf 都是元胞数组, 可用 cellfun 实现.

```
1 adjVf = cellfun(@(x)intersect(x,Vf),adjV(indexf),'UniformOutput',false);
```

- 需要注意的是, 二维循环排序的时候, 单元的顶点是自然序, 但对三维单元的面来说却不是, 需要一点修改.

```
1 % adjV of the current face
2 adjVf = cellfun(@(x)intersect(x,indexf), adjV(indexf), 'UniformOutput', false);
3 % cyclic order
4 nindexf = length(indexf);
5 order = zeros(1,nindexf);
6 pre_id = 1; previous = indexf(pre_id);
7 adj = adjVf{pre_id};
8 current = adj(1); cur_id = find(indexf==current);
9 order(1) = previous; order(2) = current;
10 for p = 2:nindexf-1
11     adj = adjVf{cur_id}; next = adj(adj≠previous);
12     order(p+1) = next;
13     previous = current; current = next;
14     cur_id = find(indexf==current);
15 end
```

这里的 $order$ 就是面的连通性.

- 面的顶点也进行逆序排列 (面作为 xy 平面时, 符合右手系规则, 即大拇指指向内部). 对三维问题, 可用重心与前三个顶点组成的四面体的有向体积判定.

```
1 % counterclockwise order
2 volumn = det([ones(4,1), [V(order(1:3),:); P(iel,:)]])/6;
3 if volumn<0, order = order(end:-1:1); end
4 elem{iel}{i} = order;
```

注意, 这里的 P 是此时的种子, 不一定是重心, 但它仍在单元内部, 故用 P 也可以.

综上, 按单元存储的 $node$ 和 $elem$ 如下获得

```
1 % ----- elementwise nodeElem, elem -----
2 nodeElem = cell(NT,1); elem = cell(NT,1);
3 for iel = 1:NT
4     [V,~,~,adjV] = extreme(Pn(iel));
5     [H,K] = double(Pn(iel)); nodeElem{iel} = V; nf = length(K);
6     for i = 1:nf % loop of faces
7         % find all points on the current hyperplane
8         indexf = find(abs(V*H(i,:)'-K(i))≤1e-6);
9         % adjV of the current face
```

```

10         adjVf = cellfun(@(x)intersect(x,indexf), adjV(indexf), 'UniformOutput', false);
11         % cyclic order
12         nindexf = length(indexf);
13         order = zeros(1,nindexf);
14         pre_id = 1; previous = indexf(pre_id);
15         adj = adjVf{pre_id};
16         current = adj(1); cur_id = find(indexf==current);
17         order(1) = previous; order(2) = current;
18         for p = 2:nindexf-1
19             adj = adjVf{cur_id}; next = adj(adj!=previous);
20             order(p+1) = next;
21             previous = current; current = next;
22             cur_id = find(indexf==current);
23         end
24         % counterclockwise order
25         column = det([ones(4,1), [V(order(1:3),:); P(iel,:)]])/6;
26         if column<0, order = order(end:-1:1); end
27         elem{iel}{i} = order; % natural numbers
28     end
29 end

```

注意, 每个单元的节点序号都是自然序号, 且与单元 nodeElem 的行索引对应.

注 2.5 这里设置 $1e-6$ 的精度来判断顶点是否在面上. 剖分可能出现很短的边, 对粗糙一点的精度可能出现错误. 例如, 精度改为 $1e-4$, 程序中发现有时候会出现错误 (后面获取面上顶点的连通性 adjVf 时).

2.4.2 节点的整体编号

- 先从 nodeElem 获得基本数据结构 node. 要注意, 此时 $P_n(1)$ 记录的是面, 对应的个数不再是顶点数.
-

```

1 % node
2 node = vertcat(nodeElem{:}); totalN = size(node,1);
3 totalid = (1:totalN)';
4 for i = 1:totalN
5     for j = i+1:totalN
6         p1 = node(i,:); p2 = node(j,:);
7         if norm(p1-p2)≤1e-4 && totalid(j)>i
8             totalid(j) = i;
9         end
10    end
11 end
12 [id, ~, totalJ] = unique(totalid, 'rows');
13 node = node(id,:);
14 elemLen = cellfun('length',nodeElem);
15 indexElem = mat2cell(totalJ',1,elemLen)';

```

这里, indexElem 对应 nodeElem, 给出单元顶点的整体编号.

- 有了 indexElem, 我们就可给出新的 elem.
-

```

1 % ----- elem -----
2 for iel = 1:NT
3     index = indexElem{iel};
4     elemf = elem{iel}; nf = length(elemf);

```

```

5      for i = 1:nf
6          indexf = elemf{i};
7          elemf{i} = index(indexf);
8      end
9      elem{iel} = elemf;
10 end

```

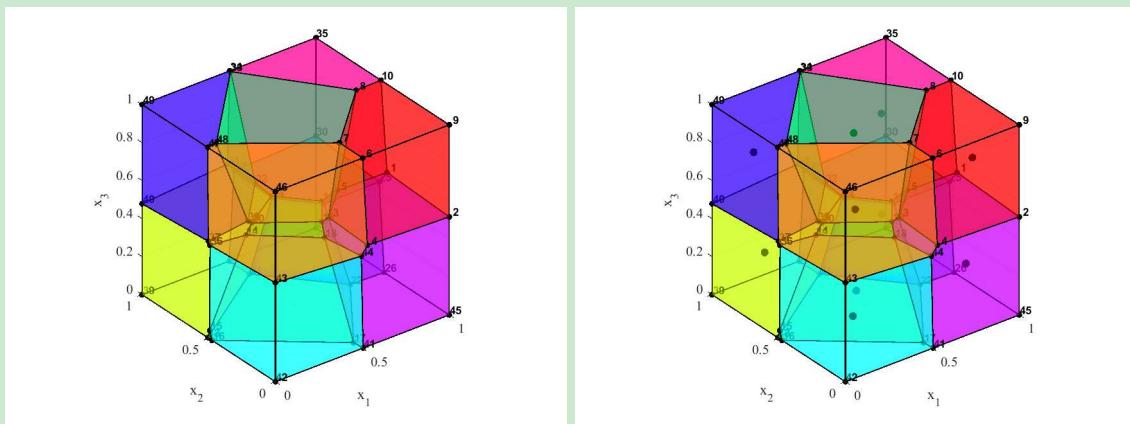
或使用 cellfun 简化程序

```

1 % ----- elem -----
2 for iel = 1:NT
3     index = indexElem{iel};
4     elemf = cellfun(@(ic) index(ic),elem{iel}), 'UniformOutput',false);
5     elem{iel} = elemf;
6 end

```

可如下画图



```

1 Options.shade = 0.5;
2 figure, mpt_plot(Pn,Options); axis equal
3 hold on
4 plot3(P(:,1),P(:,2),P(:,3),'k.', 'MarkerSize', 20);

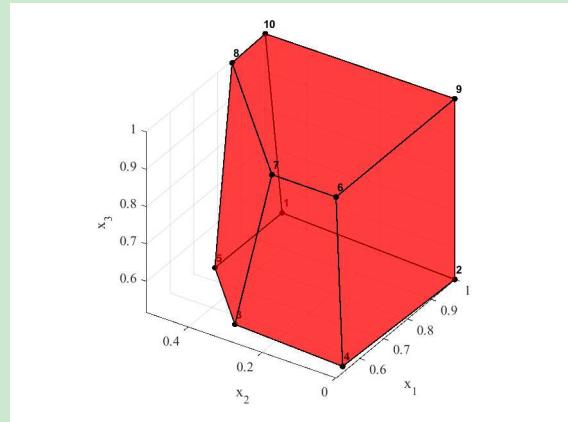
```

也可如下查看具体单元的情况 (如检查每个面的编号)

```

1 iel = 2;
2 figure, mpt_plot(Pn(iel),Options); axis equal
3 elemf = elem{iel};
4 range = unique(horzcat(elemf{:}))';
5 findnode(node,range)

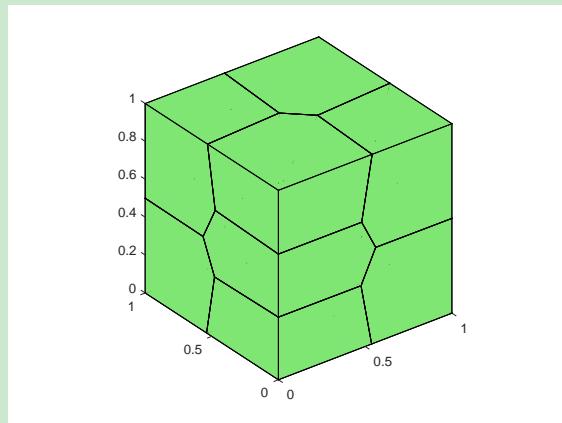
```



注 2.6 网格的图示也可使用 showmesh.m 实现, 为了匹配这里的数据结构, 我们做了一点修改. 调用如下

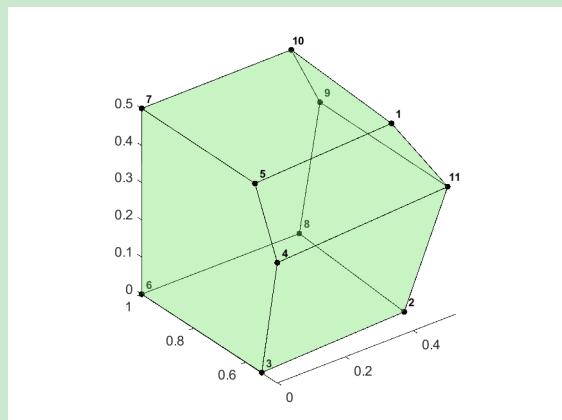
```
1 option.FaceAlpha = 1;
2 figure, showmesh(node,elem,option); axis equal
```

这里, FaceAlpha 设置透明度, 默认取 0.4. 上面取为 1, 整个剖分只能看到边界, 如下图所示.



具体单元可如下查看.

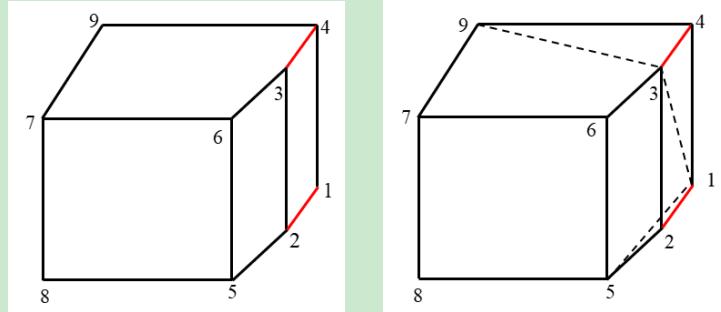
```
1 iel = 1; elemf = elem{iel};
2 option.FaceAlpha = 0.25;
3 figure, showmesh(node,elemf,option); axis equal
4 range = unique(horzcat(elemf{:}))';
5 findnode(node,range)
```



2.4.3 短边的去除

三维短边去除的共面问题

与二维不同的是，三维剖分不能轻易去除短边，否则会影响面顶点的共面性。



例如，设左图的两条红色边为短边，且保留起点编号，则上侧的面变为 9-3-6-7，右侧的两个面合为 1-3-6-5。容易发现，点 1,3,6,5 并不在一个面上。要想给出正确的面，必须对新的单元顶点重新生成凸包，然后重复前面的过程。

顶点的替换

短边的去除与二维情形类似，只要把三维的面想象成分布在二维平面上。

- 短边如下找到。

```

1 totalfaces = vertcat(elem{:});
2 T1 = cellfun(@(verts) [verts(2:end),verts(1)], totalfaces, 'UniformOutput', ...
    false);
3 v0 = horzcat(totalfaces{:})'; v1 = horzcat(T1{:})';
4 totalEdge = sort([v0,v1],2);
5 [i,j] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
6 edge = [j,i];
7 z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
8 he = sqrt(sum((z1-z2).^2,2));
9 ir = find(he < 0.1*max(he));
10 nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices

```

- 对具体单元的所有面 elemf，类似二维用替换端点的方式去除短边。如下

```

1 elemf = elem{iel};
2     [id,~,totalid] = unique(horzcat(elemf{:})');
3     totalid = id(totalid);
4     for i = 1:length(nv2)
5         v1 = nv1(i); v2 = nv2(i); iv = ir(i);
6         ax = min(abs([z1(iv,1)-BdBox(1), BdBox(2)-z1(iv,1)]));
7         ay = min(abs([z1(iv,2)-BdBox(3), BdBox(4)-z1(iv,2)]));
8         az = min(abs([z1(iv,3)-BdBox(5), BdBox(6)-z1(iv,3)]));
9         bx = min(abs([z2(iv,1)-BdBox(1), BdBox(2)-z2(iv,1)]));
10        by = min(abs([z2(iv,2)-BdBox(3), BdBox(4)-z2(iv,2)]));
11        bz = min(abs([z2(iv,3)-BdBox(5), BdBox(6)-z2(iv,3)]));
12        if (ax+ay+az)<(bx+by+bz)
13            totalid(totalid==v2) = v1; % replaced by starting indices
14            nv1(nv1==v2) = v1;
15            nv2(nv2==v2) = v1;

```

```

16     else
17         totalid(totalid==v1) = v2;
18         nv1(nv1==v1) = v2;
19         nv2(nv2==v1) = v2;
20     end
21 end
22 elemfLen = cellfun('length',elemf);
23 elemf = mat2cell(totalid', 1, elemfLen)';

```

- 接着再恢复数据结构, 删除重复点即可.

```

1 elemfLen = cellfun('length',elemf);
2 elemf = mat2cell(totalid', 1, elemfLen)';
3 for i = 1:length(elemf)
4     index = elemf{i};
5     [~,i1] = unique(index);
6     elemf{i} = index(sort(i1));
7 end
8 [id,~,totalid] = unique(horzcat(elemf{:})');
9 nodeElem{iel} = node(id,:);
10 elemfLen = cellfun('length',elemf);
11 elemf = mat2cell(totalid', 1, elemfLen)'; % 单元自然序号

```

要注意, 此时每个单元都是按照自然顺序给出的连通性, 且相应的单元顶点存储在 `nodeElem` 中. 当然, 我们直接恢复为整体编号, 但考虑到去除短边后还要重新编号, 这里就不再这样做.

新的 node

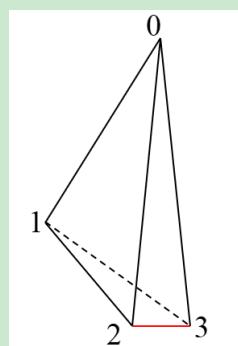
```

1 node = vertcat(nodeElem{:}); totalN = size(node,1);
2 totalid = (1:totalN)';
3 for i = 1:totalN
4     for j = i+1:totalN
5         p1 = node(i,:); p2 = node(j,:);
6         if norm(p1-p2)≤1e-6 && totalid(j)>i
7             totalid(j) = i;
8         end
9     end
10 end
11 [id, ~, totalJ] = unique(totalid, 'rows');
12 node = node(id,:);

```

新的 elem

类似二维问题, NT 个单元可能存在四面体, 而且含有一个短边, 如下图所示.



短边顶点替换后, 该单元退化为面, 成为无用单元. 同样地, 这种情形也很难出现, 为此程序中不处理这个问题. 但要注意的是, 单元的面中可能出现退化面, 如有一个三角形而且含有短边. 程序中只要保留顶点数大于或等于 3 的面即可.

```
1 % elem
2 elemLen = cellfun('length',nodeElem);
3 indexElem = mat2cell(totalJ',1,elemLen)';
4 for iel = 1:NT
5     index = indexElem{iel};
6     elemf = cellfun(@(ic) index(ic),elem{iel}, 'UniformOutput',false);
7     elemfLen = cellfun('length',elemf);
8     elemf = elemf(elemfLen≥3); % 去掉不构成面的
9     elem{iel} = elemf';
10 end
```

新的 Pn

```
1 for iel = 1:NT
2     Pn(iel) = polytope(nodeElem{iel});
3 end
```

恢复共面性

只需要重复去除短边前的过程, 略.

2.4.4 程序整理

在 tool/MPT 文件夹中给出了 mpt_meshfun3D.m 用以生成三维多面体网格, 相应的调用函数为 MPT_Mesher3D.m, 这里不再单独列出.

3 PolyMesher: 二维多角形剖分的生成

待整理

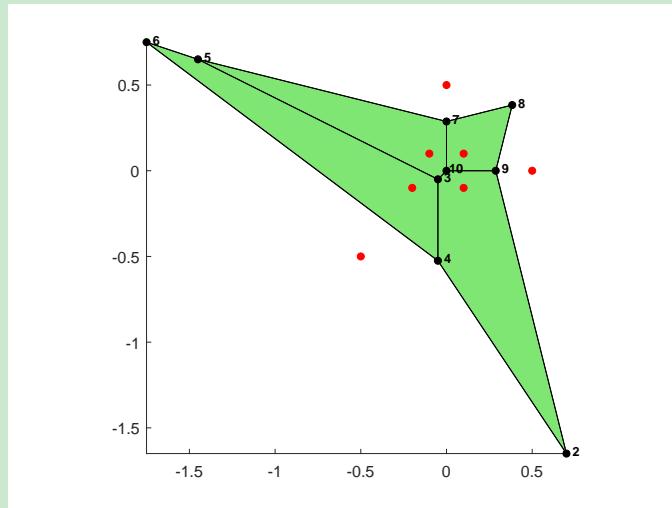
PolyMesher 是生成二维多角形剖分的工具, 它采用 (边界附近) 种子的反射集构造边界的近似.

3.1 Voronoi 图

MATLAB 自带相关函数用以生成 Voronoi 图.

```
1 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];
2 [node, elem] = voronoin(P);
3 showmesh(node, elem); findnode(node); hold on
4 plot(P(:,1),P(:,2), 'r.', 'MarkerSize', 15)
```

图形如下



返回的 `node` 的第一个点的坐标都是 (`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.

给定一个凸区域, 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.

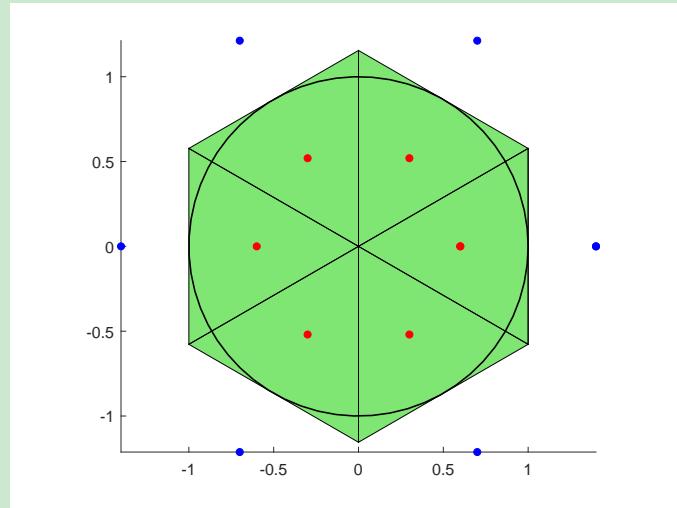
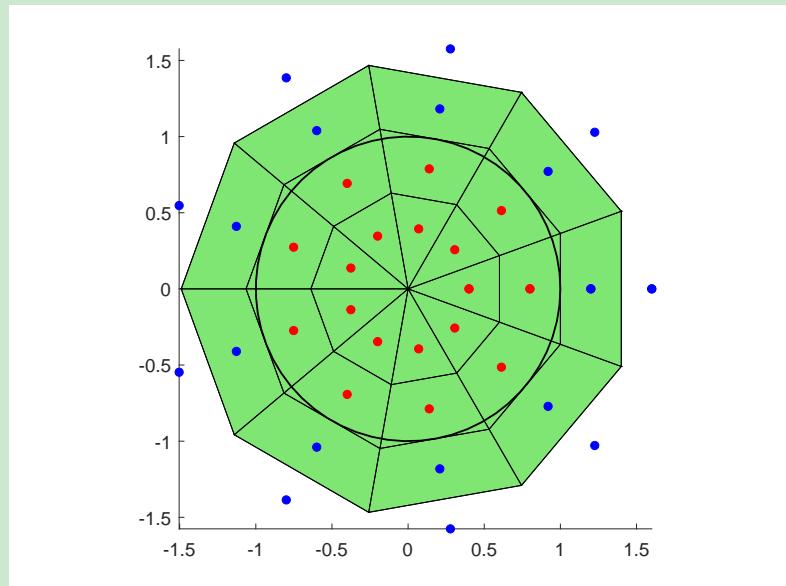


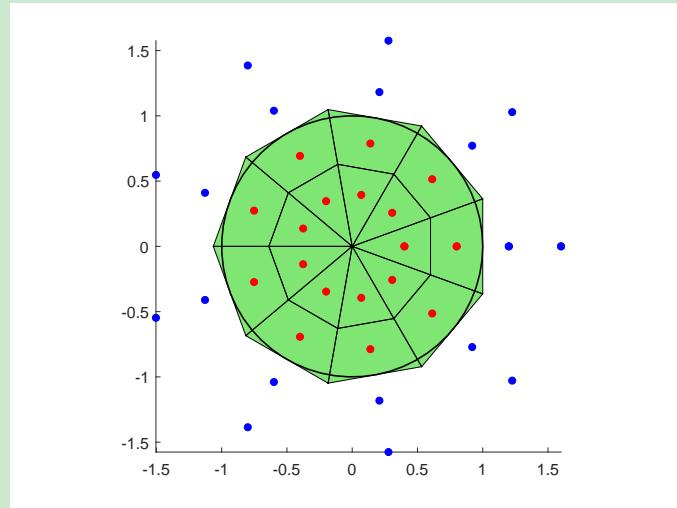
图 9. $\mathcal{T}(\tilde{P}; \Delta)$

如图 9 所示, 凸区域 Δ 为圆. 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:

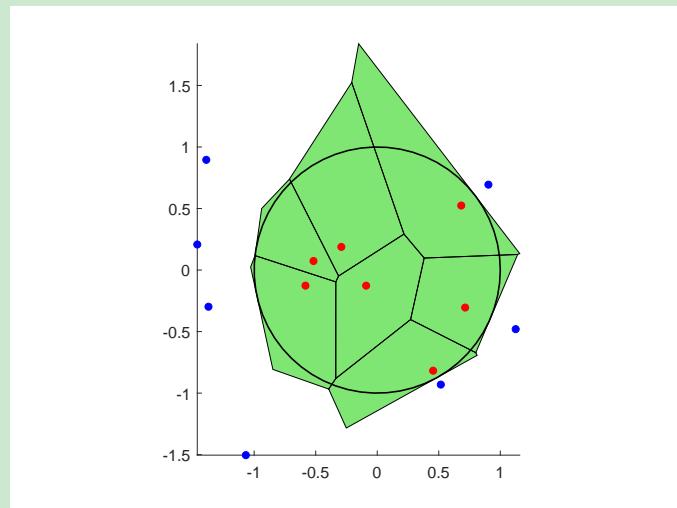
- 反射点 $R(y)$ 的 Voronoi 单元在区域外, 它与种子 y 的单元有公共边界, 且与凸区域的边界相切.
- $\mathcal{T}(\tilde{P}; \Delta)$ 有 $2 * NT$ 个单元, 前 NT 个对应区域内的种子, 因而是该区域的一个覆盖.
- 当凸区域内点充分多时, 这个覆盖就是一个较好的剖分图.
- 注意上面给出的图外部的单元恰好是空白的, 它们与无穷原点构成“多角形”. 但外部也可能存在可以直接观察到的剖分, 如下图



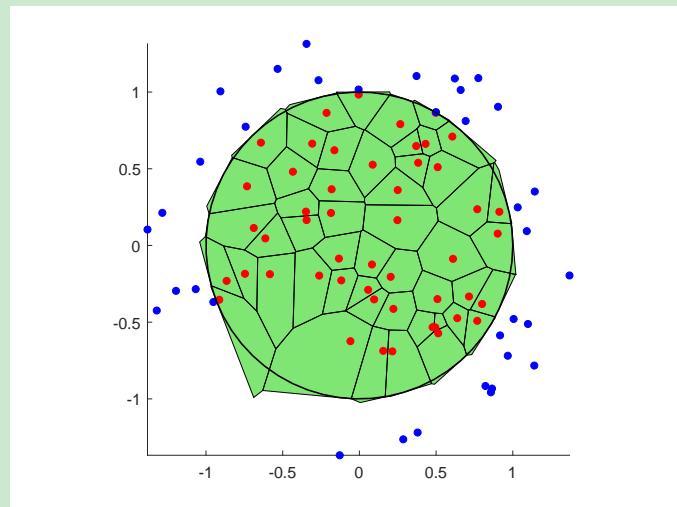
取前 NT 个的结果如下



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, 就要采用前面所述的 CVTs.

3.2 反射集的计算

3.2.1 符号距离函数

定义 3.1 设 $\Omega \subset \mathbb{R}^2$, 定义 $d_\Omega : \mathbb{R}^2 \rightarrow \mathbb{R}$ 为

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

其中,

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

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

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

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 $\nabla d_\Omega(x) = \text{sgn}(x - x_0)$. 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

$$R_\Omega(x) = x - 2d_\Omega(x)\nabla d_\Omega(x). \quad (3.3)$$

对复合区域, 通常有

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

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

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

3.2.2 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为 $d_\Omega(x)$. 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

```

1 function x = Circle_Domain(Demand,P)
2 BdBox = [-1 1 -1 1];
3 switch(Demand)
4     case('Dist'); x = DistFnc(P);
5     case('BdBox'); x = BdBox;
6 end
7 %----- the signed distance function -----
8 function Dist = DistFnc(P,BdBox)

```

```

9 Dist = dCircle(P,0,0,1);
10
11 function d = dCircle(P,xc,yc,r)
12 d = sqrt((P(:,1)-xc).^2+(P(:,2)-yc).^2)-r;
13 d = [d,d];

```

这里 $d = [d, d]$ 复制了一次 d , 是因为后面要考虑分段边界. 此时每个边界都对应一个符号距离, d 的最后一列对应整体符号距离. 为了一般性, 这里复制了一次.

随机种子如下生成

```

1 % ----- Generate random pointset -----
2 Domain = @Circle_Domain;
3 NT = 30;
4 % ----- Generate random pointset -----
5 P = zeros(NT,2); s = 0;
6 while s < NT
7     xy(:,1) = (BdBox(2)-BdBox(1))*rand(NT,1)+BdBox(1);
8     xy(:,2) = (BdBox(4)-BdBox(3))*rand(NT,1)+BdBox(3);
9     d = Domain('Dist',xy); d = d(:,end);
10    I = find(d<0); % index of seeds inside the domain
11    NumAdded = min(NT-s,length(I)); % number of seeds that can be added
12    P(s+1:s+NumAdded,:) = xy(I(1:NumAdded),:);
13    s = s+NumAdded;
14 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, 直接去掉. 称这种初始种子为均匀分布的, 如下获取

```

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

```

以上过程编写成函数文件

```

1 function P = PolyMesher_init_Pointset(Domain,varargin)
2
3 BdBox = Domain('BdBox');
4 nvar = length(varargin);
5
6 % ----- Generate random pointset -----
7 if nvar==1
8     NT = varargin{1}; P = zeros(NT,2); s = 0;
9     while s < NT
10         xy(:,1) = (BdBox(2)-BdBox(1))*rand(NT,1)+BdBox(1);
11         xy(:,2) = (BdBox(4)-BdBox(3))*rand(NT,1)+BdBox(3);
12         d = Domain('Dist',xy);
13         I = find(d(:,end)<0); % index of seeds inside the domain
14         NumAdded = min(NT-s,length(I)); % number of seeds that can be added
15         P(s+1:s+NumAdded,:) = xy(I(1:NumAdded),:);
16         s = s+NumAdded;
17     end
18     return;
19 end

```

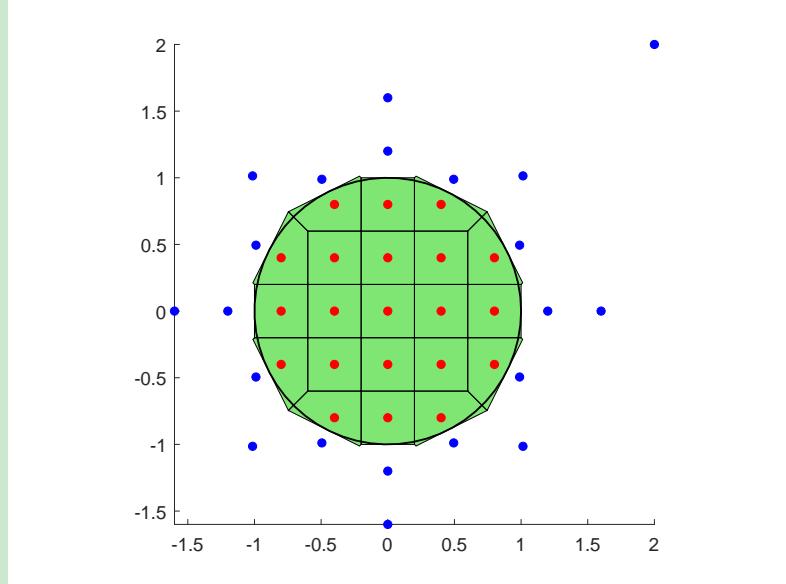
```

20
21 % ----- Generate uniform pointset -----
22 % if nvar==2
23 Nx = varargin{1}; Ny = varargin{2};
24 x = linspace(BdBox(1),BdBox(2),Nx+1)';
25 y = linspace(BdBox(3),BdBox(4),Ny+1)';
26 xc = (x(1:end-1)+x(2:end))/2; yc = (y(1:end-1)+y(2:end))/2;
27 [X,Y] = ndgrid(xc,yc); P = [X(:),Y(:)];
28 d = Domain('Dist',P); P = P(d(:,end)<0,:);
29 % end

```

3.2.3 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 (3.3) 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

$$d_\Omega(y) < \alpha(n, \Omega) \quad (3.4)$$

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

```

1 clc; clear; close all
2 % ----- Generate intial pointset -----
3 Domain = @Circle_Domain;
4 % NT = 50;
5 % P = PolyMesher_init_Pointset(Domain,NT);

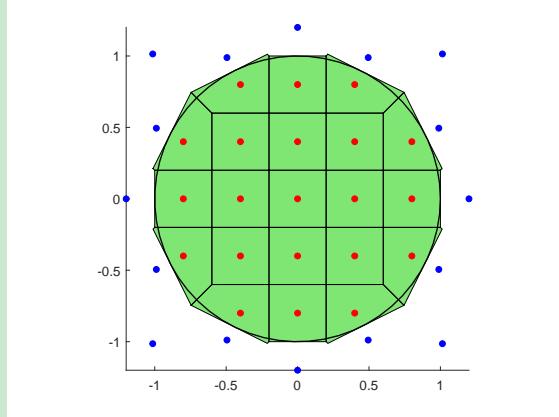
```

```

6 Nx = 5; Ny = 5;
7 P = PolyMesher_init_Pointset(Domain,Nx,Ny);
8 NT = size(P,1);
9
10 % ----- Compute the reflection pointset -----
11 eps = 1e-8; c = 1.1;
12 BdBox = Domain('BdBox');
13 Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
14 Alpha = c*sqrt(Area/NT);
15
16 d = Domain('Dist',P);
17 I = abs(d)<Alpha; % logical index of seeds near the bdry
18 n1 = (Domain('Dist',P+[eps,0])-d)/eps;
19 n2 = (Domain('Dist',P+[0,eps])-d)/eps;
20 R_P = P(I,:)-2*[n1(I),n2(I)].*d(I);
21
22 % -----
23 PP = [P; R_P];
24 [node,elem] = voronoin(PP);
25 elem = elem(1:NT);
26 showmesh(node,elem); %findnode(node);
27 hold on
28 plot(P(:,1),P(:,2),'r.',R_P(:,1),R_P(:,2),'b.','MarkerSize',15)
29 t = linspace(0,2*pi,60)';
30 plot(cos(t),sin(t),'k-','LineWidth',1)

```

取 $c = 1.1$, 结果如下



注意, Area 在 Lloyd 迭代过程中不断更新, 它会逐渐接近区域面积 $|\Omega|$.

3.2.4 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

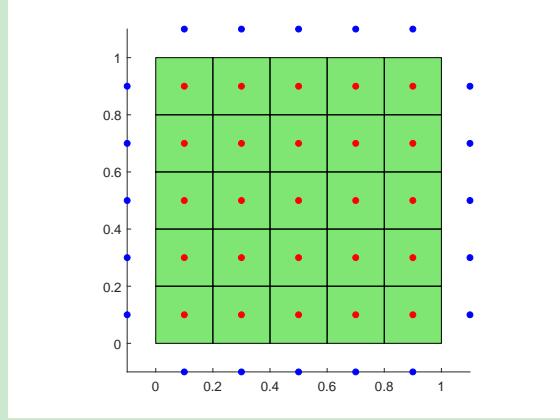
$$d_1(\mathbf{x}) = a_1 - \mathbf{x}(1),$$

类似下侧、右侧和上侧直线的符号距离函数分别为

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

整体区域是这些半平面的交, 从而

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



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 (3.4)). 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

```

1 function x = Rectangle_Domain(Demand,P)
2   BdBox = [0 1 0 1];
3   switch(Demand)
4     case('Dist'); x = DistFnc(P,BdBox);
5     case('BdBox'); x = BdBox;
6   end
7 %----- the signed distance function -----
8 function Dist = DistFnc(P,BdBox)
9   Dist = dRectangle(P,BdBox(1),BdBox(2),BdBox(3),BdBox(4));
10
11 function d = dRectangle(P,x1,x2,y1,y2)
12   d = [x1-P(:,1), P(:,1)-x2, y1-P(:,2), P(:,2)-y2];
13   d = [d,max(d,[],2)];

```

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

如下生成初始剖分

```

1 clc;clear;close all
2 % ----- Generate intial pointset -----
3 Domain = @Rectangle_Domain;
4 Nx = 5; Ny = 5;
5 P = PolyMesher_init_Pointset(Domain,Nx,Ny);
6 NT = size(P,1);
7
8 % ----- Compute the reflection pointset -----
9 eps = 1e-8; eta = 0.9; c = 1.5;
10 BdBox = Domain('BdBox');
11 Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
12 Alpha = c*sqrt(Area/NT);
13
14 d = Domain('Dist',P);
15 NBdrySegs = size(d,2)-1; %Number of constituent bdry segments
16 n1 = (Domain('Dist',P+[eps,0])-d)/eps;

```

```

17 n2 = (Domain('Dist', P+[0, eps])-d)/eps;
18 I = abs(d(:,1:NbdrySegs))<Alpha; %Logical index of seeds near the bdry
19 P1 = repmat(P(:,1), 1, NbdrySegs); %[NT x NbdrySegs] extension of P(:,1)
20 P2 = repmat(P(:,2), 1, NbdrySegs); %[NT x NbdrySegs] extension of P(:,2)
21 R_P = [P1(I), P2(I)] - 2*[n1(I), n2(I)].*d(I);
22
23 % ----- Voronoi -----
24 PP = [P; R_P];
25 [node, elem] = voronoin(PP, {'Qbb', 'Qz'});
26 elem = elem(1:NT);
27 showmesh(node, elem); %findnode(node);
28 hold on
29 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.

3.2.5 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的情形, 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

```

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

```

3.2.6 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

```

1 function R_P = PolyMesher_Reflect(P, Domain)
2
3 % ----- Compute the reflection pointset -----
4 eps = 1e-8; eta = 0.9; c = 1.5; NT = size(P, 1);
5 BdBox = Domain('BdBox');
6 Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
7 Alpha = c*sqrt(Area/NT);
8
9 d = Domain('Dist', P);

```

```

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

```

3.3 Lloyd 迭代

有了反射集 $R(P)$ 后, 如下获得 $P \cup R(P)$ 的单元划分

```

1 % Construct the Voronoi diagram
2 [node,elem] = voronoin([P;R_P],{'Qbb','Qz'});

```

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.

```

1 function [Pc,Area,Err] = PolyMesher_VoroCentroid(P,node,elem)
2 % Compute the centroid of Voronoi cell
3 NT = size(P,1);
4 Pc = zeros(NT,2); A = zeros(NT,1);
5 for iel = 1:NT
6     vx = node(elem{iel},1); vy = node(elem{iel},2); nv = length(elem{iel});
7     vxS = vx([2:nv,1]); vyS = vy([2:nv,1]);
8     temp = vx.*vyS - vy.*vxS;
9     A(iel) = 0.5*sum(temp);
10    Pc(iel,:) = 1/(6*A(iel))*[sum((vx+vxS).*temp),sum((vy+vyS).*temp)];
11 end
12 Area = sum(abs(A));
13 Err = sqrt(sum((A.^2).*sum((Pc-P).^2,2)))*NT/Area^1.5;

```

3.4 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

```

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

```

这里,

- `id` 是网格剖分中原始节点的编号, 其行索引作为新的编号, 称为自然序号.
- `totalid` 按 `elem` 的拉直模式记录每个节点的自然序号.

- 显然新的 `node` 为

```

1 node = node(id,:);

```

把 `totalid` 恢复为 `elem` 的模式就获得新的 `elem`

```

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

```

- 要注意, voronoin 函数给出的单元节点顺序不一定是逆时针的(是循环的), 可通过前三个顶点构成的三角形的有向面积进行判断.

```

1 for iel = 1:NT
2     index = elem{iel};
3     z1 = node(index(:,1),:);
4     z2 = node(index(:,2),:);
5     z3 = node(index(:,3),:);
6     e2 = z3-z1; e3 = z1-z2;
7     area = 0.5*(-e3(:,1).*e2(:,2)+e3(:,2).*e2(:,1));
8     if area<0, elem{iel} = index(end:-1:1); end
9 end

```

上面通过 elem 恢复自然序号的过程可总结为如下函数

```

1 function [node,elem] = PolyMesher_Reorder(NT,node,elem)
2 elem = elem(1:NT);
3 [id,~,totalid] = unique(horzcat(elem{:})');
4 node = node(id,:,:);
5 elemLen = cellfun('length',elem);
6 elem = mat2cell(totalid', 1, elemLen)';
7 for iel = 1:NT
8     index = elem{iel};
9     z1 = node(index(:,1),:);
10    z2 = node(index(:,2),:);
11    z3 = node(index(:,3),:);
12    e2 = z3-z1; e3 = z1-z2;
13    area = 0.5*(-e3(:,1).*e2(:,2)+e3(:,2).*e2(:,1));
14    if area<0, elem{iel} = index(end:-1:1); end
15 end

```

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, 这是由一些极短的边造成的. 在 MPT 中已经介绍了去除短边的方法, 这里再重复一下.

- 根据辅助数据结构的讨论过程, 我们可获得所有的一维边集合 edge

```

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

```

- 我们将删除如下的边 (PolyMesher 不是这样做的)

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

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.

```

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

```

这里 $nv1$, $nv2$ 是要删除的边的起点和终点编号.

- 短边可如下删除, 即把要删除边的终点编号替换为起点编号 (即保留起点编号)

```
1 [id,~,totalid] = unique(horzcat(elem{:})');
2 totalid = id(totalid);
3 for i = 1:length(nv2)
4     v1 = nv1(i); v2 = nv2(i);
5     totalid(totalid==v2) = v1;
6     nv1(nv1==v2) = v1; nv2(nv2==v2) = v1;
7 end
8 elemLen = cellfun('length',elem);
9 elem = mat2cell(totalid', 1, elemLen');
```

与 MPT 不同的是, 我们没有加上保留起点还是终点的语句, 计算结果表明加不加上没有影响.

- 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 (显然不会改变原来的节点顺序)

```
1 for iel = 1:NT
2     index = elem{iel};
3     [~,i1] = unique(index);
4     elem{iel} = index(sort(i1));
5 end
```

再次使用 PolyMesher_Reorder.m 恢复顺序即可.

上面的说明总结为如下函数

```
1 function [node,elem] = PolyMesher_rm_smalledge(NT,node,elem)
2 [node,elem] = PolyMesher_Reorder(NT,node,elem);
3 T1 = cellfun(@(verts) [verts(2:end),verts(1)], elem, 'UniformOutput', false);
4 v0 = horzcat(elem{:})'; v1 = horzcat(T1{:})';
5 totalEdge = sort([v0,v1],2);
6 [i,j] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
7 edge = [j,i];
8 z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
9 he = sqrt(sum((z1-z2).^2,2));
10 ir = find(he < 0.1*max(he));
11 nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices
12 [id,~,totalid] = unique(horzcat(elem{:})');
13 totalid = id(totalid);
14 for i = 1:length(nv2)
15     v1 = nv1(i); v2 = nv2(i);
16     totalid(totalid==v2) = v1;
17     nv1(nv1==v2) = v1; nv2(nv2==v2) = v1;
18 end
19 elemLen = cellfun('length',elem);
20 elem = mat2cell(totalid', 1, elemLen)';
21
22 for iel = 1:NT
23     index = elem{iel};
24     [~,i1] = unique(index);
25     elem{iel} = index(sort(i1));
26 end
27 [node,elem] = PolyMesher_Reorder(NT,node,elem);
```

3.5 程序整理

在 tool/PolyMesherModified 文件夹中给出了 meshfun.m 函数, 如下

```
1 %Meshfun: generate basic data structure (node,elem) representing meshes
2 % Copyright (C) Terence Yue Yu.
3
4 % ----- Usage -----
5 % Preserved as meshdata.mat
6 % load('meshdata.mat') to load basic data structure: node, elem
7 %
8
9 clc;clear;close all
10
11 % ----- Choice of the domain -----
12 % Options: Rectangle_Domain, Circle_Domain, Upper_Circle_Domain,
13 % Upper_Circle_Circle_Domain, Rectangle_Circle_Domain, Circle_Circle_Domain
14 % Michell_Domain, Horn_Domain, Suspension_Domain, Wrench_Domain
15 Domain = @Rectangle_Domain;
16 MaxIter = 100;
17 NT = 100;
18 [node,elem]= PolyMesher(Domain,MaxIter,NT);
19
20 % ----- Plot the mesh -----
21 showmesh(node,elem);
22 % findnode(node); % findelem(node,elem)
23
24 % % ----- Save the mesh data -----
25 % save meshdata node elem
```

有两种方式调用:

- 随机生成种子: [node,elem]= PolyMesher(Domain,MaxIter,NT);
- 均匀生成种子: [node,elem]= PolyMesher(Domain,MaxIter,Nx,Ny);

函数 PolyMesher.m 如下

```
1 function [node,elem] = PolyMesher(Domain,MaxIter,varargin)
2
3 % ----- Generate intial pointset -----
4 nvar = length(varargin);
5 switch nvar
6   case 1
7     NT = varargin{1};
8     P = PolyMesher_init_Pointset(Domain,NT);
9   case 2
10    Nx = varargin{1}; Ny = varargin{2};
11    P = PolyMesher_init_Pointset(Domain,Nx,Ny);
12    NT = size(P,1);
13 end
14
15 % ----- Lloyd's iteration -----
16 Iter = 0; Err = 1; Tol = 5e-6;
17 BdBox = Domain('BdBox');
18 Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
19 while(Iter<=MaxIter && Err>Tol)
20   % Compute the reflection pointset
21   R_P = PolyMesher_Reflect(P,Domain,Area);
```

```

22 % Construct the Voronoi diagram
23 [node,elem] = voronoin([P;R_P],{'Qbb','Qz'});
24 % Compute the centroid of Voronoi cell (Lloyd's update)
25 [P,Area,Err] = PolyMesher_VoroCentroid(P,node,elem);
26 Iter = Iter+1;
27 end
28
29 % ----- Remove small edges to obtain (node,elem) -----
30 [node,elem] = PolyMesher_rm_smalledge(NT,node,elem);

```

以上所有程序放置在 tool 的子文件夹 PolyMesherModified 中, 其中包含了一些区域和符号距离函数的定义, 见子文件夹 Domain 和 DistFnc.

注 3.1 对随机种子, 当点数较少时可能出现错误剖分或 voronoin 函数提示某些错误. 此时必须增加点数, 通常点数大于 20 就不会出现问题.

4 PolyMesher3D: 三维多面体剖分的生成

待优化和整理

本节考虑用 PolyMesher 工具箱的思想实现三维区域的多面体剖分 (这里的“思想”指的是用反射集来构造边界的近似).

4.1 网格的生成

4.1.1 三维 Voronoi 图

给定一些三维种子 P , voronoin.m 函数也可生成三维空间的 Voronoi 划分

```
1 [V,C] = voronoin(P,{['Qbb','Qz']});
```

这里,

- V 为所有的顶点坐标, 即 node;
- C 是元胞数组, 每个元胞存储一个单元所有顶点的整体编号. 注意前面规定的三维 elem, 每个元胞存储一个单元的所有面的连通性信息.

由单元顶点集合可获得单元信息, 即取凸包 (注意 Voronoi 单元是凸的), 如

```
1 iel = 1; X = V(C{iel},:); Tri = convhulln(X);
```

- MPT 用 polytope(X) 给出多面体的不等式表示信息: $Hx \leq K$.
- MATLAB 给出的 Tri 是单元面的三角剖分的连通性信息。
- 注意, Tri 中的序号是 X 的行索引给出的局部编号, 而整体编号由 C 获得.

现在我们来观察一下下面的三角剖分. 为了避免无界, 我们对随机种子进行反射, 这与二维类似, 如下

```
1 function R_P = PolyMesher_Reflect3D(P,Domain,volumn)
2
3 % ----- Compute the reflection pointset -----
4 eps = 1e-8; eta = 0.9; c = 1.5; NT = size(P,1);
5 Alpha = c*(volumn/NT)^(1/3);
6
7 d = Domain('Dist',P);
8 NBdrySegs = size(d,2)-1; %Number of constituent bdry segments
9 n1 = (Domain('Dist',P+repmat([eps,0,0],NT,1))-d)/eps;
10 n2 = (Domain('Dist',P+repmat([0,eps,0],NT,1))-d)/eps;
11 n3 = (Domain('Dist',P+repmat([0,0,eps],NT,1))-d)/eps;
12 I = abs(d(:,1:NBdrySegs))<Alpha; %Logical index of seeds near the bdry
13 P1 = repmat(P(:,1),1,NBdrySegs); %[NT x NBdrySegs] extension of P(:,1)
14 P2 = repmat(P(:,2),1,NBdrySegs); %[NT x NBdrySegs] extension of P(:,2)
15 P3 = repmat(P(:,3),1,NBdrySegs); %[NT x NBdrySegs] extension of P(:,3)
16 R_P = [P1(I), P2(I), P3(I)] - 2*[n1(I), n2(I), n3(I)].*repmat(d(I),1,3); %%
17 d_R_P = Domain('Dist',R_P);
18 J = d_R_P(:,end)>0 & abs(d_R_P(:,end))>eta*abs(d(I));
19 R_P = R_P(J,:);
20 R_P = unique(R_P,'rows');
```

考虑方体区域 $[0, 1]^3$, 区域函数为 Cube_Domain.m, 相应的符号距离函数为

```

1 function d = dCube(P,BdBox)
2 x1 = BdBox(1); x2 = BdBox(2);
3 y1 = BdBox(3); y2 = BdBox(4);
4 z1 = BdBox(5); z2 = BdBox(6);
5 d = [x1-P(:,1), P(:,1)-x2, y1-P(:,2), P(:,2)-y2, z1-P(:,3), P(:,3)-z2];
6 d = [d,max(d,[],2)];

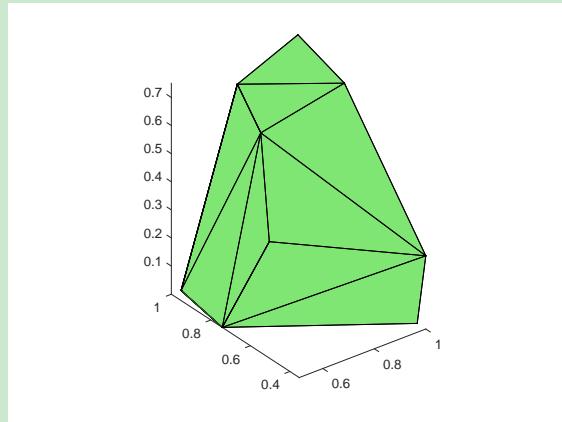
```

这样, 可如下生成特定单元面的三角剖分.

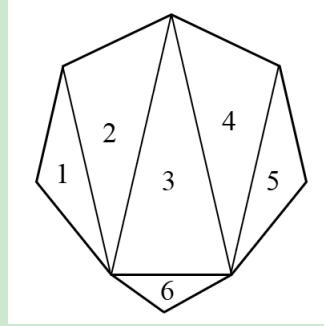
```

1 Domain = @Cube_Domain;
2 NT = 10;
3 % Generate intial pointset
4 P = PolyMesher_init_Pointset3D(Domain,NT);
5 % Compute the reflection pointset
6 BdBox = Domain('BdBox');
7 volume = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3))*(BdBox(6)-BdBox(5));
8 R_P = PolyMesher_Reflect3D(P,Domain,volume);
9 % Construct the Voronoi diagram
10 [V,C] = voronoin([P;R_P],{'Qbb','Qz'});
11 % Generate a particular cell of the Voronoi diagram
12 iel = 1;
13 X = V(C{iel},:);
14 Tri = convhulln(X);
15 % Show the cell
16 option.FaceAlpha = 1;
17 figure,showmesh(X,Tri,option); axis equal

```



- `convhulln.m` 给出的面的三角剖分不是任意的, 三角形的顶点仍是面的顶点 (否则就很难获得拓扑信息).
- 观察生成的单元, 似乎共面的三角形都是从一个顶点出发连接其他顶点给出的. 若确是如此, 则共面三角形都有公共顶点.
- 考虑到一般性, 以下认为共面的三角形可以无公共点, 从而有三种可能情形, 如下图所示.



这里, 三角形 1,2 有两个公共点, 即有公共边; 三角形 1,3 有一个公共点; 三角形 1,4 无公共点. 注意, 一个三角形最多有三个相邻三角形.

4.1.2 Lloyd 迭代

区域的重心可用微积分公式计算, 而且由 Gauss 公式, 体上的积分转化为面的积分. 正因为如此, 对面进行三角剖分后, 可在面上逐个三角形计算积分. 相应的函数为 Polycentroid.m, 其用法如下

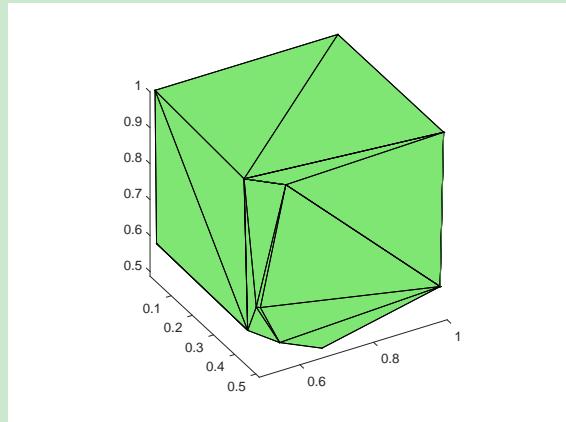
```
1 centroid = Polycentroid(V,Tri);
```

这里, 输入的 V 是单元的顶点, Tri 是面三角剖分的连通性信息, 而输出的 $centroid$ 是单元的重心 (这个算法对非凸区域也试用).

Lloyd 迭代如下实现

```
1 Domain = @Cube_Domain;
2 NT = 10; MaxIter = 400;
3 % Generate initial pointset
4 P = PolyMesher_init_Pointset3D(Domain,NT);
5 BdBox = Domain('BdBox');
6 volume = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3))*(BdBox(6)-BdBox(5));
7 iter = 1;
8 while(iter<=MaxIter)
9     % Compute the reflection pointset
10    R_P = PolyMesher_Reflect3D(P,Domain,volume);
11    % Construct the Voronoi diagram
12    [V,C] = voronoin([P;R_P],{'Qbb','Qz'});
13    P = zeros(NT,3); volume = 0;
14    for iel = 1:NT
15        X = V(C{iel},:); [Tri,vol] = convhulln(X);
16        P(iel,:) = Polycentroid(X,Tri); volume = volume+vol;
17    end
18    iter = iter+1;
19 end
```

注意, 多次迭代 Voronoi 图的质量变好, 但是面上会出现特别窄的三角形, 如下图所示.

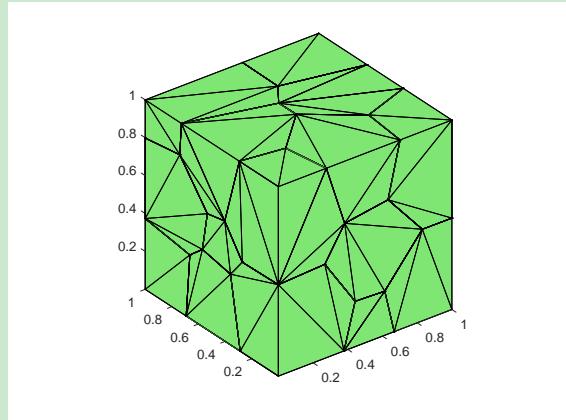


把所有三角形存在一起, 可绘制整体图像, 如下

```

1 for iel = 1:NT
2     index = C{iel}; % global indices of element
3     X = V(index,:); Tri = convhulln(X);
4     elemTri{iel} = index(Tri);
5 end
6 totalTri = vertcat(elemTri{:});
7 option.FaceAlpha = 1;
8 figure, showmesh(V,totalTri,option); axis equal

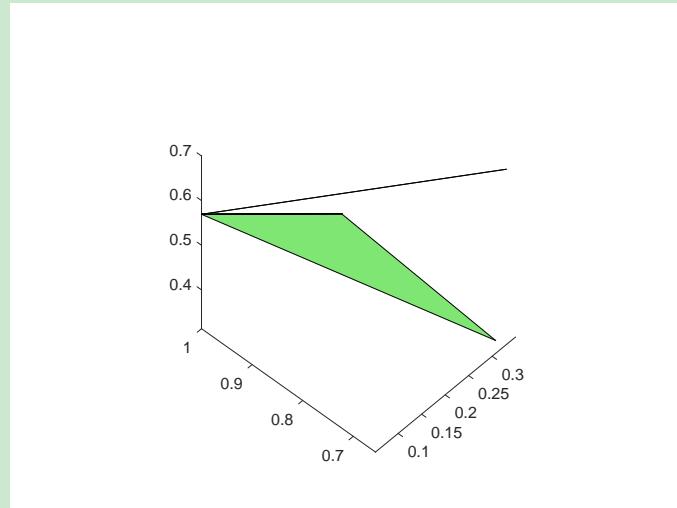
```



注意, 该图面上的是三角剖分, `totalTri` 是所有面的三角形连通性信息 (逐个单元排列且为整体编号, 含重复).

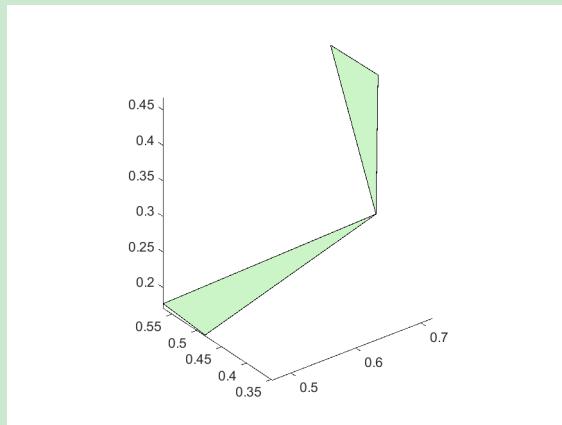
4.1.3 去除短边

对二维情形, 我们已经注意到 PolyMesher 在边界附近会生成极短的边, 这些边的两个点几乎可视为同一个点. 对三维问题, 面上的三角剖分也会出现这种情形, 即某些三角形的边几乎在一个点上 (面上三角形相当于退化为线), 如下图.

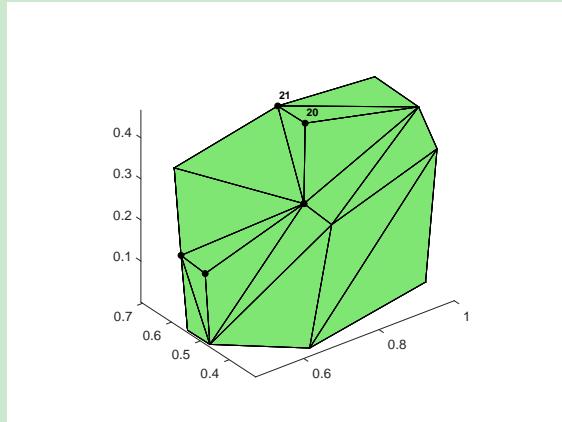


该图中的直线实际上是一个三角形, 只不过两个边几乎重合.

在 MPT 中, 我们也说明了, 短边不能轻易去除, 因为去除短边的过程会影响三角剖分的共面性. 当然, 若只去掉那些极短的边, 通常不必考虑共面问题. 但程序运行过程中偶尔也会出现错误, 如下图所示.



实际上, 上面两个三角形之间是有额外两个三角形的 (见下图), 但因去除极短边时导致所夹三角形的顶点偏离所在平面, 最终只留下上图的两个三角形.



为此, 即便只考虑去除极短的边, 最好也重新生成凸包, 再判断共面性.

现在考虑去除短边. 将所有三角形 (包含重复) 想象成展开在平面上, 则可按照二维问题处理.

我们重述一下思路 (这里仅考虑去除极短的边, 如边长小于 $1e-9$).

- 先找到所有的短边. 如下语句是为了去掉重复的三角形

```
1 Tri = unique(totalTri, 'rows');
```

但三角形的排列与顺序有关, 如三角形 1-2-3 和三角形 1-3-2 是同一个. 为此必须对三个点进行排序 (回忆基本数据结构中用 unique 获得 edge).

```
1 totalTri = sort(totalTri, 2);
2 Tri = unique(totalTri, 'rows');
3 totalEdge = sort([Tri(:, [2, 3]); Tri(:, [3, 1]); Tri(:, [1, 2])], 2);
4 [i, j] = find(sparse(totalEdge(:, 2), totalEdge(:, 1), 1));
5 edge = [j, i];
6 z1 = node(edge(:, 1), :); z2 = node(edge(:, 2), :);
7 he = sqrt(sum((z1 - z2).^2, 2));
8 ir = find(he < 1e-9);
9 nv1 = edge(ir, 1); nv2 = edge(ir, 2); % starting and ending indices
```

- 如下将极短边的终点 (或起点) 用起点 (或终点) 替换 (此时不需要关心边界的问题).

```
1 for i = 1:length(nv2)
2     v1 = nv1(i); v2 = nv2(i);
3     totalTri(totalTri == v2) = v1; % replaced by starting indices
4     nv1(nv1 == v2) = v1; nv2(nv2 == v2) = v1;
5 end
6 elemTriLen = cellfun('length', elemTri);
7 elemTri = mat2cell(totalTri, elemTriLen, 3);
```

最后的语句表示恢复为原来的单元形式.

- 接着我们要删除每个单元中含重复点的三角形. 注意, 与一般剖分不一样的是, 三角形删除一个边后就不存在了, 因而不需要保留三角形剩下的点.

```
1 for iel = 1:NT
2     index = elemTri{iel}; nindex = size(index, 1);
3     rows = true(nindex, 1);
4     for i = 1:size(index, 1)
5         id = index(i, :); nid = length(unique(id));
6         if nid < 3, rows(i) = false; end
7     end
8     elemTri{iel} = index(rows, :);
9 end
```

- 去除短边后, node 中含有一些无用点, 为此需获取新三角剖分的数据 (节点就是最终的 node). 做法与二维的 PolyMesher_Reorder.m 类似.

```
1 % ----- 获取面新三角剖分的 node, elemTri -----
2 totalTri = vertcat(elemTri{:}); % NTri * 3
3 [id, ~, totalid] = unique(totalTri);
4 node = V(id, :);
5 elemTriLen = cellfun('length', elemTri);
6 totalTri = reshape(totalid, size(totalTri, 1), 3);
7 elemTri = mat2cell(totalTri, elemTriLen, 3);
```

- 去除短边后, 我们还要恢复共面性

```

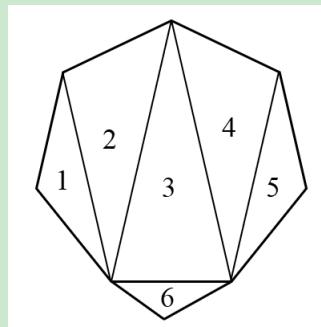
1 % restore the coplanarity
2 for iel = 1:NT
3     index = unique(elemTri{iel});
4     X = node(index,:); Tri = convhulln(X);
5     elemTri{iel} = index(Tri);
6 end

```

4.2 基本数据结构的获取

4.2.1 获得共面三角剖分

假设已经获得某个单元的面的三角剖分 Tri , 为了获得 elem , 须找到所有的共面三角形.



我们将采用 MPT 的思路获取共面三角形:

1. 给定一个三角形 T , 确定它所在平面的方程 (三点确定一个平面). 设三个顶点分别为 $z_i = (x_i, y_i, z_i)$, $i = 1, 2, 3$. 设 (x, y, z) 是平面上任一点, 则三角形所在平面的方程为

$$\vec{n} \cdot (x - x_1, y - y_1, z - z_1) = 0,$$

其中, \vec{n} 为平面的法向量, 可取为 $\vec{n} = \overrightarrow{z_2z_1} \times \overrightarrow{z_3z_1}$. 为了避免向量分量过小, 导致错误判断, 相乘的两个向量进行归一化.

2. 利用平面方程, 可确定当前单元在平面上的顶点. 这样, 对三角形单元进行循环, 只要与平面点集有三个交点的都位于该平面上.

上面的过程总结为如下程序

```

1 % ----- Determine elemf consisting of coplanar triangles -----
2 elem = cell(NT,1);
3 for iel = 1:NT
4     Tri = elemTri{iel}; elemf = []; s = 1;
5     while size(Tri,1)≥1
6         rows = false(size(Tri,1),1); rows(1) = true;
7         T = Tri(1,:); % current triangle
8         % indexf
9         index = unique(Tri); nindex = length(index); % 单元顶点编号
10        z1 = node(T(1,:)); z2 = node(T(2,:)); z3 = node(T(3,:));
11        nvec = cross(z2-z1, z3-z1);
12        nvec = nvec/norm(nvec); % normalization
13        pvec = node(index,:)-repmat(z1,nindex,1);
14        pvec = pvec/norm(pvec); % normalization
15        on = abs(dot(repmat(nvec,nindex,1)',pvec'))≤1e-8;

```

```

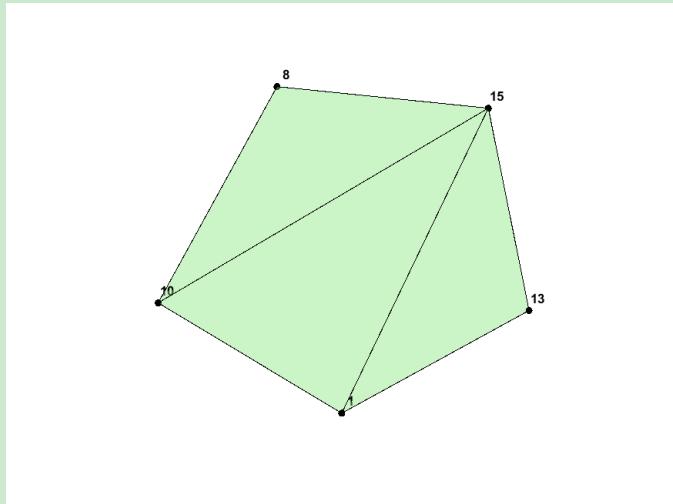
16     indexf = index(on);
17     % elemf
18     for i = 2:size(Tri,1)
19         T1 = Tri(i,:);
20         con = intersect(T1,indexf); ncon = length(con);
21         if ncon==3, rows(i) = true; end
22     end
23     elemf{s} = Tri(rows,:);
24     s = s+1;
25     Tri = Tri(~rows,:); % triangles remaining
26 end
27 elem{iel} = elemf';
28 end

```

注意, 每次循环 elemf 要重置为 [].

4.2.2 提取基本数据结构

已经获得 node, 现在获取按单元面存储的 elem.



- 如图所示, 对给定的面, 只要给出三角剖分的边界边即可 (相当于获得 MPT 中 adjVf 的信息).
-

```

1 iel = 1;
2     % edge of face
3     elemf = elem{iel};
4     s = 1;
5     face = elemf{s};
6     totalEdge = sort([face(:,[2,3]); face(:,[3,1]); face(:,[1,2])],2);
7     [i,j,sb] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
8     edge = [j,i]; edge = edge(sb==1,:); indexf = unique(edge);

```

其中的 indexf 记录面的顶点编号.

- 接下来获得每个顶点的连通节点, 用 adjVf 记录. 可将 edge 复制一遍得 repEdge, 从而由 repEdge 第一列知道每一个点的连通节点. 将 repEdge 按第一列排序, 其第二列每两个给出一个节点的结果, 例如

```

41    44
42    46
42    41
44    47
44    41
46    47
46    42
47    44
47    46

```

```

1      % adjVf
2      indexf = unique(edge); nindexf = length(indexf);
3      repEdge = [edge; edge(:,[2,1])];
4      repEdge = sortrows(repEdge,1);
5      adjVf = reshape(repEdge(:,2),[],2);

```

注意这里使用 sortrows.m 函数.

- 循环顺序如下获得

```

1      % cyclic order
2      order = zeros(1,nindexf);
3      pre_id = 1; previous = indexf(pre_id);
4      adj = adjVf(pre_id,:);
5      current = adj(1); cur_id = find(indexf==current);
6      order(1) = previous; order(2) = current;
7      for p = 2:nindexf-1
8          adj = adjVf(cur_id,:); next = adj(adj!=previous);
9          order(p+1) = next;
10         previous = current; current = next;
11         cur_id = find(indexf==current);
12     end

```

- 逆时针顺序如下

```

1      % counterclockwise order
2      column = det([ones(4,1), [node(order(1:3),:); P(iel,:)]])/6;
3      if column<0, order = order(end:-1:1); end
4      elemf{s} = order;

```

4.3 程序整理

在 tool/PolyMesher3D 文件夹中给出了 meshfun3D.m 函数, 如下

```

1 %Meshfun3D: generate basic data structure (node,elem) representing meshes
2 % Copyright (C) Terence Yue Yu.
3
4 % ----- Usage -----
5 % Preserved as meshdata.mat
6 % load('meshdata.mat') to load basic data structure: node, elem
7 %
8
9 clc;clear;close all

```

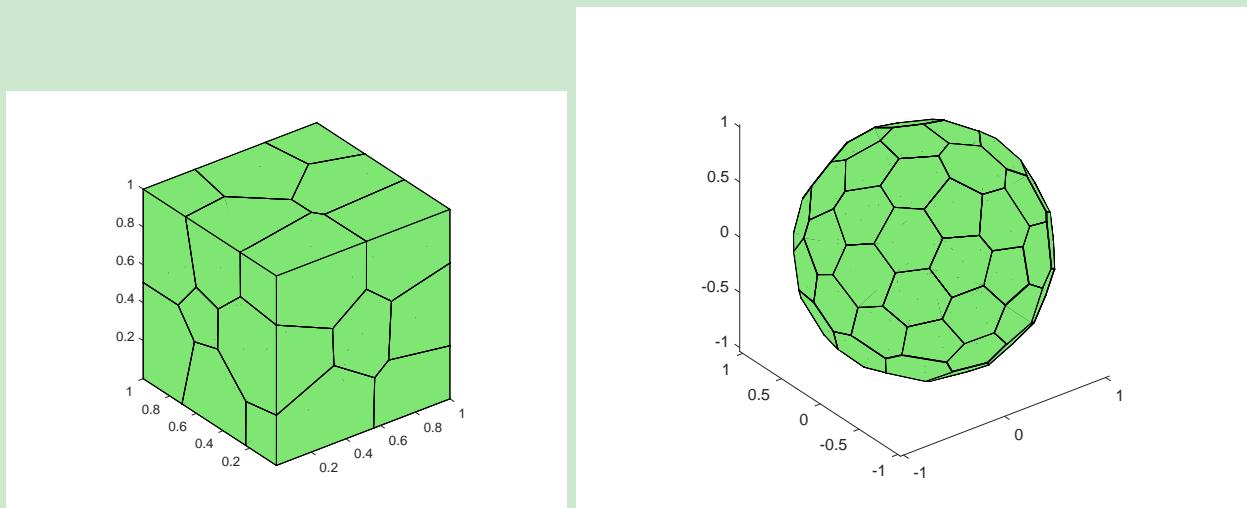
```

10
11 % ----- Choice of the domain -----
12 % Options: Cube_Domain (NT≥2), Sphere_Domain (NT≥40)
13 Domain = @Cube_Domain;
14 MaxIter = 50;
15 NT = 10;
16 [node,elem]= PolyMesher3D(Domain,MaxIter,NT);
17
18 % ----- Visualization of the mesh -----
19 option.FaceAlpha = 1;
20 figure, showmesh(node,elem,option); axis equal
21
22 iel = 2;
23 elemf = elem{iel};
24 figure, showmesh(node,elemf); axis equal
25 range = unique(horzcat(elemf{:}));
26 findnode(node,range)
27
28 % % ----- Save the mesh data -----
29 %save meshdata node elem

```

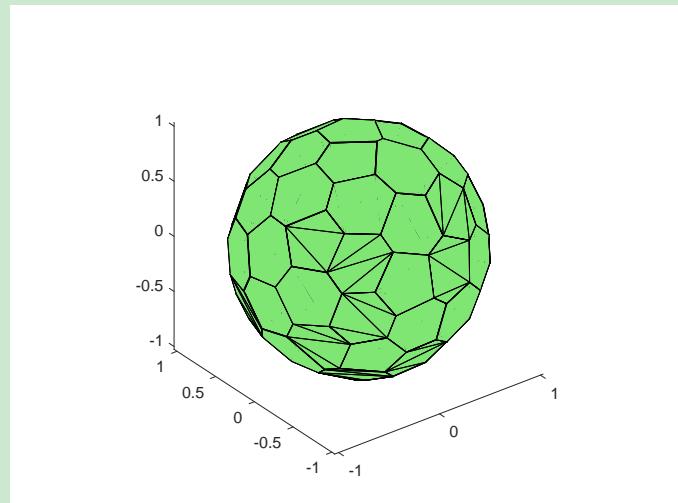
如下调用 (随机生成种子): `[node,elem]= PolyMesher(Domain,MaxIter,NT).`

以上所有程序放置在 tool 的子文件夹 PolyMesher3D 中, 其中包含了一些区域和符号距离函数的定义, 见子文件夹 Domain 和 DistFnc.



注 4.1 当点数较少时可能出现错误剖分或 voronoin 函数提示某些错误, 此时必须增加点数. 对方体区域, 一般不会出错; 对球体区域, NT 一般不小于 40 (此时面的近似还是很粗糙的).

注 4.2 对球体这种区域, 前面给出的网格剖分只是去除了极短的边. 若要去除其他一些相对短的边, 则由于共面性要求, 一些面会出现三角剖分. 例如, 去除小于最大边长 0.1^3 倍的边, 结果如下.



Part II

经典问题的虚拟元方法

5 Poisson 方程的协调 VEM

5.1 VEM 简介

考虑 Poisson 方程的 Dirichlet 问题

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega. \end{cases} \quad (5.5)$$

为了方便, 以下设 $g = 0$. 变分问题为: 找 $u \in V := H_0^1(\Omega)$, 使得

$$a(u, v) = \ell(v),$$

其中,

$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx, \quad \ell(v) = \int_{\Omega} f v \, dx.$$

VEM 的局部形函数空间为

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

式中,

$$\mathbb{B}_k(\partial K) = \{v \in C^0(\partial K) : v|_e \in \mathbb{P}_k(e), \quad e \subset \partial K\}$$

称为边界元空间. 给定区域的剖分 \mathcal{T}_h , 对模型问题 (5.5), 协调有限元空间为

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

为了表达自由度, 设 d -维区域 D 上的尺度单项式集合 $\mathbb{M}_r(D)$ 为

$$\mathbb{M}_r(D) := \left\{ \left(\frac{\mathbf{x} - \mathbf{x}_D}{h_D} \right)^s, |s| \leq r \right\},$$

其中, h_D 是 D 的直径, \mathbf{x}_D 是 D 的重心, r 是非负正整数. 对多重指标 $s \in \mathbb{N}^d$, 定义

$$\mathbf{x}^s = x_1^{s_1} \cdots x_d^{s_d}, \quad |s| = s_1 + \cdots + s_d.$$

规定, 当 $r \leq -1$ 时, $\mathbb{M}_r(D) = \{0\}$.

对偶空间为

$$\mathcal{X}_{k,k-2}(K) = \text{span}\{\chi_a, \chi_e^{k-2}, \chi_K^{k-2}\}, \quad (5.6)$$

其中, 泛函向量由如下自由度组成

- χ_a : K 顶点处的函数值,

$$\chi_a^z(v) = v(z), \quad z \text{ is a vertex of } K.$$

- χ_e^{k-2} : 边上 $k-1$ 个点或直到 $k-2$ 阶的矩量,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-2}(e), \quad e \subset \partial K.$$

- χ_K^{k-2} : 单元 K 上直到 $k-2$ 阶的矩量,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-2}(K).$$

定义局部 H^1 投影

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

满足

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

或

$$\begin{cases} (\nabla(\Pi_k^\nabla v), \nabla p)_K = (\nabla v, \nabla p)_K, & p \in \mathbb{P}_k(K), \\ P_0(\Pi_k^\nabla v) = P_0(v). \end{cases} \quad (5.7)$$

这里, 限制条件为

$$P_0(v) = \int_{\partial K} v \, ds, \quad k = 1; \quad P_0(v) = \int_K v \, dx, \quad k \geq 2.$$

近似变分问题为: 求 $u_h \in V_h$, 使得

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

式中,

$$\begin{aligned} a_h(u, v) &= \sum_{K \in \mathcal{T}_h} a_h^K(u, v), \\ a_h^K(u, v) &= \int_K \nabla(\Pi_k^\nabla u) \cdot \nabla(\Pi_k^\nabla v) \, dx + S^K(u - \Pi_k^\nabla u, v - \Pi_k^\nabla v), \\ \ell_h(v) &= \sum_{K \in \mathcal{T}_h} \ell_h^K(v), \quad \ell_h^K(v) = \int_K f P_h v \, dx. \end{aligned}$$

在上式中, $P_h v$ 是 v 的某种可计算的近似, 如

$$P_h(v) = \Pi_k^\nabla v, \quad k = 1.$$

稳定项由自由度给出:

$$S^K(v - \Pi_k^\nabla v, w - \Pi_k^\nabla w) = \sum_{i=1}^{N_k} \chi_i(v - \Pi_k^\nabla v) \chi_i(w - \Pi_k^\nabla w),$$

其中, N_k 是自由度的个数.

5.2 椭圆投影的计算

5.2.1 过渡矩阵 D

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

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

查看结构体 aux 可以看到, 重心、单元直径和单元面积与那里的一致.

类似线性代数中向量在基下的表示, 我们把虚拟元空间 $V_k(K)$ 的基函数排成行向量, 记为

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

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

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

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

$$(m_1, m_2, \dots, m_{N_p}) = (\phi_1, \phi_2, \dots, \phi_{N_k})D \quad \text{或} \quad m^T = \phi^T D, \quad (5.8)$$

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

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

且 D 是 $N_k \times N_p$ 的矩阵.

过渡矩阵是可计算的. 考虑一次元, 即 $k = 1$ 的情形. 我们有

$$\begin{aligned} \chi_i(v) &= v(z_i), \quad i = 1, \dots, N_k = N_v, \\ m^T &= \left\{ m_1(x, y) = 1, \quad m_2(x, y) = \frac{x - x_K}{h_K}, \quad m_3(x, y) = \frac{y - y_K}{h_K} \right\}, \end{aligned}$$

从而

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

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

```

1 node = [0 0; 3 0; 3 2; 3/2 4; 0 4]; elem = {1:5};
2 aux = auxgeometry(node, elem);
3
4 node = aux.node; elem = aux.elem;
5 centroid = aux.centroid; diameter = aux.diameter;
6
7 NT = size(elem, 1); D = cell(NT, 1);
8 for iel = 1:NT
9     index = elem{iel};      Nv = length(index);
10    xK = centroid(iel, 1); yK = centroid(iel, 2); hK = diameter(iel);
11
12    m = @(x,y) [1+0*x, (x-xK)./hK, (y-yK)./hK]; % m1, m2, m3
13
14    x = node(index, 1); y = node(index, 2);
15    D1 = m(x, y); D{iel} = D1;
16 end

```

注意这里多个匿名函数的使用. 所得结果与文献 [1] 的一致.

5.2.2 椭圆投影在基下的矩阵

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

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

根据自由度的定义, Π_k^∇ 的第 j 列就是 $\Pi_k^\nabla \phi_j$ 的自由度向量, 即

$$\Pi_k^\nabla = (\chi_i(\Pi_k^\nabla \phi_j)). \quad (5.9)$$

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

$$\Pi_k^\nabla \phi^T = m^T \Pi_{k*}^\nabla, \quad (5.10)$$

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

$$\Pi_k^\nabla = D \Pi_{k*}^\nabla.$$

定义 5.7 等价于

$$\begin{cases} a^K(\Pi_k^\nabla \phi_i, m_\alpha) = a^K(\phi_i, m_\alpha), & i = 1, \dots, N_k, \alpha = 1, \dots, N_p \\ P_0(\Pi_k^\nabla \phi_i) = P_0(\phi_i) \end{cases}$$

或

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

写成向量形式为

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

或

$$\begin{cases} \int_K \nabla m \cdot \nabla \Pi_k^\nabla \phi^T dx = \int_K \nabla m \cdot \nabla \phi^T dx, \\ P_0(\Pi_k^\nabla \phi^T) = P_0(\phi^T). \end{cases} \quad (5.11)$$

令

$$G = a^K(m, m^T) = \int_K \nabla m \cdot \nabla m^T dx,$$

$$B = a^K(m, \phi^T) = \int_K \nabla m \cdot \nabla \phi^T dx,$$

则由 (5.11) 知

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

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

注意, G 不可逆, 因为 ∇m 的第一个分量为零, 从而 G 的第一行全为零. 这是因为 (5.11) 不考虑限制条件时, 投影不唯一. 这样, 我们只要把第一行恒为零的方程用投影的限制条件替换, 即

$$\tilde{G} \Pi_{k*}^\nabla = \tilde{B},$$

式中,

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

则 \tilde{G} 可逆, 且

$$\Pi_{k*}^\nabla = \tilde{G}^{-1} \tilde{B}, \quad \Pi_k^\nabla = D \Pi_{k*}^\nabla.$$

注 5.1 G 不可逆, 一般是某行为零, 或某两行线性相关. 注意到 MATLAB 的 $A \setminus b$ 对非方阵也可使用, 我们可把限制条件对应的行向量直接排在矩阵的底部.

现在考虑相关矩阵的计算. 注意, \tilde{G} 或 G 不需要直接计算, 这是因为我们有如下的一致性关系 (consistency relation).

引理 5.1

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

证明 对第一式, 直接验证有

$$G = \int_K \nabla m \cdot \nabla m^T dx = \int_K \nabla m \cdot \nabla \phi^T dx D = BD.$$

对第二式,

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

□

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

先考虑 B 的计算. 分部积分有

$$B = \int_K \nabla m \cdot \nabla \phi^T dx = - \int_K \Delta m \cdot \phi^T dx + \sum_{e \subset \partial K} \int_e (\nabla m \cdot \mathbf{n}_e) \phi^T ds, \quad (5.12)$$

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

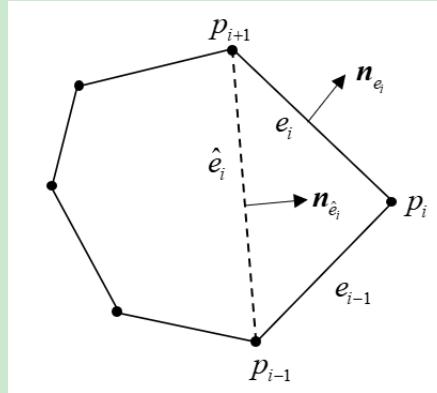


图 10. $n_{\hat{e}_i}$ 的示意图

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

点, p_{i+1} 表示下一个点, 则有

$$\begin{aligned} & \sum_{e \subset \partial K} \int_e (\nabla m_\alpha \cdot \mathbf{n}_e) \phi_i ds \\ &= \sum_{e \subset \partial K} (\nabla m_\alpha \cdot \mathbf{n}_e) \int_e \phi_i ds = \sum_{j=1}^{N_v} (\nabla m_\alpha \cdot \mathbf{n}_{e_j}) \frac{\phi_i(p_j) + \phi_i(p_{j+1})}{2} |e_j| \\ &= \nabla m_\alpha \cdot \frac{|e_{i-1}| \mathbf{n}_{e_{i-1}} + |e_i| \mathbf{n}_{e_i}}{2} \end{aligned} \quad (5.13)$$

考察三角形 $\Delta p_{i-1} p_i p_{i+1}$, 我们有

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

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

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

或

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

综上,

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

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

为了计算 B , 我们只要计算出

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

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

```

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

```

5.2.3 投影限制条件

对 $k = 1$, 计算中 P_0 通常采用如下离散形式

$$P_0 v = \frac{1}{N_v} \sum_{i=1}^{N_v} v(p_i), \quad (5.14)$$

从而

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

我们把前面所有的矩阵保存为元胞数组.

```

1 % aux = auxgeometry(node, elem);
2 node = aux.node; elem = aux.elem;
3 centroid = aux.centroid; diameter = aux.diameter;
4 NT = size(elem, 1);
5
6 D = cell(NT, 1);
7 % B = cell(NT, 1); % it is not used in the computation
8 Bs = cell(NT, 1);
9 G = cell(NT, 1); Gs = cell(NT, 1);
10 for iel = 1:NT
11     % element information
12     index = elem{iel}; Nv = length(index);
13     xK = aux.centroid(iel, 1); yK = aux.centroid(iel, 2);
14     hK = aux.diameter(iel);
15     x = node(index, 1); y = node(index, 2);
16     % scaled monomials
17     m1 = @(x,y) 1 + 0*x; m2 = @(x,y) (x-xK)./hK; m3 = @(x,y) (y-yK)./hK;
18     m = @(x,y) [m1(x,y), m2(x,y), m3(x,y)];
19     Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
20     % D
21     D1 = m(x,y); D{iel} = D1;
22     % B, Bs, G, Gs
23     rotid1 = [Nv, 1:Nv-1]; rotid2 = [2:Nv, 1]; % ending and starting indices
24     normVec = 0.5*[y(rotid2)-y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of edge ...
25     % vector
26     B1 = Gradm*normVec; % B
27     B1s = B1; B1s(1,:) = 1/Nv; Bs{iel} = B1s;
28     G{iel} = B1*D1; Gs{iel} = B1s*D1;

```

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

5.3 L^2 投影与强化技巧

5.3.1 提升空间中的椭圆投影

考虑提升空间

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

相比 $V_k(K)$, 它的自由度多出 K 上的 $k-1, k$ 阶矩量. 对 $v \in \tilde{V}_k(K)$, 因边界多项式次数未提高, 我们仍然只能如 $V_k(K)$ 中那样定义椭圆投影 $\Pi_k^\nabla v$, 即

$$\Pi_k^\nabla : \tilde{V}_k(K) \rightarrow \mathbb{P}_k(K), \quad v \mapsto \Pi_k^\nabla v$$

满足

$$(\nabla \Pi_k^\nabla v, \nabla p)_K = (\nabla v, \nabla p)_K, \quad p \in \mathbb{P}_k(K).$$

显然, $\Pi_k^\nabla v$ 的计算用不到 K 上 v 的 $k-1, k$ 阶矩量. 正因为如此, 我们可用 $\Pi_k^\nabla v$ 的 $k-1, k$ 阶矩量近似 v 的相应矩量, 从而减少自由度个数. 由此引入强化空间

$$W_k(K) = \left\{ w_h \in \tilde{V}_k(K) : (w_h - \Pi_k^\nabla w_h, q)_K = 0, \quad q \in \mathbb{M}_k \setminus \mathbb{M}_{k-2} \right\}.$$

在该空间中, 显然有

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

现在我们简单说明 $\tilde{V}_k(K)$ 中椭圆投影的计算.

- $\tilde{V}_k(K)$ 中多出 $k-1, k$ 阶矩量自由度 (不一定就是两个自由度), 将它们排列在最后. 设多出 n 个自由度 (从而多出 n 个基函数), 分别记为 $d_i(v), i = 1, \dots, n$.
- 过渡矩阵 D 多出 n 行, 即 $d_i(m^T), i = 1, \dots, n$.
- 对矩阵

$$B = \int_K \nabla m \cdot \nabla \phi^T dx,$$

它比原来的多出 n 列, 对应 $k-1, k$ 阶矩量. 由于分部积分后, 右端的计算不涉及到这些矩量, 因此多出的 n 列全为零, 其他列不变. 限制条件也是如此. 这表明, 此时的 \tilde{B} 只比原来的多出 n 列零向量.

- 有了过渡矩阵和 B, \tilde{B} , 就可计算出 $\tilde{V}_k(K)$ 中的投影矩阵 $\boldsymbol{\Pi}_k^\nabla$. 注意到该矩阵的第 j 列就是第 j 个基函数的投影的自由度向量 $\chi(\Pi_k^\nabla \phi_j)$, 我们有 $\boldsymbol{\Pi}_k^\nabla$ 的最后 n 行为

$$d_i(\Pi_k^\nabla \phi^T), \quad i = 1, \dots, n.$$

注意这里的 ϕ^T 是 $\tilde{V}_k(K)$ 的基函数, 去掉最后 n 个即为 $V_k(K)$ 中的 (虽然基函数可能不同, 但相同自由度值相同, 而椭圆投影由自由度唯一决定).

5.3.2 L^2 投影的计算

对含低阶项的问题, 即反应-扩散问题, 我们要计算 L^2 投影, 定义为

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

这等价于

$$(\Pi_k^0 \phi_i, m_\alpha)_K = (\phi_i, m_\alpha)_K, \quad i = 1, \dots, N_k, \quad \alpha = 1, \dots, N_p \quad (5.15)$$

或

$$\int_K m \Pi_k^0 \phi^T dx = \int_K m \phi^T dx. \quad (5.16)$$

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

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

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

$$\Pi_k^0 \phi^T = m^T \boldsymbol{\Pi}_{k*}^0. \quad (5.17)$$

由 (5.8) 知,

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

把这些关系代入 (5.16), 我们有 L^2 投影的矩阵表示

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

式中,

$$H = \int_K mm^T dx, \quad C = \int_K m\phi^T dx.$$

H 是可逆的 (Gram 矩阵), 但 C 不可计算, 因为它涉及到 K 上 $k-1, k$ 阶的矩量. 根据前面的说明, 我们将在强化空间 $W_k(K)$ 中考虑, 前面的 $V_k(K)$ 都换为 $W_k(K)$, 相应的基函数对应 $W_k(K)$. C 中多出的矩量 $d_i(\phi^T)$ 用 $d_i(\Pi_k^\nabla \phi^T)$ 代替.

可以证明, 当 $k=1, 2$ 时,

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

此时的 L^2 投影似乎不需要考虑计算问题, 因为已经知道

$$\Pi_k^0 = \Pi_k^\nabla, \quad \Pi_{k*}^0 = \Pi_{k*}^\nabla.$$

实则不然, 在双线性形式的表示中仍要计算矩阵 H .

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

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

式中,

$$\begin{cases} x = x_1\lambda_1 + x_2\lambda_2 + x_3\lambda_3 \\ y = y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3 \end{cases}, \quad \lambda_1 + \lambda_2 + \lambda_3 = 1, \quad (5.18)$$

注意到

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

这里 S 是三角形 T 的代数面积, 我们有

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

因变换前后点出的值不变, 故

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

利用 (5.18) 可把参考元上的 Gauss 点 $(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i})$ 转化为 T 上的点. 对 VEM 的单元 K , 给出其三角剖分的基本数据结构, 记为 `nodeT`, `elemT`, 如下进行 K 上的积分.

```

1 function Int = integralTri(fun,n,nodeT,elemT)
2 % % integralTri: approximate integrals in a polygonal domain
3 % with triangulation (nodeT, elemT).
4 % n: n-th order quadrature rule
5 % fun: one or more anonymous functions, e.g. fun = @(x,y) [f1(x,y), f2(x,y)]
6
7 [lambda,weight] = quadpts(n);
8 NT = size(elemT,1); Int = 0;

```

```

9 for iel = 1:NT
10    vT = nodeT(elemT(iel,:));
11    area = 0.5*abs(det([[1;1;1],vT]));
12    xy = lambda*vT;
13    f = fun(xy(:,1),xy(:,2));
14    Int = Int + area*weight*f;
15 end

```

注意 `fun` 容许是多个按行排列的匿名函数.

$k = 1$ 时的 H 如下计算

```

1 % H
2 nodeT = [node(index,:);centroid(iel,:)];
3 elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]';
4 m = @ (x,y) [m1(x,y).*m1(x,y), m1(x,y).*m2(x,y), m1(x,y).*m3(x,y), ...
5 m2(x,y).*m1(x,y), m2(x,y).*m2(x,y), m2(x,y).*m3(x,y), ...
6 m3(x,y).*m1(x,y), m3(x,y).*m2(x,y), m3(x,y).*m3(x,y)];
7 H1 = zeros(3,3);
8 H1(:) = integralTri(m,2,nodeT,elemT); % n = 2
9 H{iel} = H1;

```

注 5.2 当 m_α 较多时, H 可循环计算.

```

1 m1 = @ (x,y) 1 + 0*x;
2 m2 = @ (x,y) (x-xK)./hK;
3 m3 = @ (x,y) (y-yK)./hK;
4 m = {m1,m2,m3};
5
6 H1 = zeros(3,3);
7 for i = 1:3
8     for j = 1:3
9         fun = @ (x,y) m{i}(x,y).*m{j}(x,y);
10        H1(i,j) = integralTri(fun,2,nodeT,elemT);
11    end
12 end
13 H{iel} = H1;

```

5.4 Poisson 方程一阶 VEM 程序

5.4.1 问题描述

考虑更一般的问题

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

虚拟元方法为: 找 $u_h \in V_h^g$, 使得

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

式中,

- 空间为

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

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

注意, 对 $\alpha \neq 0$ 的情形, 要考虑 enhancement 空间 W_h .

- 双线性形式 $a_h^K(u, v)$ 同前, 而

$$b_h(u, v) = \sum_{K \in \mathcal{T}_h} b_h^K(u, v), \quad b_h^K(u, v) = \alpha \int_K \Pi_k^0 u \cdot \Pi_k^0 v \, dx.$$

- 右端为

$$F_h(v) = \ell_h(v) + \int_{\Gamma_N} g_N v \, ds,$$

其中,

$$\ell_h(v) = \sum_{K \in \mathcal{T}_h} \int_K f P_h v \, dx.$$

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

5.4.2 单元刚度矩阵的计算

为了方便, 令

$$A_K^1(i, j) = a^K(\Pi_k^\nabla \phi_j, \Pi_k^\nabla \phi_i) = \int_K \nabla(\Pi_k^\nabla \phi_j) \cdot \nabla(\Pi_k^\nabla \phi_i) \, dx,$$

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

第一式写成矩阵形式为

$$A_K^1 = \int_K \nabla \Pi_k^\nabla \phi \cdot \nabla \Pi_k^\nabla \phi^T \, dx = (\boldsymbol{\Pi}_{k*}^\nabla)^T \int_K \nabla m \cdot \nabla m^T \, dx \boldsymbol{\Pi}_{k*}^\nabla = (\boldsymbol{\Pi}_{k*}^\nabla)^T G \boldsymbol{\Pi}_{k*}^\nabla,$$

至于第二式, 由 (5.9) 知, $\chi_r(\Pi_k^\nabla \phi_i) = (\boldsymbol{\Pi}_k^\nabla)_{ri}$, 于是

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

或

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

类似有,

$$B_K = \int_K \Pi_k^0 \phi \cdot \Pi_k^0 \phi^T \, dx = (\boldsymbol{\Pi}_{k*}^0)^T \int_K m m^T \, dx \boldsymbol{\Pi}_{k*}^0 = (\boldsymbol{\Pi}_{k*}^0)^T H \boldsymbol{\Pi}_{k*}^0,$$

而

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

如下给出单元刚度矩阵

```

1 % ----- Elementwise stiffness matrix and load vector -----
2 ABelem = cell(NT,1); belem = cell(NT,1);
3 for iel = 1:NT
4     Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi));
5     AK = Pis'*G{iel}*Pis + (I-Pi)*(I-Pi);
6     BK = pde.c*Pis'*H{iel}*Pis + pde.c*hK^2*(I-Pi)*(I-Pi); % Pis = Pi0s;
7     ABelem{iel} = reshape(AK'+BK',1,[]); % straighten as row vector for easy assembly
8 end

```

注 5.3 这里对单元刚度矩阵进行了行拉直, 见后面装配的说明.

5.4.3 单元载荷向量的计算

对 $k = 1$, 右端为

$$\ell_h^K(v) = \int_K f \Pi_k^\nabla v dx,$$

单元载荷向量

$$F_K = \int_K f \Pi_k^\nabla \phi dx = (\Pi_{k*}^\nabla)^T \int_K f m dx, \quad (5.19)$$

这里用到 (5.10). 右端的积分若采用中点型积分公式近似, 则有

$$F_K = \int_K f \Pi_k^\nabla \phi dx = (\Pi_{k*}^\nabla)^T \int_K f m dx = (\Pi_{k*}^\nabla)^T \begin{bmatrix} f(x_K, y_K)|K| \\ 0 \\ 0 \end{bmatrix}.$$

```

1 % Load vector
2 fK = Pis'*[pde.f(centroid(iel,:))*area(iel);0;0];

```

当然, (5.19) 的右端也可用 Gauss 积分近似.

```

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

```

注 5.4 这里对载荷向量进行了行拉直, 见后面装配的说明. 两种方法误差量级一致, 本文采用第一种. 注意, 对高阶方法, 用第三种.

最终的单元刚度矩阵和载荷向量如下计算

```

1 % ----- Elementwise stiffness matrix and load vector -----
2 ABelem = cell(NT,1); belem = cell(NT,1);
3 for iel = 1:NT
4     % Projection
5     Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi));
6     % Stiffness matrix
7     AK = Pis'*G{iel}*Pis + (I-Pi)*(I-Pi);
8     BK = pde.c*Pis'*H{iel}*Pis; % Pis = Pi0s;

```

```

9      ABelem{iel} = reshape(AK'+BK',1,[]); % straighten as row vector for easy assembly
10     % Load vector
11     fK = Pis'*[pde.f(centroid(iel,:))*area(iel);0;0];
12     belem{iel} = fK'; % straighten as row vector for easy assembly
13 end

```

5.4.4 刚度矩阵与载荷向量的装配

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

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

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

```

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

```

- 对固定单元, 设单元刚度矩阵为

$$[K^e] = \begin{matrix} & u_i & u_j & u_l \\ u_i & k_{11} & k_{12} & k_{13} \\ u_j & k_{21} & k_{22} & k_{23} \\ u_l & k_{31} & k_{32} & k_{33} \end{matrix},$$

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

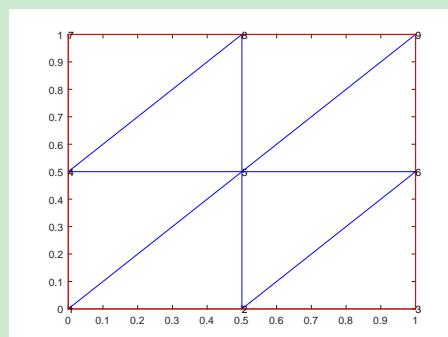
```
S = sparse(i, j, s, m, n, nzmax);
```

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

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

该命令把所有单元刚度矩阵的 $(1,2)$ 处的值放到了整体刚度矩阵的对应位置中 (k_{12} 是列向量, 存储所有单元的).

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



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

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

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

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

$$\begin{bmatrix} i_{11} & j_{11} & s_{11} \\ i_{12} & j_{12} & s_{12} \\ \vdots & \vdots & \vdots \\ i_{33} & j_{33} & s_{33} \end{bmatrix}$$

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

```

1 NT = size(elem,1); Ndof = 3; nnz = NT*Ndof^2;
2 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
3 id = 0;
4 for i = 1:Ndof
5     for j = 1:Ndof
6         ii(id+1:id+NT) = elem(:,i); % zi
7         jj(id+1:id+NT) = elem(:,j); % zj
8         ss(id+1:id+NT) = K(:,i,j); % kij
9         id = id + NT;
10    end
11 end

```

这里假设 (i, j) 位置的单元刚度矩阵值用三维数组存储.

注 5.5 可用向量法给出 ii, jj , 只需要两行语句, 即

```

1 ii = reshape(repmat(elem,Ndof,1),[],1);
2 jj = repmat(elem(:,1:Ndof),1);

```

以后都采用这种方式. 为了对应装配指标, 我们把单元刚度矩阵如下存储

$$K = [k_{11}, k_{12}, k_{13}, \dots, k_{33}],$$

这里 k_{11} 是所有单元刚度矩阵 $(1, 1)$ 位置的结果, 显然 ss 由该矩阵按列拉直得到. 本文均采用这种处理.

- 用稀疏矩阵, 右端可如下实现

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

这里, F_1 是所有单元载荷第 1 个位置的元素组成的向量. 当然也可用

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

向量不必使用 `sparse` 命令, b 一般用下面的语句代替

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

同理, 右端向量也按行拉直存储.

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

- 首先可找到所有的构型

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

假设现在只有 4,5,6,7 边形, 且分别有 N_4, N_5, N_6, N_7 个.

- 对 4 边形, 用 $N_v = 4$; $id_{Nv} = \text{find}(elemLen == N_v)$; 可找到所有 4 边形的序号. 比如, 第 67, 82 这两个单元是 4 边形. 对这两个相同构型的单元, 我们就可按照之前的思路进行装配, 找到相应的三元组, 记为 (i_4, j_4, s_4) . 类似可找到其他构型的, 分别记为 (i_5, j_5, s_5) , (i_6, j_6, s_6) , (i_7, j_7, s_7) . 由此就可生成最终的三元组

$$(i, j, s) := \begin{bmatrix} i_4 & j_4 & s_4 \\ i_5 & j_5 & s_5 \\ i_6 & j_6 & s_6 \\ i_7 & j_7 & s_7 \end{bmatrix}.$$

注意, nnz 是所有单元刚度矩阵的元素个数之和, 如 $\text{nnz} = \text{sum}(\text{elemLen}.^2)$.

装配程序如下

```
1 %% Assemble stiffness matrix and load vector
2 elemLen = cellfun('length',elem); vertNum = unique(elemLen);
3 nnz = sum(elemLen.^2);
4 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
5 id = 0; ff = zeros(N,1);
6 elem2dof = cell(NT,1);
7 for Nv = vertNum(:)' % only valid for row vector
8
9     % assemble the matrix
10    idNv = find(elemLen == Nv); % find polygons with Nv vertices
11    NTv = length(idNv); % number of elements with Nv vertices
12
13    elemNv = cell2mat(elem(idNv)); % elem
14    K = cell2mat(ABelem(idNv)); F = cell2mat(Belem(idNv));
15    Ndof = Nv;
```

```

16     ii(id+1:id+NTv*Ndof^2) = reshape(repmat(elemNv, Ndof, 1), [], 1);
17     jj(id+1:id+NTv*Ndof^2) = repmat(elemNv(:, ), Ndof, 1);
18     ss(id+1:id+NTv*Ndof^2) = K(:, );
19     id = id + NTv*Ndof^2;
20
21 % assemble the vector
22 ff = ff + accumarray(elemNv(:, ), F(:, ), [N 1]);
23
24 % elementwise global indices
25 elem2dof(idNv) = mat2cell(elemNv, ones(NTv, 1), Ndof);
26 end
27 kk = sparse(ii, jj, ss, N, N);

```

5.4.5 边界条件的处理

考虑 Neumann 边界条件. 设 e 是边界边, $u|_e$ 是一次多项式, 它可表为 $u = u_1\phi_1 + u_2\phi_2$. 这里, ϕ_1, ϕ_2 是一维单元的节点基 (一次多项式). 该条边对应的“边界单元载荷向量”为

$$F_e = [F_1, F_2]^T, \quad F_i = \int_e \partial_n u \phi_i ds.$$

用梯形公式近似 (或用中心格式), 有

$$\begin{aligned} \int_e \partial_n u \phi_1 ds &= \frac{h_e}{2} (\partial_n u(z_1)\phi_1(z_1) + \partial_n u(z_2)\phi_1(z_2)) = \frac{h_e}{2} \partial_n u(z_1), \\ \int_e \partial_n u \phi_2 ds &= \frac{h_e}{2} (\partial_n u(z_1)\phi_2(z_1) + \partial_n u(z_2)\phi_2(z_2)) = \frac{h_e}{2} \partial_n u(z_2), \end{aligned}$$

式中,

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

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

$$h_e \partial_n u = h_e \nabla u \cdot \mathbf{n} = \nabla u \cdot (h_e \mathbf{n}),$$

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

我们在 Poissonsdata.m 中定义 $g_N = [u_x(x, y), u_y(x, y)]$, 根据有限元一维问题的装配, Neumann 边界条件如下处理.

```

1 %% Assemble Neumann boundary conditions
2 bdEdgeN = bdStruct.bdEdgeN;
3 if ~isempty(bdEdgeN)
4     g_N = pde.Du;
5     z1 = node(bdEdgeN(:, 1), :); z2 = node(bdEdgeN(:, 2), :);
6     e = z1-z2; % e = z2-z1
7     Ne = [-e(:, 2), e(:, 1)]; % scaled ne
8     F1 = sum(Ne.*g_N(z1), 2)/2;
9     F2 = sum(Ne.*g_N(z2), 2)/2;
10    FN = [F1, F2];
11    ff = ff + accumarray(bdEdgeN(:, ), FN(:, ), [N 1]);
12 end

```

Dirichlet 边界条件比较容易, 如下

```

1 %% Apply Dirichlet boundary conditions
2 g_D = pde.g_D; bdNodeIdx = bdStruct.bdNodeIdx;
3 isBdNode = false(N,1); isBdNode(bdNodeIdx) = true;
4 bddof = find(isBdNode); freedof = find(~isBdNode);
5 pD = node(bddof,:);
6 u = zeros(N,1); uD = g_D(pD); u(bddof) = uD(:);
7 ff = ff - kk*u;

```

5.4.6 L^2 和 H^1 误差分析

设 u 是精确解, u_h 是数值解. 注意 u_h 在单元内部的值未知, 我们用 $\Pi_k^\nabla u_h$ 来代替. 也就是说, 我们用椭圆投影的误差代替数值解的误差. 离散 L^2 误差和离散 H^1 误差分别定义为

$$\text{ErrL2} = \left(\sum_{K \in \mathcal{T}_h} \|\boldsymbol{u} - \boldsymbol{\Pi}_k^\nabla \boldsymbol{u}_h\|_{0,K}^2 \right)^{1/2}$$

和

$$\text{ErrH1} = \left(\sum_{K \in \mathcal{T}_h} |\boldsymbol{u} - \boldsymbol{\Pi}_k^\nabla \boldsymbol{u}_h|_{1,K}^2 \right)^{1/2}.$$

下面给出误差可计算的表达式. 设单元 K 上的局部自由度向量为 $\boldsymbol{\chi}_K$, 则 $\boldsymbol{u}_h|_K = \boldsymbol{\phi}^T \boldsymbol{\chi}_K$. 需要注意的是, 这里的 $\boldsymbol{\chi}_K$ 是局部自由度向量, 它可能与整体自由度存在差别. 这是因为, 对某些带方向的自由度, 整体上要先固定方向, 而整体限制在局部单元上时, 方向可能恰好与单元方向相反. 这种问题, 后面再处理.

椭圆投影用自由度表示为

$$\boldsymbol{\Pi}_k^\nabla \boldsymbol{u}_h = \boldsymbol{\Pi}_k^\nabla \boldsymbol{\phi}^T \boldsymbol{\chi}_K = \boldsymbol{m}^T \boldsymbol{\Pi}_{k*}^\nabla \boldsymbol{\chi}_K =: a_1 m_1 + a_2 m_2 + a_3 m_3,$$

其中, $(a_1, a_2, a_3)^T = \boldsymbol{\Pi}_{k*}^\nabla \boldsymbol{\chi}_K$. 于是,

$$\begin{aligned} |\boldsymbol{u} - \boldsymbol{\Pi}_k^\nabla \boldsymbol{u}_h|_{1,K}^2 &= |\boldsymbol{u} - (a_1 m_1 + a_2 m_2 + a_3 m_3)|_{1,K}^2 \\ &= \|\nabla \boldsymbol{u} - \nabla(a_1 m_1 + a_2 m_2 + a_3 m_3)\|_{0,K}^2 \\ &= \int_K (\nabla \boldsymbol{u} - \nabla(a_1 m_1 + a_2 m_2 + a_3 m_3))^2 dx \\ &= \sum_\tau \int_\tau (\nabla \boldsymbol{u} - \nabla(a_1 m_1 + a_2 m_2 + a_3 m_3))^2 dx, \end{aligned}$$

最后一步转化为三角形的积分和.

可以看到, 误差的计算仅需要矩阵 $\boldsymbol{\Pi}_{k*}^\nabla$, 程序中用元胞数组 Ph 存储所有单元的结果 (带方向的问题后面会引入带符号的投影矩阵). 对第 i_{el} 个单元, $\boldsymbol{a} = (a_1, a_2, a_3)^T$ 如下计算

```

1 index = elem{iel};
2 Pis = Ph{iel}; chi = u(index);
3 a = Pis*chi;

```

对涉及到边上更多自由度的问题, 装配指标最好存起来, 以避免额外的计算, 程序中命名为 elem2dof .

给定精确解和尺度单项式

```

1 % exact solution
2 ue = pde.uexact; Due = pde.Du;
3 % scaled monomials
4 m1 = @(x,y) 1+0*x;
5 m2 = @(x,y) (x-xK)./hK;
6 m3 = @(x,y) (y-yK)./hK;
7 gradm1 = @(x,y) [0+0*x 0+0*x];
8 gradm2 = @(x,y) 1./hK*[1+0*x, 0+0*x];
9 gradm3 = @(x,y) 1./hK*[0+0*x, 1+0*x];

```

单元上被基函数及其梯度如下计算 (已经平方)

```

1 am = @(x,y) a(1)*m1(x,y)+a(2)*m2(x,y)+a(3)*m3(x,y);
2 agradm = @(x,y) a(1)*gradm1(x,y)+a(2)*gradm2(x,y)+a(3)*gradm3(x,y);
3 f = @(x,y) (ue([x,y])-am(x,y)).^2;
4 Df = @(x,y) (Due([x,y])-agradm(x,y)).^2;

```

这样, 单元上误差的平方为

```

1 % elementwise error
2 nodeT = [node(index,:);aux.centroid(iel,:)];
3 elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]';
4 ErrL2 = ErrL2 + integralTri(f,2,nodeT,elemT);
5 ErrH1 = ErrH1 + sum(integralTri(Df,2,nodeT,elemT));

```

计算 L^2 误差的过程可编写为一个通用函数, 只需要修改尺度单项式和数值自由度.

CODE 12. getL2error.m (计算 L^2 误差)

```

1 function [ErrL2,h] = getL2error(node,elem,Ph,chi,pde,kOrder)
2 % Ph: elementwise PIs
3 % chi: elementwise numerical d.o.f.s
4 % kOrder: VE space containing k-th order polynomials
5
6 k = kOrder; % Pk-VEM (k≤3)
7 Nm = (k+1)*(k+2)/2;
8 % exact solution
9 ue = pde.uexact;
10 % auxiliary data structure
11 aux = auxgeometry(node,elem); elem = aux.elem;
12 NT = size(elem,1);
13 ErrL2 = 0;
14 for iel = 1:NT
15     % element information
16     index = elem{iel}; Nv = length(index);
17     xK = aux.centroid(iel,1); yK = aux.centroid(iel,2); hK = aux.diameter(iel);
18     % scaled monomials
19     m1 = @(x,y) 1+0*x;
20     m2 = @(x,y) (x-xK)./hK;
21     m3 = @(x,y) (y-yK)./hK;
22     m4 = @(x,y) (x-xK).^2./hK.^2;
23     m5 = @(x,y) (x-xK).*(y-yK)./hK.^2;
24     m6 = @(x,y) (y-yK).^2./hK.^2;
25     m7 = @(x,y) (x-xK).^3./hK.^3;
26     m8 = @(x,y) (x-xK).^2.* (y-yK)./hK.^3;
27     m9 = @(x,y) (x-xK).* (y-yK).^2./hK.^3;
28     m10 = @(x,y) (y-yK).^3./hK.^3;
29     m = {m1,m2,m3,m4,m5,m6,m7,m8,m9,m10};

```

```

30
31 % vector a
32 Pis = Ph{iel}; a = Pis*chi{iel};
33 % integration
34 am = @ (x,y) 0+0*x;
35 for i = 1:Nm
36     am = @ (x,y) am(x,y) + a(i)*m{i}(x,y);
37 end
38 f = @ (x,y) (ue([x,y])-am(x,y)).^2;
39 % elementwise error
40 nodeT = [node(index,:); aux.centroid(iel,:)];
41 elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
42 ErrL2 = ErrL2 + integralTri(f,k+1,nodeT,elemT);
43 end
44 h = mean(aux.diameter);
45 ErrL2 = sqrt(ErrL2);

```

类似可给出 getH1error.m.

5.4.7 能量范数误差

我们也可以计算能量范数 $a_h(u - u_h, u - u_h)^{1/2}$. 同样地, 它不可计算, 下面尝试用插值 u_I 代替 u . 定义离散能量范数的相对误差为

$$\text{ErrI} = \frac{a_h(u_I - u_h, u_I - u_h)^{1/2}}{a_h(u_I, u_I)^{1/2}}.$$

现在给出显式表达式. 设整体基函数为 Φ_1, \dots, Φ_n , 则

$$u_I = \sum_{i=1}^n \chi_i(u) \Phi_i = \boldsymbol{\Phi}^T \boldsymbol{\chi}(u).$$

于是,

$$a_h(u_I, u_I) = \boldsymbol{\chi}(u)^T \mathbf{A} \boldsymbol{\chi}(u),$$

式中, $\mathbf{A} = (a_h(\Phi_i, \Phi_j))$ 是未施加边界条件的整体刚度矩阵. 这样, 分子为

$$\begin{aligned} a_h(u_I - u_h, u_I - u_h) &= \boldsymbol{\chi}(u_I - u_h)^T \mathbf{A} \boldsymbol{\chi}(u_I - u_h) \\ &= \boldsymbol{\chi}(u - u_h)^T \mathbf{A} \boldsymbol{\chi}(u - u_h). \end{aligned}$$

注意, $\boldsymbol{\chi}(u_h)$ 就是数值解. 于是

$$\text{E}_{\text{relative}} = \left(\frac{\boldsymbol{\chi}(u - u_h)^T \mathbf{A} \boldsymbol{\chi}(u - u_h)}{\boldsymbol{\chi}(u)^T \mathbf{A} \boldsymbol{\chi}(u)} \right)^{1/2}.$$

对 Poisson 方程的一阶问题, $\boldsymbol{\chi}(u)$ 就是所有顶点处的值, 我们要把刚度矩阵 kk 输出, 即 `info.kk = kk`. 为此, 离散能量范数的相对误差如下计算.

```

1 % discrete energy norm
2 kk = info.kk;
3 ue = pde.uexact(node);
4 ErrI = sqrt(abs((u-ue)'*kk*(u-ue))/abs(u'*kk*u))

```

注 5.6 对

$$\chi(v) = \int_e \partial_n v ds$$

这种类型的自由度, 它在边 e 两侧的单元上方向不同, 但整体上必须取定固定方向, 以作为最终的整体自由度. 我们规定: 整体方向取为边 e 的定向, 而边的定向规定满足起点序号小于终点序号, 即 $\text{edge}(:, 1) < \text{edge}(:, 2)$. 然而, 为了计算的方便, 我们通常保留边界的逆时针序号, 此时边界上的 e 不一定满足定向的规定. 为了解决这个问题, 我们在计算误差时, 去除边界条件给出的自由度 (注意 Neumann 边界也要去掉).

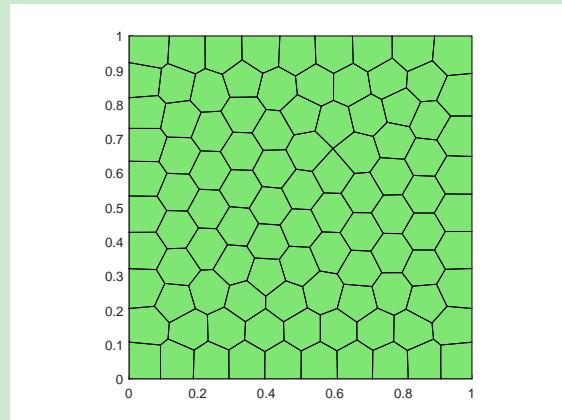
上面的误差计算可修改为

```
1 % discrete energy norm
2 kk = info.kk; freeDof = info.freeDof;
3 chie = pde.ueexact(node); chi = u;
4 kk = kk(freeDof, freeDof);
5 chie = chie(freeDof); chi = chi(freeDof);
6 ErrI = sqrt(abs((chi-chie)'*kk*(chi-chie))/abs(chie'*kk*chie))
```

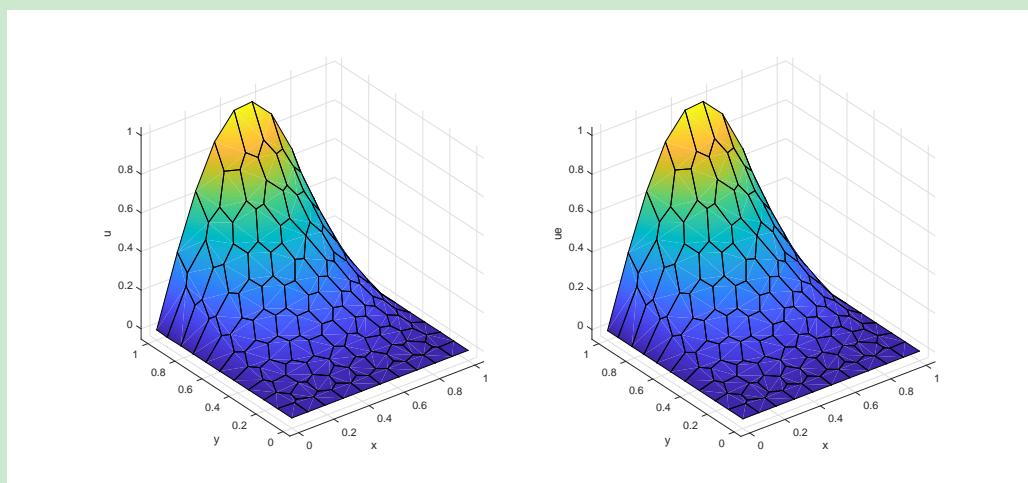
5.4.8 程序整理

例 5.1 设精确解为 $u(x, y) = y^2 \sin(\pi x)$, $\Omega = (0, 1)^2$.

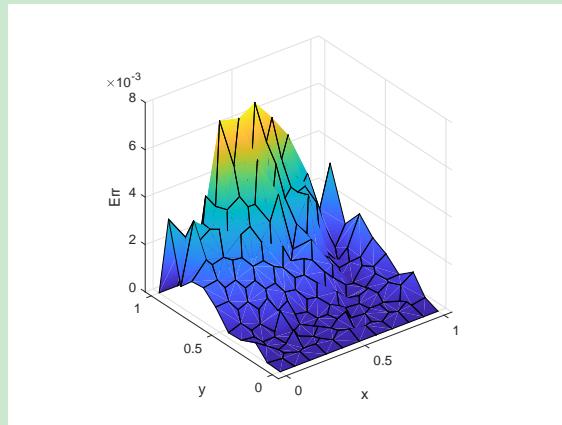
网格剖分由 tool 文件夹下的 meshfun.m 生成, 保存为 Poissonmesh.mat, 网格图示如下



设区域的左侧和右侧为 Neumann 边界, 数值解和精确解的图像如下



绝对误差为



主程序为

CODE 13. main_PoissonVEM.m

```
1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [200, 400, 600, 800, 1000];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10 ErrI = zeros(maxIt,1);
11
12 %% PDE data
13 c = 1;
14 pde = Poissonsdata1(c);
15 %bdNeumann = 'abs(x-0)<1e-4 | abs(x-1)<1e-4'; % string for Neumann
16 bdNeumann = [];
17
18 %% Virtual element method
19 for k = 1:maxIt
20     % load mesh
21     load( [ 'meshdata' , num2str(nameV(k)) , '.mat' ] );
22     % get boundary information
23     bdStruct = setboundary(node,elem,bdNeumann);
24     % solve
25     [u,info] = PoissonVEM(node,elem,pde,bdStruct);
26     % record and plot
27     N(k) = length(u); h(k) = 1/sqrt(size(elem,1));
28     if size(elem,1)<2e3
29         figure(1); clf;
30         subplot(1,2,1), showmesh(node,elem);
31         subplot(1,2,2), showsolution(node,elem,u);
32     end
33     % compute errors in discrete L2 and H1 norms
34     index = info.elem2dof; % elemenwise global index
35     chi = cellfun(@(id) u(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
36     kOrder = 1;
37     Ph = info.Ph;
```

```

38 ErrL2(k) = getL2error(node, elem, Ph, chi, pde, kOrder);
39 ErrH1(k) = getH1error(node, elem, Ph, chi, pde, kOrder);
40 % compute errors in discrete energy norm
41 kk = info.kk; freeDof = info.freeDof;
42 chie = pde.uxexact(node); chi = u;
43 kk = kk(freeDof, freeDof);
44 chie = chie(freeDof); chi = chi(freeDof);
45 ErrI(k) = sqrt(abs((chi-chie)'*kk*(chi-chie)));
46 end
47
48 %% Plot convergence rates and display error table
49 figure(2);
50 showrateh(h, ErrL2, ErrH1, ErrI);
51
52 fprintf('\n');
53 disp('Table: Error')
54 colname = {'#Dof', 'h', '|u-u_h|_1', '|u-u_h|_E'};
55 disptable(colname, N, [], h, '%0.3e', ErrL2, '%0.5e', ErrH1, '%0.5e', ErrI, '%0.5e');

```

函数文件为

CODE 14. PoissonVEM.m

```

1 function [u,info] = PoissonVEM(node,elem,pde,bdStruct)
2 %PoissonVEM solves Poisson equation using virtual element method in V1
3 %
4 % -\Delta u = f, in Omega
5 % Dirichlet boundary condition u=g_D on \Gamma_D,
6 % Neumann boundary condition grad(u)*n=g_N on \Gamma_N
7 %
8 % Copyright (C) Terence Yu.
9
10 %% Get auxiliary data
11 aux = auxgeometry(node,elem);
12 node = aux.node; elem = aux.elem;
13 N = size(node,1); NT = size(elem,1);
14
15 %% Compute projection matrices
16 D = cell(NT,1); Bs = cell(NT,1);
17 G = cell(NT,1); Gs = cell(NT,1); H = cell(NT,1);
18 for iel = 1:NT
19     % element information
20     index = elem{iel}; Nv = length(index);
21     xK = aux.centroid(iel,1); yK = aux.centroid(iel,2);
22     hK = aux.diameter(iel);
23     x = node(index,1); y = node(index,2);
24     % scaled monomials
25     m1 = @(x,y) 1 + 0*x; m2 = @(x,y) (x-xK)./hK; m3 = @(x,y) (y-yK)./hK;
26     m = @(x,y) [m1(x,y), m2(x,y), m3(x,y)];
27     Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
28     % D
29     D1 = m(x,y); D{iel} = D1;
30     % B, Bs, G, Gs
31     rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
32     normVec = 0.5*[y(rotid2)-y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of edge ...
33     % vector
34     B1 = Gradm*normVec; % B
35     B1s = B1; B1s(1,:) = 1/Nv; Bs{iel} = B1s;

```

```

35     G{iel} = B1*D1;      Gs{iel} = B1s*D1;
36
37 % H
38 nodeT = [node(index,:);aux.centroid(iel,:)];
39 elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
40
41 mm = @ (x,y) [m1(x,y).*m1(x,y), m1(x,y).*m2(x,y), m1(x,y).*m3(x,y), ...
42             m2(x,y).*m1(x,y), m2(x,y).*m2(x,y), m2(x,y).*m3(x,y), ...
43             m3(x,y).*m1(x,y), m3(x,y).*m2(x,y), m3(x,y).*m3(x,y)];
44 H1 = zeros(3,3);
45 H1(:) = integralTri(mm,3,nodeT,elemT); % n = 2
46 H{iel} = H1;
47
48 %% Get elementwise stiffness matrix and load vector
49 ABelem = cell(NT,1); belem = cell(NT,1);
50 Ph = cell(NT,1); % matrix for error evaluation
51 for iel = 1:NT
52
53     % Projection
54     Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi));
55
56     % Stiffness matrix
57     AK = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
58     BK = pde.c*Pis'*H{iel}*Pis + pde.c*hK^2*(I-Pi)'*(I-Pi); % Pis = Pi0s;
59     ABelem{iel} = reshape(AK'+BK',1,[]); % straighten as row vector for easy assembly
60
61     % Load vector
62     fK = Pis'*[pde.f(aux.centroid(iel,:))*aux.area(iel);0;0];
63     belem{iel} = fK'; % straighten as row vector for easy assembly
64
65     % matrix for error evaluation
66     Ph{iel} = Pis;
67
68
69 %% Assemble stiffness matrix and load vector
70 elemLen = cellfun('length',elem); vertNum = unique(elemLen);
71 nnz = sum(elemLen.^2);
72 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
73 id = 0; ff = zeros(N,1);
74 elem2dof = cell(NT,1);
75 for Nv = vertNum(:)' % only valid for row vector
76
77     % assemble the matrix
78     idNv = find(elemLen == Nv); % find polygons with Nv vertices
79     NTv = length(idNv); % number of elements with Nv vertices
80
81     elemNv = cell2mat(elem(idNv)); % elem
82     K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
83     s = 1; Ndof = Nv;
84     for i = 1:Ndof
85         for j = 1:Ndof
86             ii(id+1:id+NTv) = elemNv(:,i); % zi
87             jj(id+1:id+NTv) = elemNv(:,j); % zj
88             ss(id+1:id+NTv) = K(:,s);
89             id = id + NTv; s = s+1;
90         end
91     end
92
93     % assemble the vector
94     ff = ff + accumarray(elemNv(:,),F(:,',[N 1]));
95
96     % elementwise global indices

```

```

93     elem2dof(idNv) = mat2cell(elemNv, ones(NTv,1), Ndof);
94 end
95 kk = sparse(ii,jj,ss,N,N);
96
97 %% Assemble Neumann boundary conditions
98 bdEdgeN = bdStruct.bdEdgeN;
99 if ~isempty(bdEdgeN)
100    g_N = pde.Du;
101    z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
102    e = z1-z2; % e = z2-z1
103    Ne = [-e(:,2),e(:,1)]; % scaled ne
104    F1 = sum(Ne.*g_N(z1),2)./2;
105    F2 = sum(Ne.*g_N(z2),2)./2;
106    FN = [F1,F2];
107    ff = ff + accumarray(bdEdgeN(:,1), FN(:,1));
108 end
109
110 %% Apply Dirichlet boundary conditions
111 g_D = pde.g_D; bdNodeIdx = bdStruct.bdNodeIdx;
112 isBdNode = false(N,1); isBdNode(bdNodeIdx) = true;
113 bddof = find(isBdNode); freedof = find(~isBdNode);
114 pD = node(bddof,:);
115 u = zeros(N,1); uD = g_D(pD); u(bddof) = uD(:);
116 ff = ff - kk*u;
117
118 %% Set solver
119 solver = 'amg';
120 if N < 2e3, solver = 'direct'; end
121 % solve
122 switch solver
123 case 'direct'
124     u(freedof) = kk(freedof, freedof)\ff(freedof);
125 case 'amg'
126     option.solver = 'CG';
127     u(freedof) = amg(kk(freedof, freedof), ff(freedof), option);
128 end
129
130 %% Store information for computing errors
131 info.Ph = Ph; info.elem2dof = elem2dof;
132 info.kk = kk; info.freeDof = freedof;

```

注 5.7 结构体 `info` 中存储了两个信息: `Ph` 和 `elem2dof`, 它们在计算 H^1 和 L^2 等误差中用到.

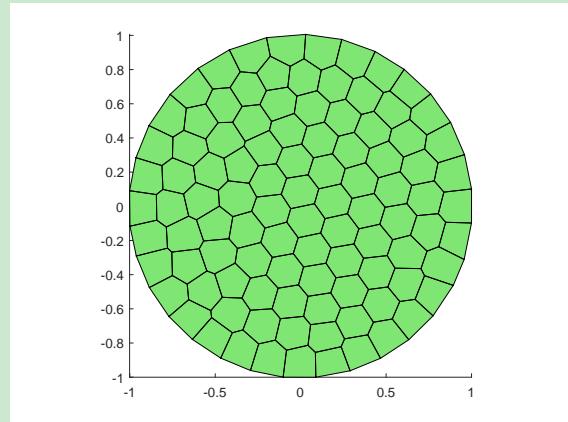
注 5.8 涉及到的其他函数文件见 GitHub. 上面的主程序也可直接改为其他区域的例子. 例如, 我们给出了单位圆的剖分数据 `PoissonmeshCircle.m`, 可规定上半圆部分为 Neumann 边界:

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

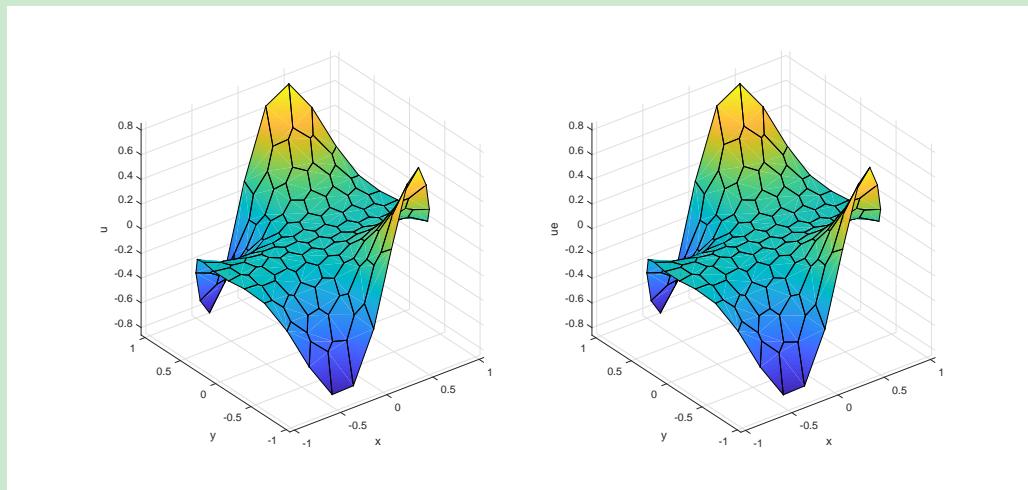
```
bdNeumann = 'y>=0'; %string for Neumann (circle)
```

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

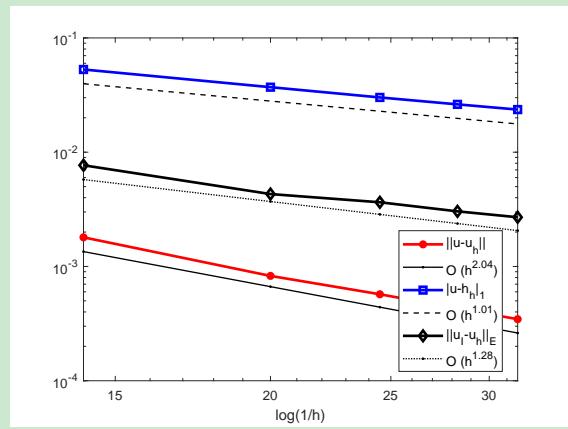
网格剖分如下



数值解与精确解为



各种误差阶的结果如下图.



可以看到, L^2 误差的阶为 2, H^1 误差的阶为 1, 能量误差的阶为 1.

5.5 Poisson 方程的二阶虚拟元方法

5.5.1 装配指标

对 $k = 2$, 边 e 内要增加一个函数值, 单元上要增加一个零阶矩量, 自由度如下图排列.

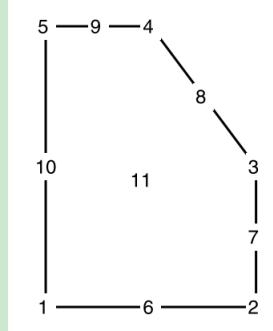


图 11. $k = 2$ 时 VEM 的自由度排列

在辅助数据结构中, 我们给出了与边相关的两个数据 `edge` 和 `elem2edge`:

- `elem2edge` 按单元记录边的自然序号, 且顺序与图 11 一致. 只需在这些序号上加上 `N` 即可.
- 单元 K 的自由度编号只要在 $(1:N_T)$ 上加上 $N+NE$ 即可, 其中 NE 是一维边的个数, 即 `edge` 的行数.

我们是按照同种构型装配的, 装配指标如下给出

```

1 load meshhex1.mat;
2
3 aux = auxstructure(node, elem);
4 elem2edge = aux.elem2edge; edge = aux.edge;
5
6 N = size(node, 1); NT = size(elem, 1); NE = size(edge, 1);
7
8 elemLen = cellfun('length', elem); vertNum = unique(elemLen);
9 nnz = sum((2*elemLen+1).^2);
10 ii = zeros(nnz, 1); jj = zeros(nnz, 1); ss = zeros(nnz, 1);
11
12 id = 0; ff = zeros(N, 1);
13 for Nv = vertNum(:) % only valid for row vector
14
15     idNv = find(elemLen == Nv); % find polygons with Nv vertices
16     NTv = length(idNv); % number of elements with Nv vertices
17
18     elemNv = cell2mat(elem(idNv)); % elem
19     elem2edgeNv = cell2mat(elem2edge(idNv));
20     elem2 = [elemNv, elem2edgeNv+N, idNv+N+NE];
21     Ndof = 2*Nv+1;
22     ii(id+1:id+NTv*Ndof^2) = reshape(repmat(elem2, Ndof, 1), [], 1);
23     jj(id+1:id+NTv*Ndof^2) = repmat(elem2(:, ), Ndof, 1);
24     ss(id+1:id+NTv*Ndof^2) = K(:, );
25     id = id + NTv*Ndof^2;
26 end

```

5.5.2 投影矩阵的计算

先考虑过渡矩阵. 对 $k = 2$, 自由度为

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v,$$

$$\chi_{N_v+i}(v) = v(z_{i+1/2}), \quad i = 1, \dots, N_v,$$

$$\chi_{2N_v+1}(v) = \frac{1}{|K|} \int_K v dx,$$

其中, z_i 为单元顶点, $z_{i+1/2}$ 为边的中点. 尺度单项式集合 m^T 由如下元素组成:

$$m_1(x, y) = 1, \quad m_2(x, y) = \frac{x - x_K}{h_K}, \quad m_3(x, y) = \frac{y - y_K}{h_K},$$

$$m_4(x, y) = \frac{(x - x_K)^2}{h_K^2}, \quad m_5(x, y) = \frac{(x - x_K)(y - y_K)}{h_K^2}, \quad m_6(x, y) = \frac{(y - y_K)^2}{h_K^2}.$$

由此可计算

$$D = (D_{i\alpha})_{N_v \times 6}, \quad D_{i,\alpha} = \chi_i(m_\alpha).$$

类似前面的处理, 最后一个自由度的积分转化为三角剖分上的计算.

```

1      % transition matrix
2      nodeT = [node(index,:);centroid(iel,:)];
3      elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]';
4      D1 = zeros(Ndof,6);
5      D1(1:Nv,:) = m(x,y);
6      D1(Nv+1:2*Nv,:) = m(xe,ye);
7      D1(end,:) = 1/area(iel)*integralTri(m,3,nodeT,elemT);
8      D{iel} = D1;

```

接着计算 B . 由 (5.12),

$$B = \int_K \nabla m \cdot \nabla \phi^T dx = - \int_K \Delta m \cdot \phi^T dx + \sum_{e \subset \partial K} \int_e (\nabla m \cdot \mathbf{n}_e) \phi^T ds.$$

为了方便, 记

$$I_1 = \int_K \Delta m \cdot \phi^T dx, \quad I_2 = \sum_{e \subset \partial K} \int_e (\nabla m \cdot \mathbf{n}_e) \phi^T ds.$$

对第一项, 注意到 Δm 是常向量, 由 Kronecker 性质,

$$I_1 = \int_K \Delta m \cdot \phi^T dx = \Delta m \cdot \int_K \phi^T dx = \Delta m \cdot [\mathbf{0}, \mathbf{0}, |K|],$$

其中,

$$\Delta m = \left[0, 0, 0, \frac{2}{h_K^2}, 0, \frac{2}{h_K^2} \right]^T.$$

对第二项, 记 $\mathbf{f} = (\nabla m \cdot \mathbf{n}_e) \phi^T$, 显然它在每条边上都是三次多项式, 右端的积分可用 Simpson 公式精确计算, 于是

$$I_2 = \frac{1}{6} \sum_{i=1}^{N_v} h_{e_i} [\mathbf{f}(z_i) + 4\mathbf{f}(z_{i+1/2}) + \mathbf{f}(z_{i+1})],$$

其中,

$$\begin{aligned} \mathbf{f}(z_i) &= \partial_{\mathbf{n}} m(z_i)[\mathbf{e}_i, \mathbf{0}, 0], \\ \mathbf{f}(z_{i+1/2}) &= \partial_{\mathbf{n}} m(z_{i+1/2})[\mathbf{0}, \mathbf{e}_i, 0], \\ \mathbf{f}(z_{i+1}) &= \partial_{\mathbf{n}} m(z_{i+1})[\mathbf{e}_{i+1}, \mathbf{0}, 0]. \end{aligned}$$

这里, \mathbf{e}_i 是长度为 N_v 的第 i 个自然向量, $\mathbf{0}$ 是长度为 N_v 的全零向量.

```

1 % ----- elliptic projection -----
2      % first term
3      Lapm = zeros(6,1); Lapm([4,6]) = 2/hK^2;

```

```

4      Dof = [zeros(1,2*Nv), area(iel)];
5      I1 = Lapm*Dof;
6      % second term
7      I2 = 0;
8      for j = 1:Nv % loop of edges
9          % he*\partial_n(m) at x_j, xe, x_j+1
10         DnL = Gradm(x(j),y(j))*Ne(j,:)';
11         Dne = Gradm(xe(j),ye(j))*Ne(j,:)';
12         DnR = Gradm(x(v2(j)),y(v2(j)))*Ne(j,:)';
13         % [ei,0,0]
14         e1 = zeros(1,Ndof); e1(j) = 1;
15         e2 = zeros(1,Ndof); e2(Nv+j) = 1;
16         e3 = zeros(1,Ndof); e3(v2(j)) = 1;
17         % f
18         f1 = DnL*e1; f2 = Dne*e2; f3 = DnR*e3;
19         I2 = I2 + (f1+4*f2+f3);
20     end
21     B1 = -I1 + 1/6*I2;

```

投影的限制条件由 (5.11) 给出, B_1 的第一行将用 $P_0(\phi^T)$ 代替, 其中

$$P_0(v) = \int_K v dx,$$

这在 I_1 的计算中已给出.

```

1      % constraint
2      B1s = B1;   B1s(1,:) = Dof;   Bs{iel} = B1s;
3      G{iel} = B1*D1;       Gs{iel} = B1s*D1;

```

L^2 投影对应的矩阵 H 按照注 5.2 的说明进行计算.

```

1      m = {m1,m2,m3,m4,m5,m6};
2      H1 = zeros(6,6);
3      for i = 1:6
4          for j = 1:6
5              fun = @(x,y) m{i}(x,y).*m{j}(x,y);
6              H1(i,j) = integralTri(fun,2,nodeT,elemT);
7          end
8      end
9      H{iel} = H1;

```

注意对 $k = 2$, 在 $W_k(K)$ 中有 $\Pi_k^\nabla = \Pi_k^0$.

5.5.3 单元刚度矩阵和载荷向量的计算

与 $k = 1$ 类似, 程序如下

```

1  % ----- Elementwise stiffness matrix and load vector -----
2 ABelem = cell(NT,1); belem = cell(NT,1);
3 Ph = cell(NT,1); % matrix for error evaluation
4 for iel = 1:NT
5     % element information
6     index = elem{iel};    Nv = length(index);
7     xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
8     nodeT = [node(index,:);centroid(iel,:)];
9     elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
10    % Projection
11    Pis = Gs{iel}\Bs{iel};    Pi = D{iel}*Pis;    I = eye(size(Pi));

```

```

12 % Stiffness matrix
13 AK = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
14 BK = pde.c*Pis'*H{iel}*Pis + pde.c*hK^2*(I-Pi)'*(I-Pi); % Pis = Pi0s;
15 ABelem{iel} = reshape(AK'+BK',1,[]); % straighten as row vector for easy assembly
16 % Load vector
17 m1 = @ (x,y) 1+0*x;
18 m2 = @ (x,y) (x-xK)./hK;
19 m3 = @ (x,y) (y-yK)./hK;
20 m4 = @ (x,y) (x-xK).^2/hK^2;
21 m5 = @ (x,y) (x-xK).* (y-yK)./hK^2;
22 m6 = @ (x,y) (y-yK).^2./hK^2;
23 m = @ (x,y) [m1(x,y), m2(x,y), m3(x,y), m4(x,y), m5(x,y), m6(x,y)];
24 fm = @ (x,y) repmat(pde.f([x,y]),1,6).*m(x,y); % f(p) = f([x,y])
25 rhs = integralTri(fm,3,nodeT,elemT); rhs = rhs';
26 fK = Pis'*rhs;
27 belem{iel} = fK'; % straighten as row vector for easy assembly
28 % matrix for error evaluation
29 Ph{iel} = Pis;
30 end

```

5.5.4 刚度矩阵和载荷向量的装配

与 $k = 1$ 类似, 程序如下

```

1 % ----- Assemble the stiffness matrix and load vector -----
2 elemLen = cellfun('length', elem); vertNum = unique(elemLen);
3 nnz = sum((2*elemLen+1).^2);
4 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
5 id = 0; ff = zeros(NNdof,1);
6 IndexDof = cell(NT,1);
7 for Nv = vertNum(:) % only valid for row vector
8
9     idNv = find(elemLen == Nv); % find polygons with Nv vertices
10    NTv = length(idNv); % number of elements with Nv vertices
11
12    elemNv = cell2mat(elem(idNv)); % elem
13    elem2edgeNv = cell2mat(elem2edge(idNv));
14    elem2 = [elemNv, elem2edgeNv+N, idNv+N+NE];
15    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
16    Ndof = 2*Nv+1;
17    ii(id+1:id+NTv*Ndof^2) = reshape(repmat(elem2, Ndof, 1), [], 1);
18    jj(id+1:id+NTv*Ndof^2) = repmat(elem2(:, ), Ndof, 1);
19    ss(id+1:id+NTv*Ndof^2) = K(:, );
20    id = id + NTv*Ndof^2;
21
22    % assemble the vector
23    ff = ff + accumarray(elem2(:, ), F(:, ), [NNdof 1]);
24
25    % elementwise global indices
26    IndexDof(idNv) = mat2cell(elem2, ones(NTv,1), Ndof);
27 end
28 kk = sparse(ii,jj,ss,NNdof,NNdof);

```

5.5.5 边界条件的处理

考虑 Neumann 边界条件. 设 e 是 Neumann 边, $u|_e$ 是二次多项式, 它可表为 $u = u_1\phi_1 + u_2\phi_2 +$

$u_3\phi_3$. 这里, ϕ_1, ϕ_2, ϕ_3 分别对应 e 的左右端点和中点. 载荷向量为

$$F_e = [F_1, F_2, F_3]^T, \quad F_i = \int_e \partial_n u \phi_i ds.$$

用 Simpson 公式近似, 有

$$\begin{aligned} F_1 &= \frac{h_e}{6} (\partial_n u(z_1)\phi_1(z_1) + \partial_n u(z_2)\phi_1(z_2) + 4\partial_n u(z_3)\phi_1(z_3)) = \frac{h_e}{6}\partial_n u(z_1), \\ F_2 &= \frac{h_e}{6} (\partial_n u(z_1)\phi_2(z_1) + \partial_n u(z_2)\phi_2(z_2) + 4\partial_n u(z_3)\phi_2(z_3)) = \frac{h_e}{6}\partial_n u(z_2), \\ F_3 &= \frac{h_e}{6} (\partial_n u(z_1)\phi_3(z_1) + \partial_n u(z_2)\phi_3(z_2) + 4\partial_n u(z_3)\phi_3(z_3)) = \frac{4h_e}{6}\partial_n u(z_3). \end{aligned}$$

类似一阶问题的处理, Neumann 条件如下装配.

```

1 %% Assemble Neumann boundary conditions
2 bdEdgeN = bdStruct.bdEdgeN; bdEdgeIdxN = bdStruct.bdEdgeIdxN;
3 if ~isempty(bdEdgeN)
4     g_N = pde.Du;
5     z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:); zc = (z1+z2)/2;
6     e = z1-z2; % e = z2-z1
7     Ne = [-e(:,2), e(:,1)]; % scaled ne
8     F1 = sum(Ne.*g_N(z1),2)/6;
9     F2 = sum(Ne.*g_N(z2),2)/6;
10    F3 = sum(Ne.*g_N(zc),2)*4/6;
11    FN = [F1,F2,F3];
12    bdEdgeN2 = [bdEdgeN, bdEdgeIdxN+N];
13    ff = ff + accumarray(bdEdgeN2(:,1), FN(:,1), [NNdof 1]);
14 end

```

这里, $bdEdgeN$ 记录 Neumann 边的起点和中点序号, 即一维连通性信息, $bdEdgeIdx$ 是 Neumann 边的自然序号.

Dirichlet 边界如下处理

```

1 %% Apply Dirichlet boundary conditions
2 % boundary information
3 g_D = pde.g_D;
4 bdNodeIdx = bdStruct.bdNodeIdx;
5 bdEdgeD = bdStruct.bdEdgeD;
6 bdEdgeIdxD = bdStruct.bdEdgeIdxD;
7 id = [bdNodeIdx; bdEdgeIdxD+N];
8 isBdNode = false(NNdof,1); isBdNode(id) = true;
9 bdDof = (isBdNode); freeDof = (~isBdNode);
10 % vertices on the boundary
11 pD = node(bdNodeIdx,:); uD = g_D(pD);
12 % mid-edge on the boundary
13 z1 = node(bdEdgeD(:,1),:); z2 = node(bdEdgeD(:,2),:); zc = (z1+z2)./2;
14 uDc = g_D(zc);
15 % rhs
16 u = zeros(NNdof,1); u(bdDof) = [uD; uDc];
17 ff = ff - kk*u;

```

这里, $bdEdgeD$ 是 Dirichlet 边的连通性信息, $bdEdgeIdxD$ 是 Dirichlet 边的自然序号.

5.5.6 误差分析

与一阶虚拟元类似, 主程序如下

```

1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [100, 200, 300, 400, 500];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10 ErrI = zeros(maxIt,1);
11
12 %% PDE data
13 c = 1e3;
14 pde = Poissondata1(c);
15 %bdNeumann = 'abs(x-0)<1e-4 | abs(x-1)<1e-4'; % string for Neumann
16 bdNeumann = [];
17
18 %% Virtual element method
19 for k = 1:maxIt
20     % load mesh
21     load( ['meshdata', num2str(nameV(k)), '.mat'] );
22     % get boundary information
23     bdStruct = setboundary(node, elem, bdNeumann);
24     % solve
25     [u,info] = PoissonVEMk2(node, elem, pde, bdStruct);
26     % record and plot
27     N(k) = length(u); h(k) = 1/sqrt(size(elem,1));
28     if size(elem,1)<2e3
29         figure(1); clf;
30         subplot(1,2,1), showmesh(node, elem);
31         subplot(1,2,2), showsolution(node, elem, u(1:size(node,1)));
32     end
33     % compute errors in discrete L2 and H1 norms
34     index = info.elem2dof; % elementwise global index
35     chi = cellfun(@(id) u(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
36     kOrder = 2;
37     Ph = info.Ph;
38     ErrL2(k) = getL2error(node, elem, Ph, chi, pde, kOrder);
39     ErrH1(k) = getH1error(node, elem, Ph, chi, pde, kOrder);
40     % compute errors in discrete energy norm
41     % auxgeometry
42     aux = auxgeometry(node, elem);
43     % auxstructure
44     auxT = auxstructure(node, elem);
45     edge = auxT.edge;
46     % chi1: evaluation at all vertices
47     uexact = pde.uexact;
48     chi1 = uexact(node);
49     % chi2: evaluation at all mid-edge points
50     z1 = node(edge(:,1,:)); z2 = node(edge(:,2,:)); zc = (z1+z2)/2;
51     chi2 = uexact(zc);
52     % chi3: moments on element K
53     NT = size(elem,1);
54     chi3 = zeros(NT,1);
55     fun = @(x,y) uexact([x,y]);
56     for iel = 1:NT

```

```

57     index = elem{iel};      Nv = length(index);      Ndof = 2*Nv+1;
58     nodeT = [node(index,:);aux.centroid(iel,:)];
59     elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
60     chi3(iel) = 1/aux.area(iel)*integralTri(fun,3,nodeT,elemT);
61 end
62 % errors in energy norm
63 kk = info.kk; freeDof = info.freeDof;
64 kk = kk(freeDof,freeDof);
65 chie = [chi1; chi2; chi3]; chi = u;
66 chie = chie(freeDof); chi = chi(freeDof);
67 ErrI(k) = sqrt(abs((chi-chie)'*kk*(chi-chie))/abs(chie'*kk*chie));
68 end
69
70 %% Plot convergence rates and display error table
71 figure(2);
72 showrateh(h,ErrL2,ErrH1,ErrI);
73
74 fprintf('\n');
75 disp('Table: Error')
76 colname = {'#Dof','h','|||u-u_h|||','|u-u_h|_1','|||u_I-u_h|||_E'};
77 disptable(colname,N,[],h,'%0.3e',ErrL2,'%0.5e',ErrH1,'%0.5e',ErrI,'%0.5e');

```

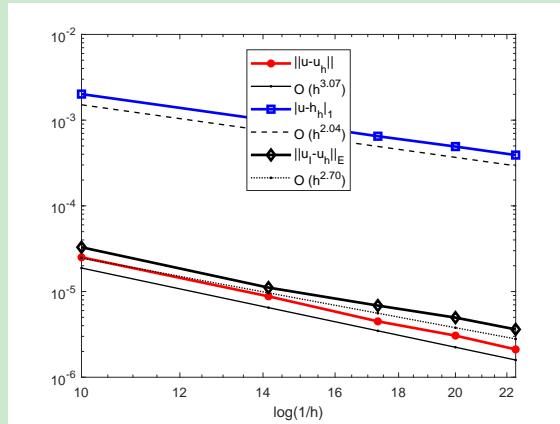


图 12. Poisson 方程二阶虚拟元方法的误差阶

可以看到, L^2 的误差阶为 3, H^1 的误差阶为 2, 而能量误差的误差阶基本上也为 3.

5.6 Poisson 方程的三阶虚拟元方法

5.6.1 椭圆投影的计算

对 $k = 3$, 边 e 内要增加 2 个函数值, 单元上要增加零阶和一阶矩量. 自由度排列如下:

- 顶点处的函数值,

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v;$$

- 边上第一个内部点的函数值,

$$\chi_{N_v+i}(v) = v(a_i), \quad i = 1, \dots, N_v;$$

- 边上第二个内部点的函数值,

$$\chi_{2N_v+i}(v) = v(b_i), \quad i = 1, \dots, N_v;$$

- 单元 K 上的零阶和一阶矩量,

$$\chi_{3N_v+1}(v) = \frac{1}{|K|} \int_K m_1 v dx, \quad m_1 = 1,$$

$$\begin{aligned}\chi_{3N_v+2}(v) &= \frac{1}{|K|} \int_K m_2 v dx, \quad m_2 = \frac{x - x_K}{h_K}, \\ \chi_{3N_v+3}(v) &= \frac{1}{|K|} \int_K m_3 v dx, \quad m_3 = \frac{y - y_K}{h_K}.\end{aligned}$$

- 这样, 局部有 $3N_v + 3$ 个自由度.

为了精确计算边界积分, 边 e_i 上的四个点 z_i, a_i, b_i, z_{i+1} 要形成 Gauss-Lobatto 点. 查找相关资料知, 对 $[-1, 1]$, 四个点的求积公式如下

$$\int_{-1}^1 f(r) dr \approx w_1 f(r_1) + w_2 f(r_2) + w_3 f(r_3) + w_4 f(r_4),$$

其中,

```
1 r = [-1, -1/sqrt(5), 1/sqrt(5), 1];
2 w = [1/6, 5/6, 5/6, 1/6];
```

变换到指定区间 $[a, b]$ 上后, 内部的两个点如下计算

$$s(r) = \frac{b-a}{2}r + \frac{b+a}{2}, \quad r \in [-1, 1],$$

从而 $ds = (b-a)/2dr$. 节点 a_i, b_i 可根据线段的比例获得.

```
1 r = 0.5*(r(2:3)+1); % [0,1]: gives the ratios of interior nodes
2 index = elem{iel};
3 x = node(index,1); y = node(index,2); z = [x,y];
4 % Gauss-Lobatto
5 za = z(v1,:)+r(1)*(z(v2,:)-z(v1,:));
6 zb = z(v1,:)+r(2)*(z(v2,:)-z(v1,:));
```

设新的积分点为 s_1, s_2, s_3, s_4 , 则

$$\int_e f(s) ds = \frac{h_e}{2} (w_1 f(s_1) + w_2 f(s_2) + w_3 f(s_3) + w_4 f(s_4)). \quad (5.20)$$

先计算过渡矩阵 D . 尺度单项式集合 m 的元素共 10 个, 分别为

$$\begin{aligned}m_1 &= 1, & m_2 &= \frac{x - x_K}{h_K}, & m_3 &= \frac{y - y_K}{h_K}, \\ m_4 &= \frac{(x - x_K)^2}{h_K^2}, & m_5 &= \frac{(x - x_K)(y - y_K)}{h_K^2}, & m_6 &= \frac{(y - y_K)^2}{h_K^2}, \\ m_7 &= \frac{(x - x_K)^3}{h_K^3}, & m_8 &= \frac{(x - x_K)^2(y - y_K)}{h_K^3}, & m_9 &= \frac{(x - x_K)(y - y_K)^2}{h_K^3}, & m_{10} &= \frac{(y - y_K)^3}{h_K^3}.\end{aligned}$$

由此可计算

$$D = (D_{i\alpha})_{N_{dof} \times 10}, \quad D_{i,\alpha} = \chi_i(m_\alpha).$$

```

1 % ----- transition matrix -----
2 nodeT = [node(index,:);centroid(iel,:)];
3 elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]';
4 D1 = zeros(Ndof,10);
5 D1(1:Nv,:) = m(x,y); % at zi
6 D1(Nv+1:2*Nv,:) = m(za(:,1), za(:,2)); % at ai
7 D1(2*Nv+1:3*Nv,:) = m(zb(:,1), zb(:,2)); % at bi
8 m1v = @ (x,y) repmat(m1(x,y),1,10).*m(x,y); % moment 1
9 m2v = @ (x,y) repmat(m2(x,y),1,10).*m(x,y); % moment 2
10 m3v = @ (x,y) repmat(m3(x,y),1,10).*m(x,y); % moment 3
11 D1(3*Nv+1,:) = 1/area(iel)*integralTri(m1v,4,nodeT,elemT);
12 D1(3*Nv+2,:) = 1/area(iel)*integralTri(m2v,4,nodeT,elemT);
13 D1(3*Nv+3,:) = 1/area(iel)*integralTri(m3v,4,nodeT,elemT);
14 D{iel} = D1;

```

矩量部分的计算也可循环处理 (后面 L^2 投影的计算需要所有 m_j 对应的矩量)

```

1 for j = 1:3
2     mjv = @ (x,y) repmat(mm{j}(x,y),1,10).*m(x,y); % moment j on K
3     D1(3*Nv+j,:) = 1/area(iel)*integralTri(mjv,4,nodeT,elemT);
4 end

```

接着计算 B . 由 (5.12),

$$B = \int_K \nabla m \cdot \nabla \phi^T dx = - \int_K \Delta m \cdot \phi^T dx + \sum_{e \subset \partial K} \int_e (\nabla m \cdot \mathbf{n}_e) \phi^T ds.$$

为了方便, 记

$$I_1 = \int_K \Delta m \cdot \phi^T dx, \quad I_2 = \sum_{e \subset \partial K} \int_e (\nabla m \cdot \mathbf{n}_e) \phi^T ds.$$

对第一项, 直接计算有

$$\Delta m = \left[0, 0, 0, \frac{2}{h_K^2}, 0, \frac{2}{h_K^2}, \frac{6(x-x_K)}{h_K^3}, \frac{2(y-y_K)}{h_K^3}, \frac{2(x-x_K)}{h_K^3}, \frac{6(y-y_K)}{h_K^3} \right]^T,$$

于是由 Kronecker 性质,

$$\begin{aligned} I_1(i,:) &= \int_K \Delta m_i \cdot \phi^T dx = \mathbf{0}_{N_{\text{dof}}}, \quad i = 1, 2, 3, 5, \\ I_1(i,:) &= \frac{2}{h_K^2} \int_K \phi^T dx = \frac{2}{h_K^2} |K| [\mathbf{0}_{N_v}, \mathbf{0}_{N_v}, \mathbf{0}_{N_v}, 1, 0, 0], \quad i = 4, 6, \\ I_1(7,:) &= \frac{6}{h_K^2} \int_K m_2 \phi^T dx = \frac{6}{h_K^2} |K| [\mathbf{0}_{N_v}, \mathbf{0}_{N_v}, \mathbf{0}_{N_v}, 0, 1, 0], \\ I_1(8,:) &= \frac{2}{h_K^2} \int_K m_3 \phi^T dx = \frac{2}{h_K^2} |K| [\mathbf{0}_{N_v}, \mathbf{0}_{N_v}, \mathbf{0}_{N_v}, 0, 0, 1], \\ I_1(9,:) &= \frac{2}{h_K^2} \int_K m_2 \phi^T dx = \frac{2}{h_K^2} |K| [\mathbf{0}_{N_v}, \mathbf{0}_{N_v}, \mathbf{0}_{N_v}, 0, 1, 0], \\ I_1(10,:) &= \frac{6}{h_K^2} \int_K m_3 \phi^T dx = \frac{6}{h_K^2} |K| [\mathbf{0}_{N_v}, \mathbf{0}_{N_v}, \mathbf{0}_{N_v}, 0, 0, 1]. \end{aligned}$$

```

1 % first term
2 I1 = zeros(10,Ndof);
3 I1([4,6],3*Nv+1) = 2*area(iel)/hK^2;
4 I1(7,3*Nv+2) = 6*area(iel)/hK^2;
5 I1(8,3*Nv+3) = 2*area(iel)/hK^2;
6 I1(9,3*Nv+2) = 2*area(iel)/hK^2;
7 I1(10,3*Nv+3) = 6*area(iel)/hK^2;

```

对第二项, 记 $\mathbf{f} = (\nabla m \cdot \mathbf{n}_e)\phi^T$, 边界积分用 Gauss-Lobatto 公式 (5.20) 精确计算, 我们有

$$I_2 = \frac{1}{2} \sum_{i=1}^{N_v} h_{e_i} [w_1 \mathbf{f}(z_i) + w_2 \mathbf{f}(a_i) + w_3 \mathbf{f}(b_i) + w_4 \mathbf{f}(z_{i+1})],$$

其中,

$$\begin{aligned}\mathbf{f}(z_i) &= \partial_n m(z_i)[\mathbf{e}_i, \mathbf{0}, \mathbf{0}, 0, 0, 0], \\ \mathbf{f}(a_i) &= \partial_n m(a_i)[\mathbf{0}, \mathbf{e}_i, \mathbf{0}, 0, 0, 0], \\ \mathbf{f}(b_i) &= \partial_n m(b_i)[\mathbf{0}, \mathbf{0}, \mathbf{e}_i, 0, 0, 0], \\ \mathbf{f}(z_{i+1}) &= \partial_n m(z_{i+1})[\mathbf{e}_{i+1}, \mathbf{0}, \mathbf{0}, 0, 0, 0],\end{aligned}$$

这里, \mathbf{e}_i 是长度为 N_v 的第 i 个自然向量, $\mathbf{0}$ 是长度为 N_v 的全零向量.

```

1 % second term
2 I2 = 0;
3 for j = 1:Nv % loop of edges
4     % he*\partial_n(m) at zj, aj, bj, zj+1
5     Dn1 = Gradm(x(j),y(j))*Ne(j,:)';
6     Dn2 = Gradm(za(j,1),za(j,2))*Ne(j,:)';
7     Dn3 = Gradm(zb(j,1),zb(j,2))*Ne(j,:)';
8     Dn4 = Gradm(x(v2(j)),y(v2(j)))*Ne(j,:)';
9     % [ei,0,0]
10    e1 = zeros(1,Ndof); e1(j) = 1;
11    e2 = zeros(1,Ndof); e2(Nv+j) = 1;
12    e3 = zeros(1,Ndof); e3(2*Nv+j) = 1;
13    e4 = zeros(1,Ndof); e4(v2(j)) = 1;
14    % f
15    f1 = Dn1*e1; f2 = Dn2*e2; f3 = Dn3*e3; f4 = Dn4*e4;
16    I2 = I2 + (w(1)*f1+w(2)*f2+w(3)*f3+w(4)*f4);
17 end
18 B1 = -I1 + 1/2*I2;

```

投影的限制条件由 (5.11) 给出, $B1$ 的第一行将用 $P_0(\phi^T)$ 代替, 其中

$$P_0(v) = \int_K v dx.$$

```

1 % constraint
2 B1s = B1; B1s(1,3*Nv+1) = area(iel); Bs{iel} = B1s;
3 G{iel} = B1*D1; Gs{iel} = B1s*D1;

```

5.6.2 L^2 投影的计算

考虑提升空间

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

相比 $V_k(K)$, 它的自由度多出 K 上的 $k-1, k$ 阶矩量. 对 $v \in \tilde{V}_k(K)$, 因边界多项式次数未提高, 我们仍然只能如 $V_k(K)$ 中那样定义椭圆投影 $\Pi_k^\nabla v$, 即

$$\Pi_k^\nabla : \tilde{V}_k(K) \rightarrow \mathbb{P}_k(K), \quad v \mapsto \Pi_k^\nabla v$$

满足

$$(\nabla \Pi_k^\nabla v, \nabla p)_K = (\nabla v, \nabla p)_K, \quad p \in \mathbb{P}_k(K).$$

显然, $\Pi_k^\nabla v$ 的计算用不到 K 上 v 的 $k-1, k$ 阶矩量. 正因为如此, 我们可用 $\Pi_k^\nabla v$ 的 $k-1, k$ 阶矩量近似 v 的相应矩量, 从而降低自由度. 由此引入 enhancement 空间

$$W_k(K) = \left\{ w_h \in \tilde{V}_k(K) : (w_h - \Pi_k^\nabla w_h, q)_K = 0, \quad q \in \mathbb{M}_k \setminus \mathbb{M}_{k-2} \right\}.$$

在该空间中, 显然有

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

现在我们说明 $\tilde{V}_k(K)$ 中椭圆投影的计算, 这里 $k = 3$.

- $\tilde{V}_k(K)$ 中多出 2, 3 阶矩量自由度, 接着前面的自由度排列为 (多出的自由度个数 $n = 7$)

$$\chi_{3N_v+j}(v) = \frac{1}{|K|} \int_K m_j v dx, \quad j = 4, 5, \dots, 10,$$

此时自由度共 $3N_v + 10$ 个 (为了方便, 这些自由度重新编号为 $d_i(v), i = 1, \dots, n$). 过渡矩阵的前 $3N_v + 3$ 行与 D 相同, 后面的如下循环计算

```

1      DD = zeros(3*Nv+10, 10);
2      DD(1:Ndof,:) = D1;
3      for j = 4:10
4          mjv = @ (x,y) repmat(mmm{j}(x,y), 1, 10).*m(x,y); % moment j on K
5          DD(3*Nv+j,:) = 1/area(iel)*integralTri(mjv, 6, nodeT, elemT);
6      end

```

- 对矩阵

$$B = \int_K \nabla m \cdot \nabla \phi^T dx,$$

它比原来的多出 n 列, 对应 $k-1, k$ 阶矩量. 由于分部积分后, 右端的计算不涉及到这些矩量, 因此多出的 n 列全为零, 其他列不变. 限制条件也是如此. 这表明, 此时的 \tilde{B} 只比原来的多出 n 列零向量.

```

1      BBs = zeros(10,3*Nv+10);
2      BBs(:,1:Ndof) = B1s;

```

有了过渡矩阵和 \tilde{B} , 就可计算出 $\tilde{V}_k(K)$ 中的投影矩阵 Π_k^∇ .

```

1      GGs = BBs*DD;
2      PPis = GGs\BBs;    PPi = DD*PPis;

```

- 注意到 Π_k^∇ 的第 j 列就是第 j 个基函数的投影的自由度向量 $\chi(\Pi_k^\nabla \phi_j)$, 我们有 Π_k^∇ 的最后 n 行为

$$d_i(\Pi_k^\nabla \phi^T), \quad i = 1, \dots, n.$$

注意这里的 ϕ^T 是 $\tilde{V}_k(K)$ 的基函数, 去掉最后 n 个即为 $W_k(K)$ 中的 (虽然基函数可能不同, 但相同自由度值相同, 而椭圆投影由自由度唯一决定).

```
1 Dof = PPi(Ndof+1:3*Nv+10, 1:Ndof);
```

L^2 投影定义为

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

这等价于

$$(\Pi_k^0 \phi_i, m_\alpha)_K = (\phi_i, m_\alpha)_K, \quad i = 1, \dots, N_k, \quad \alpha = 1, \dots, N_p$$

或

$$\int_K m \Pi_k^0 \phi^T dx = \int_K m \phi^T dx.$$

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

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

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

$$\Pi_k^0 \phi^T = m^T \boldsymbol{\Pi}_{k*}^0. \quad (5.21)$$

由 (5.8) 知,

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

把这些关系代入 (5.16), 我们有 L^2 投影的矩阵表示

$$H \boldsymbol{\Pi}_{k*}^0 = C, \quad (5.22)$$

式中,

$$H = \int_K mm^T dx, \quad C = \int_K m \phi^T dx.$$

H 是可逆的 (Gram 矩阵), 但 C 不可计算, 因为它涉及到 K 上 $k-1, k$ 阶的矩量. 根据前面的说明, 我们将在 enhancement 空间 $W_k(K)$ 中考虑, 前面的 $V_k(K)$ 都换为 $W_k(K)$, 相应的基函数对应 $W_k(K)$. C 中多出的矩量 $d_i(\phi^T)$ 用 $d_i(\Pi_k^\nabla \phi^T)$ 代替. 显然 C 的结构如下 (基函数个数与 $V_k(K)$ 的相同, 共 $3N_v + 3$ 个):

- 前 3 行为

$$C = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & |K| & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & |K| & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & |K| \end{bmatrix}$$

- 后面若干行为 `area(iel)*Dof.`

```
1 % --- matrices of L2 projection ---
2 H = zeros(Nm,Nm);
3 for i = 1:Nm
4     for j = 1:Nm
5         fun = @(x,y) mm{i}(x,y).*mm{j}(x,y);
6         H(i,j) = integralTri(fun,4,nodeT,elemT);
7     end
8 end
9 C1 = zeros(Nm,Ndof);
```

```

10    C1(1,3*Nv+1) = area(iel);
11    C1(2,3*Nv+2) = area(iel);
12    C1(3,3*Nv+3) = area(iel);
13    C1(4:end,:) = area(iel)*Dof;
14    C{iel} = C1;

```

5.6.3 单元刚度矩阵和载荷向量的计算

根据 $k = 1$ 的说明, 单元刚度矩阵如下计算

```

1 % Projection
2 Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi)); % elliptic
3 Pi0s = H{iel}\C{iel}; Pi0 = D{iel}*Pi0s; % L2
4 % Stiffness matrix
5 AK = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi);
6 BK = pde.c*Pis'*H{iel}*Pis + pde.c*hK^2*(I-Pi0)'*(I-Pi0);
7 ABelem{iel} = reshape(AK+BK',1,[]); % straighten as row vector for easy assembly

```

载荷向量可按照式 (5.19) 计算, 不过, 对 $k \geq 3$, 也可如下计算获得最优误差阶

$$\ell_h^K(v) = \int_K f \Pi_{k-2}^0 v dx.$$

单元载荷向量为

$$F_K = \int_K f \Pi_{k-2}^0 \phi dx = (\Pi_{k-2,*}^0)^T \int_K f m^{k-2} dx,$$

式中, m^{k-2} 为直到 $k - 2$ 阶的尺度单项式向量. 投影矩阵 $\Pi_{k-2,*}^0$ 按照 (5.22) 类似计算. 易知,

```

1 % L2 projection for rhs
2 Hf{iel} = H1(1:3,1:3);
3 Cf{iel} = C1(1:3,:);

```

这样, 右端如下计算

```

1 % Load vector (L2 projection Pi_{k-2})
2 m = @x,y [m1(x,y), m2(x,y), m3(x,y)];
3 fm = @x,y repmat(pde.f([x,y]),1,3).*m(x,y); % f(p) = f([x,y])
4 rhs = integralTri(fm,4,nodeT,elemT); rhs = rhs';
5 Pifs = Hf{iel}\Cf{iel};
6 fK = Pifs'*rhs;
7 belem{iel} = fK'; % straighten as row vector for easy assembly

```

5.6.4 刚度矩阵和载荷向量的装配

装配过程类似前面处理, 只需要给定局部整体对应即可, 记为 elem2 . 对 N_v 个顶点的单元构型, 它有 $3 \times N_v + 3$ 列, 对应自由度的整体编号. 顶点和单元矩量的编号比较容易, 现在考虑边上的两个内部点的编号.

我们仍然要使用 elem2edge , 它按单元记录边的自然序号.

- 对内部边 e_i , 其中 i 是其自然序号, 我们要规定一个方向, 从而将两个内部点按该方向排列.
- 我们规定内部边的定向为: 从较小序号指向较大序号. 对第 iel 个单元, 执行下面语句即可获得所有边的定向.

```

1 index = elem{iel};
2 sgnL = sign(index(v2)-index(v1));

```

sgnL 的值显然为 $+1$ 或 -1 . 对 $+1$ 的边, 内部点要从较小顶点序号到较大顶点序号方向进行编号. 对 -1 的边, 编号方向相反.

- 假设 e_i 是正向边, 第一个点接着顶点编号为 $i+N$, 第二个点接着所有顶点和第一个点编号为 $i+N+NE$. 可如下实现

```

1     elem2edgeNv = elem2edge{iel};
2     elemNva = elem2edgeNv + N*sgnL + (N+NE)*(-sgnL);
3     elemNvb = elem2edgeNv + (N+NE)*sgnL + N*(-sgnL);

```

- 为了方便处理边界条件, 我们规定边界边的定向为逆时针方向, 如下修改 sgnL .

```

1     bdIndex = bdStruct.bdIndex; E = false(NE,1); E(bdIndex) = 1;
2     sgnL = sign(index(v2)-index(v1));
3     id = elem2edge{iel}; sgnbd = E(id); sgnL(sgnbd) = 1;
4     sgnL(sgnL== -1) = 0;
5     sgnelem{iel} = sgnL;

```

这里按单元存储所有边的符号, 记为 sgnelem .

根据上面的讨论, 最终的装配程序如下

```

1 %% Assemble stiffness matrix and load vector
2 elemLen = cellfun('length', elem); vertNum = unique(elemLen);
3 nnz = sum((3*elemLen+3).^2);
4 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
5 id = 0; ff = zeros(NNdof,1);
6 elem2dof = cell(NT,1);
7 for Nv = vertNum(:)' % only valid for row vector
8
9     idNv = find(elemLen == Nv); % find polygons with Nv vertices
10    NTv = length(idNv); % number of elements with Nv vertices
11
12    elemNv = cell2mat(elem(idNv)); % elem
13    elem2edgeNv = cell2mat(elem2edge(idNv));
14    sgnelemNv = cell2mat(sgnelem(idNv));
15    elemNva = elem2edgeNv+N*sgnelemNv+(N+NE)*(-sgnelemNv);
16    elemNvb = elem2edgeNv+(N+NE)*sgnelemNv +N*(-sgnelemNv);
17    elem2 = [elemNv, elemNva, elemNvb, ...
18              idNv+N+2*NE, idNv+N+2*NE+NT, idNv+N+2*NE+2*NT];
19    K = cell2mat(ABelem(idNv)); F = cell2mat(Belem(idNv));
20    Ndof = 3*Nv+3;
21    ii(id+1:id+NTv*Ndof^2) = reshape(repmat(elem2, Ndof,1), [], 1);
22    jj(id+1:id+NTv*Ndof^2) = repmat(elem2(:, Ndof, 1);
23    ss(id+1:id+NTv*Ndof^2) = K(:, );
24    id = id + NTv*Ndof^2;
25
26    % assemble the vector
27    ff = ff + accumarray(elem2(:, ), F(:, ), [NNdof 1]);
28
29    % elementwise global indices
30    elem2dof(idNv) = mat2cell(elem2, ones(NTv,1), Ndof);
31 end
32 kk = sparse(ii,jj,ss,NNdof,NNdof);

```

5.6.5 边界条件的处理

先考虑 Neumann 边界条件. 前面已经规定边界边的定向为逆时针, 则第一个内部点靠近起点. 对给定边 e , 设 4 个点分别为 z_1, \dots, z_4 , 相应的基函数为 ϕ_1, \dots, ϕ_4 , 则载荷向量为

$$F_e = [F_1, \dots, F_4]^T, \quad F_i = \int_e \partial_n u \phi_i ds.$$

用 Gauss-Lobatto 积分近似, 有

$$F_j = \frac{h_e}{2} \sum_{i=1}^4 w_i \partial_n u(z_i) \phi_j(z_i) = \frac{h_e}{2} w_j \partial_n u(z_j), \quad j = 1, \dots, 4.$$

Neumann 条件如下装配.

```

1 %% Assemble Neumann boundary conditions
2 bdEdgeN = bdStruct.bdEdgeN; bdEdgeIdxN = bdStruct.bdEdgeIdxN;
3 if ~isempty(bdEdgeN)
4     g_N = pde.Du;
5     z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
6     za = z1+r(1)*(z2-z1); zb = z1+r(2)*(z2-z1); % Gauss-Lobatto
7     e = z1-z2; % e = z2-z1
8     Ne = [-e(:,2),e(:,1)]; % scaled ne
9     F1 = w(1)*sum(Ne.*g_N(z1),2)/2;
10    F2 = w(4)*sum(Ne.*g_N(z2),2)/2;
11    Fa = w(2)*sum(Ne.*g_N(za),2)/2;
12    Fb = w(3)*sum(Ne.*g_N(zb),2)/2;
13    FN = [F1,F2,Fa,Fb];
14    bdEdgeN2 = [bdEdgeN, bdEdgeIdxN+N, bdEdgeIdxN+N+NE];
15    ff = ff + accumarray(bdEdgeN2(:,1), FN(:,1), [NNdof 1]);
16 end

```

Dirichlet 边界条件如下处理.

```

1 %% Apply Dirichlet boundary conditions
2 % boundary information
3 g_D = pde.g_D;
4 bdNodeIdx = bdStruct.bdNodeIdx; bdEdgeD = bdStruct.bdEdgeD;
5 bdEdgeIdxD = bdStruct.bdEdgeIdxD;
6 bdEdgeIdxa = bdEdgeIdxD + N;
7 bdEdgeIdxb = bdEdgeIdxD + N+NE;
8 id = [bdNodeIdx; bdEdgeIdxa; bdEdgeIdxb];
9 isBdNode = false(NNdof,1); isBdNode(id) = true;
10 bdDof = (isBdNode); freeDof = (~isBdNode);
11 % vertices on the boundary
12 pD = node(bdNodeIdx,:); uD = g_D(pD);
13 % ai, bi on the boundary
14 z1 = node(bdEdgeD(:,1),:); z2 = node(bdEdgeD(:,2),:);
15 za = z1+r(1)*(z2-z1); zb = z1+r(2)*(z2-z1); % Gauss-Lobatto
16 uDa = g_D(za); uDb = g_D(zb);
17 % rhs
18 u = zeros(NNdof,1); u(bdDof) = [uD; uDa; uDb];
19 ff = ff - kk*u;

```

5.6.6 误差分析

为了方便, 我们把函数文件中的 elem2 逐个单元存起来, 并输出.

```

1 elem2dof = cell(NT,1);
2 elem2dof(idNv) = mat2cell(elem2, ones(NTv,1), Ndof);
3 info.Ph = Ph; info.elem2dof = elem2dof;

```

H^1 和 L^2 误差如下计算.

```

1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [100, 200, 300, 400, 500];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10
11 %% PDE data
12 c = 1;
13 pde = Poissondata1(c);
14 bdNeumann = 'abs(x-0)<1e-4 | abs(x-1)<1e-4'; % string for Neumann
15
16 %% Virtual element method
17 for k = 1:maxIt
18     % load mesh
19     load( ['meshdata', num2str(nameV(k)), '.mat'] );
20     % get boundary information
21     bdStruct = setboundary(node,elem,bdNeumann);
22     % solve
23     [u,info] = PoissonVEMk3(node,elem,pde,bdStruct);
24     % record and plot
25     N(k) = length(u); h(k) = 1/sqrt(size(elem,1));
26     if size(elem,1)<2e3
27         figure(1); clf;
28         subplot(1,2,1), showmesh(node,elem);
29         subplot(1,2,2), showsolution(node,elem,u(1:size(node,1)));
30     end
31     % compute errors in discrete L2 and H1 norms
32     index = info.elem2dof; % elemenwise global index
33     chi = cellfun(@(id) u(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
34     kOrder = 3;
35     Ph = info.Ph;
36     ErrL2(k) = getL2error(node,elem,Ph,chi,pde,kOrder);
37     ErrH1(k) = getH1error(node,elem,Ph,chi,pde,kOrder);
38 end
39
40 %% Plot convergence rates and display error table
41 figure(2);
42 showrateh(h,ErrL2,ErrH1);
43
44 fprintf('\n');
45 disp('Table: Error')
46 colname = {'#Dof','h','||u-u_h||','|u-u_h|_1'};
47 disptable(colname,N,[],h,'%0.3e',ErrL2,'%0.5e',ErrH1,'%0.5e');

```

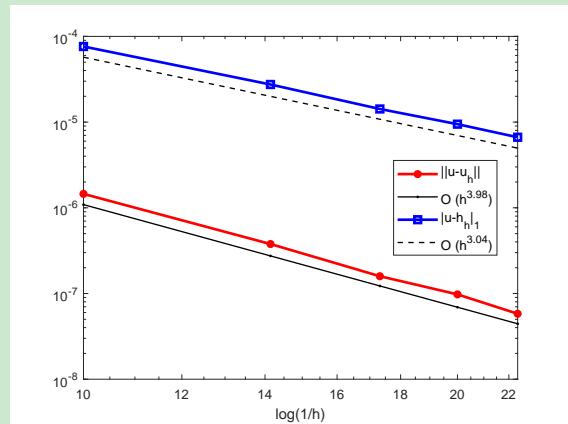


图 13. Poisson 方程三阶虚拟元方法的误差阶

可以看到, L^2 的误差阶为 4, H^1 的误差阶为 3.

6 三维 Poisson 方程的虚拟元方法

待添加

6.1 变分问题

6.1.1 虚拟元空间

设 K 是三维多面体单元, 定义边界元空间

$$\mathbb{B}_k(\partial K) = \{v \in C^0(\partial K) : v|_F \in V_k(F), \quad F \subset \partial K\},$$

则三维的虚拟元空间为

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

自由度由两部分组成, 一是边界元空间产生的, 一是 $\Delta v \in \mathbb{P}_{k-2}(K)$ 产生的. 对前者, 自由度包括

- 顶点值泛函: $\chi_a(v) = v(z)$, 其中 z 是 K 的顶点;
- 边 e 上 $k-1$ 个点处的值或直到 $k-1$ 阶的矩量;
- 面 F 上直到 $k-2$ 阶的矩量

$$\chi_F(v) = |F|^{-1}(m, v)_F, \quad m \in \mathbb{M}_{k-2}(F).$$

对后者, 我们还要加上 K 上直到 $k-2$ 阶的矩量.

$$\chi_K(v) = |K|^{-1}(m, v)_K, \quad m \in \mathbb{M}_{k-2}(K).$$

特别地, 当 $k=1$ 时, 只有顶点值处的自由度.

L^2 投影可类似二维问题处理.

$$B = \int_K \nabla m \cdot \nabla \phi^T dx = - \int_K \Delta m \cdot \phi^T dx + \sum_{f \subset \partial K} \int_f (\nabla m \cdot \mathbf{n}_f) \phi^T d\sigma,$$

面可如下计算

$$\int_f (\nabla m \cdot \mathbf{n}_f) \phi^T d\sigma = \int_f (\nabla m \cdot \mathbf{n}_f) \Pi_{k,f}^0 \phi^T d\sigma$$

6.1.2 虚拟元方法的变分问题

椭圆投影类似二维定义, 稳定化泛函为

$$S^K(u, v) = h_K \boldsymbol{\chi}(u) \cdot \boldsymbol{\chi}(v),$$

双线性形式为

$$a_h^K(u, v) = a^K(\Pi_k^\nabla u, \Pi_k^\nabla v) + S^K(u - \Pi_k^\nabla u, v - \Pi_k^\nabla v).$$

7 线弹性边值问题的虚拟元方法

7.1 线弹性边值问题简介

7.1.1 问题说明

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

$$\begin{cases} -\partial_j \sigma_{ij}(\mathbf{u}) = f_i, & i = 1, 2, 3 \quad \text{in } \Omega, \\ \mathbf{u} = \mathbf{0} & \text{on } \Gamma_0, \\ \sigma_{ij}(\mathbf{u}) n_j = g_i, & i = 1, 2, 3 \quad \text{on } \Gamma_1, \end{cases} \quad (7.23)$$

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

$$\begin{cases} -\operatorname{div} \boldsymbol{\sigma} = \mathbf{f} & \text{in } \Omega, \\ \mathbf{u} = \mathbf{0} & \text{on } \Gamma_0, \\ \boldsymbol{\sigma} \mathbf{n} = \mathbf{g} & \text{on } \Gamma_1, \end{cases} \quad (7.24)$$

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

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

$$\sigma_{ij}(\mathbf{u}) = \sigma_{ji}(\mathbf{u}) = \lambda \varepsilon_{kk}(\mathbf{u}) \delta_{ij} + 2\mu \varepsilon_{ij}(\mathbf{u}), \quad (7.25)$$

$$\varepsilon_{ij}(\mathbf{u}) = \varepsilon_{ji}(\mathbf{u}) = \frac{1}{2}(\partial_j u_i + \partial_i u_j), \quad (7.26)$$

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

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

这样, 平衡方程也可写为如下的紧凑形式

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

把 (7.25)-(7.26) 代入 (7.23), 可获得平衡方程的另一形式

$$-\mu \Delta \mathbf{u} - (\lambda + \mu) \operatorname{grad}(\operatorname{div} \mathbf{u}) = \mathbf{f} \quad \text{in } \Omega, \quad (7.27)$$

有时候会直接考虑该方程, 因为其中每个算子都是熟悉的. 该形式常称为 Navier 形式, 因为它与 Navier-Stokes 问题有点像.

7.1.2 连续变分问题

设

$$V = \left\{ \mathbf{v} \in H^1(\Omega)^3 : \mathbf{v} = \mathbf{0} \text{ on } \Gamma_0 \right\}, \quad (7.28)$$

令 $\mathbf{v} = (v_1, v_2, v_3)^T \in V$, 在 (7.23) 的平衡方程的两边乘以 v_i , 有

$$\int_{\Omega} -\partial_j \sigma_{ij}(\mathbf{u}) v_i dx = \int_{\Omega} f_i v_i dx,$$

这里遵循求和约定, 即上式实际上求和. 分部积分有

$$-\int_{\partial\Omega} \sigma_{ij}(\mathbf{u}) v_i n_j ds + \int_{\Omega} \sigma_{ij}(\mathbf{u}) \partial_j v_i dx = \int_{\Omega} f_i v_i dx$$

或

$$-\int_{\partial\Omega} g_i v_i ds + \int_{\Omega} \sigma_{ij}(\mathbf{u}) \partial_j v_i dx = \int_{\Omega} f_i v_i dx.$$

由对称性,

$$\sigma_{ij}(\mathbf{u}) \partial_j v_i = \sigma_{ij}(\mathbf{u}) \frac{1}{2} (\partial_j v_i + \partial_i v_j) = \sigma_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}),$$

故

$$\int_{\Omega} \sigma_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) dx = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx + \int_{\Gamma_1} \mathbf{g} \cdot \mathbf{v} ds. \quad (7.29)$$

于是, 变分问题为: 求 $\mathbf{u} \in V$ 使得,

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

式中,

$$a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \sigma_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) dx, \quad \ell(\mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx + \int_{\Gamma_1} \mathbf{g} \cdot \mathbf{v} ds.$$

文献中一般习惯引入如下记号

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

从而双线性形式可写为

$$a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) dx.$$

注意到

$$\begin{aligned} \sigma_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) &= (\lambda \varepsilon_{kk}(\mathbf{u}) \delta_{ij} + 2\mu \varepsilon_{ij}(\mathbf{u})) \varepsilon_{ij}(\mathbf{v}) \\ &= (\lambda \partial_k u_k \delta_{ij} + 2\mu \varepsilon_{ij}(\mathbf{u})) \varepsilon_{ij}(\mathbf{v}) \\ &= 2\mu \varepsilon_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) + \lambda \partial_k u_k \varepsilon_{ii}(\mathbf{v}) \\ &= 2\mu \varepsilon_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) + \lambda \partial_k u_k \partial_i u_i, \end{aligned}$$

双线性形式还可写为

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &= 2\mu \int_{\Omega} \varepsilon_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) dx + \lambda \int_{\Omega} \partial_i u_i \partial_j u_j dx, \\ &= 2\mu \int_{\Omega} \boldsymbol{\varepsilon}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) dx + \lambda \int_{\Omega} (\operatorname{div} \mathbf{u})(\operatorname{div} \mathbf{v}) dx. \end{aligned} \quad (7.30)$$

注 7.1 也可采用 (7.27) 的形式, 具体写出来即

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

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

$$\begin{aligned} & \mu \left(\int_{\Omega} \nabla u_1 \cdot \nabla v_1 dx - \int_{\partial\Omega} \partial_n u_1 v_1 ds \right) \\ & - (\lambda + \mu) \left(\int_{\partial\Omega} (\operatorname{div} \mathbf{u}) v_1 n_x ds - \int_{\Omega} (\operatorname{div} \mathbf{u}) \partial_x v_1 dx \right) = \int_{\Omega} f_1 v_1 dx. \end{aligned}$$

类似可获得第二个式子对应的结果, 将它们相加有

$$\begin{aligned} & \mu \int_{\Omega} \nabla \mathbf{u} \cdot \nabla \mathbf{v} dx + (\lambda + \mu) \int_{\Omega} (\operatorname{div} \mathbf{u})(\operatorname{div} \mathbf{v}) dx \\ & - \mu \int_{\partial\Omega} \partial_n \mathbf{u} \cdot \mathbf{v} ds - (\lambda + \mu) \int_{\partial\Omega} (\operatorname{div} \mathbf{u})(\mathbf{v} \cdot \mathbf{n}) ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx. \end{aligned} \quad (7.31)$$

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

7.1.3 近似变分形式 I — Navier 形式

先考虑 (7.31) 的近似变分形式, 令

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

式中,

$$a_{\nabla}^K(\mathbf{u}, \mathbf{v}) = \int_K \nabla \mathbf{u} \cdot \nabla \mathbf{v} dx, \quad a_{\operatorname{div}}^K(\mathbf{u}, \mathbf{v}) = \int_K (\operatorname{div} \mathbf{u})(\operatorname{div} \mathbf{v}) dx.$$

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

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

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

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

根据文献,

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

而稳定双线性形式为 ($k = 1$)

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

稍后解释这里的投影算子.

7.1.4 近似变分形式 II — tensor 形式

考虑 (7.30) 的变分问题, 令

$$\begin{aligned} a^K(\mathbf{u}, \mathbf{v}) &= 2\mu \int_K \varepsilon_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) dx + \lambda \int_K \partial_i u_i \partial_j u_j dx \\ &= 2\mu \int_K \boldsymbol{\varepsilon}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) dx + \lambda \int_K (\operatorname{div} \mathbf{u})(\operatorname{div} \mathbf{v}) dx \\ &=: 2\mu a_\mu^K(\mathbf{u}, \mathbf{v}) + \lambda a_\lambda^K(\mathbf{u}, \mathbf{v}), \end{aligned}$$

式中,

$$\begin{aligned} a_\mu^K(\mathbf{u}, \mathbf{v}) &= \int_K \varepsilon_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) dx = \int_K \boldsymbol{\varepsilon}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) dx, \\ a_\lambda^K(\mathbf{u}, \mathbf{v}) &= \int_K \partial_i u_i \partial_j u_j dx = \int_K (\operatorname{div} \mathbf{u})(\operatorname{div} \mathbf{v}) dx. \end{aligned}$$

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

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

根据文献,

$$\begin{aligned} a_{h,\mu}^K(\mathbf{u}, \mathbf{v}) &= a_\mu^K(\Pi^a \mathbf{u}, \Pi^a \mathbf{v}) + S^K(\mathbf{u} - \Pi^a \mathbf{u}, \mathbf{v} - \Pi^a \mathbf{v}), \\ a_{h,\lambda}^K(\mathbf{u}, \mathbf{v}) &= \int_K \Pi_{k-1}^0(\operatorname{div} \mathbf{u}) \Pi_{k-1}^0(\operatorname{div} \mathbf{v}) dx, \end{aligned}$$

稍后解释这里的投影算子.

7.2 刚度矩阵和载荷向量的装配

7.2.1 矩阵分析法

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

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

其中,

$$\Psi = [\varphi_1, \dots, \varphi_N]^T, \quad \bar{U} = [\bar{u}_1, \dots, \bar{u}_N]^T, \quad \underline{U} = [\underline{u}_1, \dots, \underline{u}_N]^T.$$

于是,

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

式中,

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

为了方便, 用 a 代替 a_h , 有

$$a(\mathbf{u}, \mathbf{v}) = a(\bar{u}, \bar{v}) + a(\bar{u}, \underline{v}) + a(\underline{u}, \bar{v}) + a(\underline{u}, \underline{v}).$$

而

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

$$\begin{aligned}
a(\bar{\mathbf{u}}, \bar{\mathbf{v}}) &= \sum_{i,j=1}^N a(\bar{\varphi}_i, \underline{\varphi}_j) \bar{u}_i \bar{v}_j = \underline{V}^T A_{21} \bar{U}, \quad A_{21}(j,i) = a(\bar{\varphi}_i, \underline{\varphi}_j), \\
a(\underline{\mathbf{u}}, \bar{\mathbf{v}}) &= \bar{V}^T A_{12} \underline{U}, \quad A_{12}(j,i) = a(\underline{\varphi}_i, \bar{\varphi}_j), \\
a(\underline{\mathbf{u}}, \underline{\mathbf{v}}) &= \underline{V}^T A_{22} \underline{U}, \quad A_{22}(j,i) = a(\underline{\varphi}_i, \underline{\varphi}_j),
\end{aligned}$$

于是

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

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

对右端的线性形式, 有

$$\ell(\mathbf{v}) = \ell(\bar{\mathbf{v}}) + \ell(\underline{\mathbf{v}}),$$

而

$$\begin{aligned}
\ell(\bar{\mathbf{v}}) &= \sum_{i=1}^N \bar{v}_i \ell(\bar{\varphi}_i) = \bar{V}^T F_1, \quad F_1 = [\ell(\bar{\varphi}_1), \dots, \ell(\bar{\varphi}_N)]^T, \\
\ell(\underline{\mathbf{v}}) &= \sum_{i=1}^N \underline{v}_i \ell(\underline{\varphi}_i) = \underline{V}^T F_2, \quad F_2 = [\ell(\underline{\varphi}_1), \dots, \ell(\underline{\varphi}_N)]^T,
\end{aligned}$$

于是

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

注 7.2 对 A_{ij} , 例如, $A_{21} = a(\bar{\varphi}_i, \underline{\varphi}_j)$, 双线性形式中只会留下 (\underline{v}, \bar{u}) 这样的配对项. 这对应标量情形的计算. 其他项以及右端类似.

7.2.2 sparse 装配指标

先考虑三角形单元. 刚度矩阵

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

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

```

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

```

为了避免对稀疏矩阵进行运算, 上面分块装配可如下改进

CODE 15. 向量方程的分块 sparse 装配指标 (三角剖分)

```

1 ii11 = ii; jj11 = jj; ii12 = ii; jj12 = jj+N;
2 ii21 = ii+N; jj21 = jj; ii22 = ii+N; jj22 = jj+N;

```

```

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

```

对有些问题, 单元刚度矩阵

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

并不是分块各自生成, 而是直接生成 (例如 VEM), 此时可找到每个块执行上面的装配, 即将 A_{ij}^K 行拉直拼接. 为了方便, 我们将 A_K 看成整体进行装配. 回忆一下装配指标, 对单元刚度矩阵 $[K^e] = [k_{ij}]_{3 \times 3}$, 装配指标首先给出 k_{11} 所有单元的 (i, j, s) , 记为 i_{11}, j_{11}, s_{11} , 接着按行给出其他位置的稀疏指标, 它们拼接如下

$$\begin{bmatrix} i_{11} & j_{11} & s_{11} \\ i_{12} & j_{12} & s_{12} \\ \vdots & \vdots & \vdots \\ i_{33} & j_{33} & s_{33} \end{bmatrix}.$$

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

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

可如下给出装配指标

CODE 16. 向量方程的 sparse 装配指标 (三角剖分)

```

1 NT = size(elem,1); N = size(node,1);
2 Ndof = 3; Ndof2 = 2*Ndof; nnz = NT*Ndof2^2;
3 ii = zeros(nnz,1); jj = zeros(nnz,1);
4 elem2 = [elem, elem+N];
5 ii(id+1:id+NTv*Ndof2^2) = reshape(repmat(elem2, Ndof2, 1), [], 1);
6 jj(id+1:id+NTv*Ndof2^2) = repmat(elem2(:, ), Ndof2, 1);
7 ss(id+1:id+NTv*Ndof2^2) = K(:, );
8 id = id + NTv*Ndof2^2;

```

类似 CODE. 17, 对多角形剖分, 我们可如下给出装配指标

CODE 17. 弹性方程 VEM 的 sparse 装配

```

1 % ----- Assemble the stiff matrix and load vector -----
2 elemLen = cellfun('length', elem); vertNum = unique(elemLen);
3 nnz = sum(4*elemLen.^2);
4 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
5
6 id = 0; ff = zeros(2*N,1);
7 for Nv = vertNum(:)' % only valid for row vector
8
9     % assemble the matrix
10    idNv = find(elemLen == Nv); % find polygons with Nv vertices
11    NTv = length(idNv); % number of elements with Nv vertices
12
13    elemNv = cell2mat(elem(idNv)); % elem
14    K = cell2mat(ABelem(idNv)); F = cell2mat(Belem(idNv));
15    elem2 = [elemNv, elemNv+N];

```

```

16 Ndof = Nv; Ndof2 = 2*Ndof;
17 ii(id+1:id+NTv*Ndof2^2) = reshape(repmat(elem2, Ndof2, 1), [], 1);
18 jj(id+1:id+NTv*Ndof2^2) = repmat(elem2(:, ), Ndof2, 1);
19 ss(id+1:id+NTv*Ndof2^2) = K(:, );
20 id = id + NTv*Ndof2^2;
21
22 % assemble the vector
23 ff = ff + accumarray([elem2(:, ), F(:, ), [2*N 1]]);
24 end
25 kk = sparse(ii,jj,ss,2*N,2*N);

```

7.3 变分形式 I: Navier 型

7.3.1 过渡矩阵

设分量空间 $V_k(K)$ 的基函数为 ϕ_1, \dots, ϕ_N , 则向量空间 $\mathbf{V}_k(K)$ 的基函数为

$$\bar{\phi}_1, \dots, \bar{\phi}_N, \underline{\phi}_1, \dots, \underline{\phi}_N,$$

式中,

$$\bar{\phi}_i = \begin{bmatrix} \phi_i \\ 0 \end{bmatrix}, \quad \underline{\phi}_i = \begin{bmatrix} 0 \\ \phi_i \end{bmatrix}, \quad i = 1, \dots, N.$$

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

$$\mathbf{u} = \sum_{i=1}^{N_k} \bar{u}_i \bar{\phi}_i + \sum_{i=1}^{N_k} u_i \underline{\phi}_i, \quad \bar{u}_i = \chi_i(\bar{u}), \quad u_i = \chi_i(\underline{u}),$$

于是

$$\text{dof}_i(\mathbf{u}) = \chi_i(u_1), \quad \text{dof}_{i+N}(\mathbf{u}) = \chi_i(u_2), \quad 1 \leq i \leq N.$$

类似地, 设

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

并把它们排列为

$$\begin{aligned} \mathbf{m}^T &= [\bar{m}_1, \bar{m}_2, \bar{m}_3, m_1, \underline{m}_2, \underline{m}_3] =: [\bar{m}^T, \underline{m}^T], \\ \boldsymbol{\phi}^T &= [\bar{\phi}_1, \dots, \bar{\phi}_{N_k}, \underline{\phi}_1, \dots, \underline{\phi}_{N_k}] =: [\bar{\phi}^T, \underline{\phi}^T]. \end{aligned}$$

设过渡矩阵为 \mathbf{D} , 即

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

由

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

知

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

7.3.2 椭圆投影

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

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

满足

$$\begin{cases} a_\nabla^K(\Pi^\nabla \mathbf{v}, \mathbf{p}) = a_\nabla^K(\mathbf{v}, \mathbf{p}), & \mathbf{p} \in (\mathbb{P}_k(K))^2, \\ \int_{\partial K} \Pi^\nabla \mathbf{v} \, ds = \int_{\partial K} \mathbf{v} \, ds \end{cases}$$

或

$$\begin{cases} \int_K \nabla \Pi^\nabla \mathbf{v} \cdot \nabla \mathbf{p} \, dx = \int_K \nabla \mathbf{v} \cdot \nabla \mathbf{p} \, dx, & \mathbf{p} \in (\mathbb{P}_k(K))^2, \\ \int_{\partial K} \Pi^\nabla \mathbf{v} \, ds = \int_{\partial K} \mathbf{v} \, ds. \end{cases} \quad (7.32)$$

定义的向量形式为

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

或

$$\begin{cases} \int_K \nabla \mathbf{m} \cdot \nabla \Pi^\nabla \boldsymbol{\phi}^T \, dx = \int_K \nabla \mathbf{m} \cdot \nabla \boldsymbol{\phi}^T \, dx, \\ P_0(\Pi^\nabla \boldsymbol{\phi}^T) = P_0(\boldsymbol{\phi}^T), \end{cases}$$

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

$$\Pi^\nabla \boldsymbol{\phi}^T = \boldsymbol{\phi}^T \mathbf{\Pi}^\nabla, \quad \mathbf{\Pi}^\nabla = (\text{dof}_i(\Pi^\nabla \boldsymbol{\phi}_j)).$$

根据前面自由度的说明, 有

$$\mathbf{\Pi}^\nabla = \begin{bmatrix} (\chi_i(\Pi^\nabla \boldsymbol{\phi}_j)) & \\ & (\chi_i(\Pi^\nabla \boldsymbol{\phi}_j)) \end{bmatrix} = \begin{bmatrix} \mathbf{\Pi}_\nabla & \\ & \mathbf{\Pi}_\nabla \end{bmatrix}.$$

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

$$\Pi_\nabla \boldsymbol{\phi}^T = \boldsymbol{\phi}^T \mathbf{\Pi}_\nabla, \quad \mathbf{\Pi}_\nabla = (\chi_i(\Pi^\nabla \boldsymbol{\phi}_j)). \quad (7.33)$$

也可如下获得. 由 (7.33),

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

此即 Π^∇ 在基 $\boldsymbol{\phi}^T$ 下的矩阵为

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

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

$$\Pi_{\nabla}\phi^T = m^T \mathbf{\Pi}_{a*}, \quad (7.34)$$

则有

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

其中, D 为过渡矩阵. 类似由 (7.33) 得

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

显然,

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

代入前面的矩阵表示, 有

$$\begin{cases} \mathbf{G} \mathbf{\Pi}^{\nabla*} = \mathbf{B}, \\ P_0(\mathbf{m}^T) \mathbf{\Pi}^{\nabla*} = P_0(\phi^T), \end{cases} \quad (7.35)$$

其中,

$$\begin{aligned} \mathbf{G} &= a_{\nabla}^K(\mathbf{m}, \mathbf{m}^T) = \int_K \nabla \mathbf{m} \cdot \nabla \mathbf{m}^T dx, \\ \mathbf{B} &= a_{\nabla}^K(\mathbf{m}, \phi^T) = \int_K \nabla \mathbf{m} \cdot \nabla \phi^T dx. \end{aligned}$$

直接计算有

$$\begin{aligned} \mathbf{G} &= \int_K \nabla \mathbf{m} \cdot \nabla \mathbf{m}^T dx = \begin{bmatrix} (\nabla \bar{m}, \nabla \bar{m}^T)_K & (\nabla \bar{m}, \nabla \underline{m}^T)_K \\ (\nabla \underline{m}, \nabla \bar{m}^T)_K & (\nabla \underline{m}, \nabla \underline{m}^T)_K \end{bmatrix} \\ &= \begin{bmatrix} (\nabla m, \nabla m^T)_K & \\ & (\nabla m, \nabla m^T)_K \end{bmatrix} = \begin{bmatrix} G & \\ & G \end{bmatrix}, \end{aligned}$$

类似地,

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

根据 Poisson 方程, G 不可逆, 它的第一行全为零, 为此 \mathbf{G} 的分块的第一行都为零. 注意到 (7.35) 中的

$$\begin{aligned} P_0(\mathbf{m}^T) &= [P_0(\bar{m}^T), P_0(\underline{m}^T)] = \begin{bmatrix} P_0(m^T) & \mathbf{0}^T \\ \mathbf{0}^T & P_0(m^T) \end{bmatrix}, \\ P_0(\phi^T) &= [P_0(\bar{\phi}^T), P_0(\underline{\phi}^T)] = \begin{bmatrix} P_0(\phi^T) & \mathbf{0}^T \\ \mathbf{0}^T & P_0(\phi^T) \end{bmatrix}, \end{aligned}$$

它们是两行的方程, 用它们分别替换分块的首行, 则有

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

其中,

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

由分量矩阵的关系, 我们有

引理 7.1

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

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

注 7.3 正因为向量情形的矩阵只是分量矩阵的对角分块, 投影矩阵自然也是, 从而不必再计算向量情形的各种矩阵.

7.3.3 L^2 投影

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

$$a_{\text{div}}^K(\boldsymbol{u}, \boldsymbol{v}) = \int_K (\text{div } \boldsymbol{u})(\text{div } \boldsymbol{v}) dx,$$

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

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

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

这是因为此时的右端可计算. 事实上,

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

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

定义的向量形式为

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

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

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

则投影的矩阵形式为

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

式中,

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

显然 H_0 是 H 的前若干行若干列组成.

对 $k=1$, 有 $m^0 = m_1 = 1$, 且

$$C_0 = \int_K m^0(\text{div } \boldsymbol{\phi}^T) dx = \int_K \text{div } \boldsymbol{\phi}^T dx = \int_{\partial K} \boldsymbol{\phi}^T \cdot \boldsymbol{n} ds, \tag{7.36}$$

这里

$$\int_{\partial K} \boldsymbol{\phi}^T \cdot \boldsymbol{n} ds = \int_{\partial K} [\bar{\boldsymbol{\phi}}^T \cdot \boldsymbol{n}, \underline{\boldsymbol{\phi}}^T \cdot \boldsymbol{n}] ds = \int_{\partial K} [\boldsymbol{\phi}^T \cdot \boldsymbol{n}_x, \boldsymbol{\phi}^T \cdot \boldsymbol{n}_y] ds.$$

由 Poisson 方程那里的 (5.13),

$$\int_{\partial K} \boldsymbol{\phi}_i \cdot \boldsymbol{n} ds = \frac{1}{2} (|e_{i-1}| \boldsymbol{n}_{e_{i-1}} + |e_i| \boldsymbol{n}_{e_i}) = \frac{1}{2} |\hat{e}_i| \boldsymbol{n}_{\hat{e}_i},$$

此即

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

由此可获得 C_0 . 显然此时有 $H_0 = |K|$, 于是

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

前面需要的矩阵可如下生成

```

1 %% Compute projection matrices
2 D = cell(NT,1);
3 % B = cell(NT,1); % it is not used in the computation
4 Bs = cell(NT,1);
5 G = cell(NT,1); Gs = cell(NT,1);
6 H0 = cell(NT,1); C0 = cell(NT,1);
7 for iel = 1:NT
8     index = elem{iel}; Nv = length(index);
9     xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
10
11    m1 = @x,y) 1 + 0*x; m2 = @x,y) (x-xK)./hK; m3 = @x,y) (y-yK)./hK;
12    m = @x,y) [m1(x,y), m2(x,y), m3(x,y)];
13    x = node(index,1); y = node(index,2);
14
15    % D
16    D1 = m(x,y); D{iel} = D1;
17
18    % B, Bs, G, Gs
19    rotid1 = [Nv, 1:Nv-1]; rotid2 = [2:Nv, 1]; % ending and starting indices
20    Gradm = [0 0; 1./hK*[1, 0]; 1./hK*[0, 1]];
21    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of ...
        edge vector
22    B1 = Gradm*normVec; B1s = B1; B1s(1,:) = 1/Nv;
23    % B{iel} = B1;
24    Bs{iel} = B1s;
25    G{iel} = B1*D1; Gs{iel} = B1s*D1;
26
27    % H0, C0
28    H0{iel} = area(iel);
29    C0{iel} = reshape(normVec', 1, []);
30 end

```

7.3.4 单元刚度矩阵和载荷向量的计算

类似 Poisson 方程, 有

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

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

$$\mathbf{B}_K = \int_K \Pi_{k-1}^0(\operatorname{div} \boldsymbol{\phi}) \Pi_{k-1}^0(\operatorname{div} \boldsymbol{\phi}^T) dx = (\Pi_{0*})^T H_0 \Pi_{0*}.$$

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

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

这里,

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

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

再考虑右端的计算. 右端的向量形式为

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

为此只要考虑

$$F_K(\boldsymbol{\phi}) = \int_K \Pi_{k-2}^0 f \boldsymbol{\phi} dx, \quad f = f_1, f_2.$$

当 $k = 1$ 时, 规定 $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (5.14) 定义, 即单元顶点值的平均. 注意到

$$F_K(\boldsymbol{\phi}) = \int_K \Pi_0^0 f \boldsymbol{\phi} dx = \int_K \Pi_0^0 f \Pi_k^0 \boldsymbol{\phi} dx,$$

我们有

$$F_K = \int_K \Pi_0^0 f \Pi_0^0 \boldsymbol{\phi} dx = f(x_K, y_K) |K| \frac{1}{N_v} [1, \dots, 1]_{N_v}^T,$$

这里为了方便, $\Pi_0^0 f$ 用中点值近似.

如下获得刚度矩阵和载荷向量

```

1 %% Get elementwise stiffness matrix and load vector
2 ABelem = cell(NT,1); belem = cell(NT,1);
3 Ph = cell(NT,1); % matrix for error evaluation
4 for iel = 1:NT
5     % Projection
6     Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi));
7     PiOs = H0{iel}\C0{iel};
8     % Stiffness matrix
9     AK = Pis'*G{iel}*Pis + (I-Pi)'*(I-Pi); AK = para.mu*blkdiag(AK,AK);
10    BK = (para.mu+para.lambda)*PiOs'*H0{iel}*PiOs;
11    ABelem{iel} = reshape(AK'+BK',1,[]); % straighten
12    % Load vector
13    Nv = length(elem{iel});
14    fK = pde.f(centroid(iel,:))*area(iel)/Nv; fK = repmat(fK,Nv,1);
15    belem{iel} = fK(:)'; % straighten
16    % matrix for L2 and H1 error evaluation
17    Ph{iel} = blkdiag(Pis,Pis);
18 end

```

7.3.5 边界条件的处理

该问题只有 Dirichlet 边界条件, 如下处理.

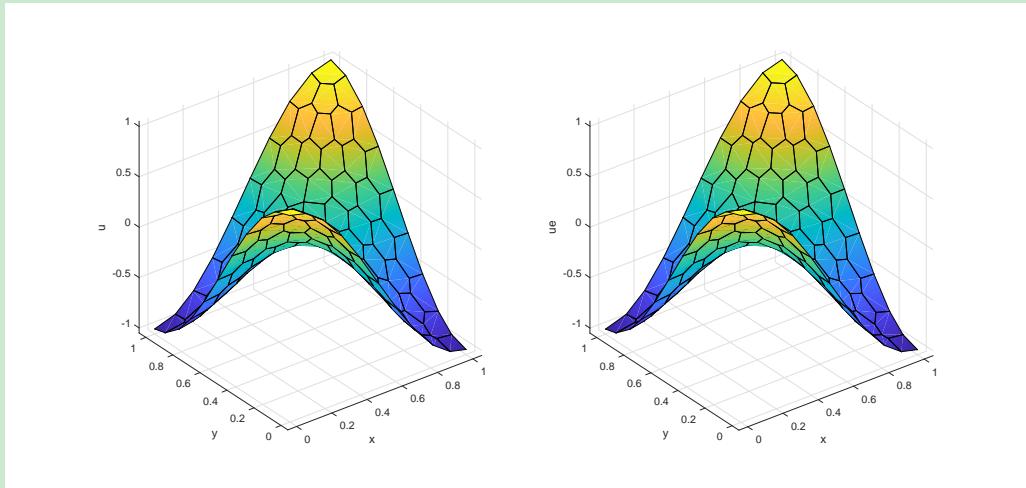
```

1 %% Apply Dirichlet boundary conditions
2 g_D = pde.g_D;
3 bdNodeIdx = bdStruct.bdNodeIdx;
4 isBdNode = false(N,1); isBdNode(bdNodeIdx) = true;
5 bdNode = find(isBdNode); freeNode = find(~isBdNode);
6 pD = node(bdNode,:);
7 bdDof = [bdNode; bdNode+N]; freeDof = [freeNode; freeNode+N];
8 u = zeros(2*N,1); uD = g_D(pD); u(bdDof) = uD(:);
9 ff = ff - kk*u;

```

7.3.6 程序整理

数值解和精确解的图像如下



主程序为

CODE 18. main_elasticityVEM_Navier.m

```
1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [200, 400, 600, 800, 1000];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10
11 %% PDE data
12 pde = elasticitydata();
13
14 %% Virtual element method
15 for k = 1:maxIt
16     % load mesh
17     load( [ 'meshdata' , num2str(nameV(k)) , '.mat' ] );
18     % get boundary information
19     bdStruct = setboundary(node,elem);
20     % solve
21     [u,info] = elasticityVEM_Navier(node,elem,pde,bdStruct);
22     % record and plot
23     N(k) = length(u); h(k) = 1/sqrt(size(elem,1));
24     if size(elem,1)<2e3
25         figure(1); clf;
26         subplot(1,2,1), showmesh(node,elem);
27         subplot(1,2,2), showsolution(node,elem,u(1:size(node,1)));
28     end
29     % compute errors in discrete L2 and H1 norms
30     index = info.elem2dof; % elemenwise global index
31     chi = cellfun(@(id) u(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
32     kOrder = 1;
33     Ph = info.Ph;
```

```

34 ErrL2(k) = getL2error_vector(node, elem, Ph, chi, pde, kOrder);
35 ErrH1(k) = getH1error_vector(node, elem, Ph, chi, pde, kOrder);
36 end
37
38 %% Plot convergence rates and display error table
39 figure(2);
40 showrateh(h, ErrL2, ErrH1);
41
42 fprintf('\n');
43 disp('Table: Error')
44 colname = {'#Dof', 'h', '|||u-u_h|||', '||Du-Du_h|||'};
45 disptable(colname, N, [], h, '%0.3e', ErrL2, '%0.5e', ErrH1, '%0.5e');

```

函数文件见 GitHub.

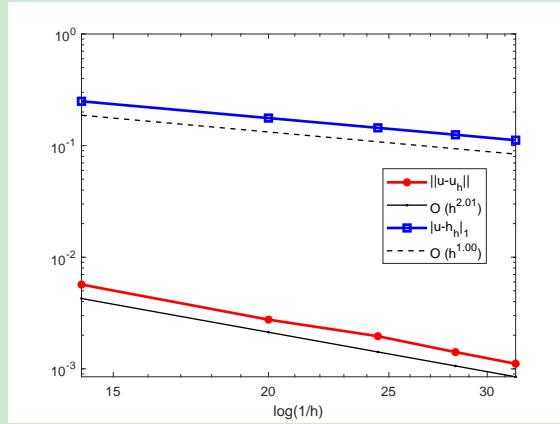


图 14. 线弹性问题 Navier 形式一阶虚拟元方法的误差阶

可以看到, L^2 的误差阶为 2, H^1 的误差阶为 1.

7.4 变分形式 II: tensor 型

7.4.1 椭圆投影的定义

以下的分部积分公式 (7.37) 可直接参考节 7.1.2 的计算过程 (式 (7.29)). 设 $\mathbf{u}_h \in V_k(K)$, $\mathbf{p} \in (\mathbb{P}_k(K))^2$, 分部积分有

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

由对称性,

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

此即

$$a_\mu^K(\mathbf{u}_h, \mathbf{p}) = \int_K \boldsymbol{\varepsilon}(\mathbf{u}_h) : \boldsymbol{\varepsilon}(\mathbf{p}) dx = - \int_K \operatorname{div}(\boldsymbol{\varepsilon}(\mathbf{p})) \cdot \mathbf{u}_h dx + \int_{\partial K} (\boldsymbol{\varepsilon}(\mathbf{p}) \mathbf{n}) \cdot \mathbf{u}_h ds, \quad (7.37)$$

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

为此, 定义椭圆投影

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

满足

$$a_\mu^K(\Pi^a \mathbf{v}, \mathbf{p}) = a_\mu^K(\mathbf{v}, \mathbf{p}), \quad \mathbf{p} \in (\mathbb{P}_k(K))^2. \quad (7.38)$$

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

$$\int_K \boldsymbol{\varepsilon}(\mathbf{p}) : \boldsymbol{\varepsilon}(\mathbf{p}) dx = \int_K \varepsilon_{ij}(\mathbf{p}) \varepsilon_{ij}(\mathbf{p}) dx = 0,$$

即

$$\begin{cases} \varepsilon_{11}(\mathbf{p}) = \partial_1 p_1 = 0, \\ \varepsilon_{22}(\mathbf{p}) = \partial_2 p_2 = 0, \\ \varepsilon_{12}(\mathbf{p}) = \varepsilon_{21}(\mathbf{p}) = \frac{1}{2}(\partial_1 p_2 + \partial_2 p_1) = 0. \end{cases}$$

由前两个条件可知

$$\begin{cases} p_1 = c_1 y + c_2, \\ p_2 = d_1 x + d_2, \end{cases}$$

代入最后一个条件有 $d_1 + c_1 = 0$, 即

$$\begin{cases} p_1 = -d_1 y + c_2, \\ p_2 = d_1 x + d_2 \end{cases}, \quad \mathbf{p} = c_2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + d_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + d_1 \begin{bmatrix} -y \\ x \end{bmatrix},$$

这表明

$$\mathbf{p} \in \operatorname{span} \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -y \\ x \end{bmatrix} \right\} =: K_0. \quad (7.39)$$

注意到上式中空间的自由度为 3, 为了保证唯一性, 必须施加三个限制条件.

限制条件一

一个简单的做法为

$$\sum_{i=1}^{N_v} (\Pi^a \mathbf{v}(z_i), \mathbf{p}(z_i)) = \sum_{i=1}^{N_v} (\mathbf{v}(z_i), \mathbf{p}(z_i)), \quad \mathbf{p} \in K_0, \quad (7.40)$$

式中的配对表示欧式内积.

限制条件二

也可以施加如下限制条件

$$\int_{\partial K} (\Pi^a \mathbf{v}) \cdot \mathbf{p} ds = \int_{\partial K} \mathbf{v} \cdot \mathbf{p} ds, \quad \mathbf{p} \in K_0. \quad (7.41)$$

考虑 $k = 1$, 设 $\mathbf{v} = (v_1, v_2)$, 则 v_l 在每条边上是 1 次多项式, 于是可设

$$v|_{e_i} = v(z_i)\phi_1 + v(z_{i+1})\phi_2, \quad v = v_1, v_2,$$

其中的 ϕ_i 是一维的节点基. 这样, 右端可计算.

限制条件三

第三个限制条件为

$$\begin{aligned} \int_K \nabla \times \Pi_K^1 \mathbf{v} dx &= \int_K \nabla \times \mathbf{v} dx, \\ \int_{\partial K} \Pi_K^1 \mathbf{v} ds &= \int_{\partial K} \mathbf{v} ds \end{aligned} \quad (7.42)$$

注意到

$$\int_K \nabla \times \mathbf{v} dx = \int_{\partial K} \mathbf{v} \cdot \boldsymbol{\tau}_K ds, \quad (7.43)$$

右端可由梯形公式计算.

定义 (7.38) 等价于如下的向量形式

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

或

$$\int_K \varepsilon_{ij}(\mathbf{m}) \varepsilon_{ij}(\Pi^a \boldsymbol{\phi}^T) dx = \int_K \varepsilon_{ij}(\mathbf{m}) \varepsilon_{ij}(\boldsymbol{\phi}^T) dx.$$

记 K_0 三个基的列向量为 $\{\mathbf{p}\}$, 则限制条件一为 (三行方程)

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

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

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

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

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

易知有

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

将以上表达式代入向量形式, 有

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

式中,

$$\mathbf{G} = a_{\mu}^K(\mathbf{m}, \mathbf{m}^T) = \int_K \varepsilon_{ij}(\mathbf{m}) \varepsilon_{ij}(\mathbf{m}^T) dx,$$

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

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

引理 7.2

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

7.4.2 椭圆投影的计算

当 $k = 1$ 时, $\varepsilon_{ij}(m_{\alpha})$ 是常数, 而且 (7.37) 右端的第一项为零, 于是

$$\mathbf{G} = |K| \cdot \varepsilon_{ij}(\mathbf{m}) \varepsilon_{ij}(\mathbf{m}^T), \quad \mathbf{B} = \int_{\partial K} (\boldsymbol{\varepsilon}(\mathbf{m}) \cdot \mathbf{n}) \cdot \boldsymbol{\phi}^T ds.$$

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

$$\begin{aligned} \varepsilon_{11}(\mathbf{m}) &= [0, \frac{1}{h_K}, 0, 0, 0, 0]^T, & \varepsilon_{22}(\mathbf{m}) &= [0, 0, 0, 0, 0, \frac{1}{h_K}]^T, \\ \varepsilon_{12}(\mathbf{m}) &= \varepsilon_{21}(\mathbf{m}) = [0, 0, \frac{1}{h_K}, 0, \frac{1}{h_K}, 0]^T, \end{aligned}$$

由此获得 \mathbf{G}

```

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

```

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

现计算 \mathbf{B} . 注意向量的内积, 有

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

于是

$$\begin{aligned} \mathbf{B}(i,:) &= \int_{\partial K} [(\varepsilon_{11}(\mathbf{m}_i) n_x + \varepsilon_{12}(\mathbf{m}_i) n_y) \boldsymbol{\phi}^T, (\varepsilon_{21}(\mathbf{m}_i) n_x + \varepsilon_{22}(\mathbf{m}_i) n_y) \boldsymbol{\phi}^T], \\ \mathbf{B} &= \int_{\partial K} [(\varepsilon_{11}(\mathbf{m}) n_x + \varepsilon_{12}(\mathbf{m}) n_y) \boldsymbol{\phi}^T, (\varepsilon_{21}(\mathbf{m}) n_x + \varepsilon_{22}(\mathbf{m}) n_y) \boldsymbol{\phi}^T]. \end{aligned}$$

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

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

注意到 (7.36), 即

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

从而可获得 \mathbf{B} . 或直接考慮

$$\int_{\partial K} \phi_i n_{i,x} ds = \frac{1}{2} \sum_{e_j \subset \partial K} n_{i,x} h_{e_j} (\phi_i(z_j) + \phi_i(z_{j+1})) = \frac{n_{i-1,x} h_{e_{i-1}} + n_{i,x} h_{e_i}}{2},$$

它在后面还要遇到.

限制条件一

若考慮限制条件 (7.40), 此时右端为 $\sum_{i=1}^{N_v} (\{\mathbf{p}(z_i)\}, \boldsymbol{\phi}^T(z_i))$, 直接计算有

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

由基函数的 Kronecker 性质,

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

它们要替换 \mathbf{B} 的 1,3,4 行, 从而获得 $\tilde{\mathbf{B}}$. 前面提到的矩阵如下计算

```

1 %% Compute projection matrices
2 D = cell(NT,1);
3 % B = cell(NT,1); % not used in the computation
4 Bs = cell(NT,1);
5 G = cell(NT,1); Gs = cell(NT,1);
6 HO = cell(NT,1); CO = cell(NT,1);
7 for iel = 1:NT
8     index = elem{iel}; Nv = length(index);
9     xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
10
11    m1 = @(x,y) 1 + 0*x; m2 = @(x,y) (x-xK)./hK; m3 = @(x,y) (y-yK)./hK;
12    m = @(x,y) [m1(x,y), m2(x,y), m3(x,y)];
13    x = node(index,1); y = node(index,2);
14
15    % D
16    D1 = m(x,y); D1 = blkdiag(D1,D1); D{iel} = D1;
17
18    % HO,CO
19    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
20    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of ...
        edge vector
21    HO{iel} = area(iel); CO1 = reshape(normVec',1,[]);
22    CO{iel} = CO1;
23
24    % B,Bs,G,Gs
25    E = zeros(6,4); % E = [E11,E12,E21,E22]
26    E(2,1) = 1/hK; E([3,5],[2,3]) = 1/hK; E(6,4) = 1/hK;
27    B1 = [E(:,1)*CO1(1:Nv)+E(:,2)*CO1(Nv+1:end), ...
28           E(:,3)*CO1(1:Nv)+E(:,4)*CO1(Nv+1:end)];
29    B0 = zeros(3,2*Nv); B0(1,1:Nv) = 1; B0(2,Nv+1:end) = 1;
30    B0(3,1:Nv) = -y; B0(3,Nv+1:end) = x;
31    % B{iel} = B1;

```

```

32      B1s = B1; B1s([1,3,4],:) = B0;    Bs{iel} = B1s;
33      G{iel} = B1*D1;        Gs{iel} = B1s*D1;
34 end

```

限制条件二

若考虑限制条件 (7.41), 则右端为

$$\int_{\partial K} \phi^T \cdot p \, ds = \left[\int_{\partial K} \phi^T p_1 \, ds, \quad \int_{\partial K} \phi^T p_2 \, ds \right].$$

相应的三行向量为

$$\left[\int_{\partial K} \phi^T \, ds, \quad \mathbf{0} \right], \quad \left[\mathbf{0}, \quad \int_{\partial K} \phi^T \, ds \right], \quad \left[-\int_{\partial K} \phi^T y \, ds, \quad \int_{\partial K} \phi^T x \, ds \right].$$

它们可用梯形公式或 Simpson 公式精确计算. 为了方便, 我们对边进行循环计算. 前面提到的矩阵如下计算

```

1 %% Compute projection matrices
2 D = cell(NT,1);
3 % B = cell(NT,1); % not used in the computation
4 Bs = cell(NT,1);
5 G = cell(NT,1); Gs = cell(NT,1);
6 H0 = cell(NT,1); C0 = cell(NT,1);
7 for iel = 1:NT
8     % element information
9     index = elem{iel}; Nv = length(index);
10    xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
11    x = node(index,1); y = node(index,2);
12    v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
13    Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors
14    he = sqrt(sum(Ne.^2,2));
15    ne = Ne./repmat(he,1,2); % outer normal vectors
16
17    % D
18    m1 = @(x,y) 1 + 0*x; m2 = @(x,y) (x-xK)./hK; m3 = @(x,y) (y-yK)./hK;
19    m = @(x,y) [m1(x,y), m2(x,y), m3(x,y)];
20    D1 = m(x,y); D1 = blkdiag(D1,D1); D{iel} = D1;
21
22    % H0,C0
23    rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
24    normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of ...
        edge vector
25    H0{iel} = area(iel); C01 = reshape(normVec',1,[]);
26    C0{iel} = C01;
27
28    % ---- B,Bs,G,Gs -----
29    % E = [E11,E12,E21,E22]
30    E = zeros(6,4);
31    E(2,1) = 1/hK; E([3,5],[2,3]) = 1/hK; E(6,4) = 1/hK;
32    % phinx, phiny
33    nx = ne(:,1); ny = ne(:,2);
34    id = [Nv,1:Nv-1];
35    phinx = 0.5*(nx(id).*he(id) + nx.*he)';
36    phiny = 0.5*(ny(id).*he(id) + ny.*he)';
37    % B1

```

```

38     B1 = [E(:,1)*phinx+E(:,2)*phiny, E(:,3)*phinx+E(:,4)*phiny];
39 % constraint
40 B0 = zeros(3,2*Nv);
41 base = eye(Nv);
42 phi = zeros(1,Nv); phix = zeros(1,Nv); phiy = zeros(1,Nv);
43 for i = 1:Nv % loop of edges
44     base1 = base(i,v1); base2 = base(i,v2); basec = (base1+base2)./.2;
45     x1 = x(v1(i)); x2 = x(v2(i)); xc = (x1+x2)/2;
46     y1 = y(v1(i)); y2 = y(v2(i)); yc = (y1+y2)/2;
47     phi = phi + he(i)*basec;
48     phix = phix + he(i)/6*(base1*x1 + 4*basec*xc + base2*x2);
49     phiy = phiy + he(i)/6*(base1*y1 + 4*basec*yc + base2*y2);
50 end
51 B0(1,1:Nv) = phi;
52 B0(2,Nv+1:end) = phi;
53 B0(3,:) = [-phi, phix];
54 % B{iel} = B1;
55 B1s = B1; B1s([1,3,4],:) = B0; Bs{iel} = B1s;
56 G{iel} = B1*D1; Gs{iel} = B1s*D1;
57 end

```

限制条件三

若考虑限制条件 (7.42), 第一式右端为

$$\int_K \nabla \times \phi^T dx = \int_{\partial K} \phi^T \cdot \tau_K ds = \int_{\partial K} [\phi^T \cdot t_x, \phi^T \cdot t_y] ds,$$

第二式右端为

$$\int_{\partial K} \phi^T ds = \begin{bmatrix} \int_{\partial K} \phi^T ds & \mathbf{0} \\ \mathbf{0} & \int_{\partial K} \phi^T ds \end{bmatrix}.$$

注意到

$$\int_{\partial K} \phi_i t_{i,x} ds = \frac{1}{2} \sum_{e_j \subset \partial K} t_{i,x} h_{e_j} (\phi_i(z_j) + \phi_i(z_{j+1})) = \frac{t_{i-1,x} h_{e_{i-1}} + t_{i,x} h_{e_i}}{2},$$

$$\int_{\partial K} \phi_i ds = \frac{1}{2} \sum_{e_j \subset \partial K} h_{e_j} (\phi_i(z_j) + \phi_i(z_{j+1})) = \frac{h_{e_{i-1}} + h_{e_i}}{2},$$

由此可得它们的计算. 前面提到的矩阵如下计算

```

1 %% Compute projection matrices
2 D = cell(NT,1);
3 % B = cell(NT,1); % not used in the computation
4 Bs = cell(NT,1);
5 G = cell(NT,1); Gs = cell(NT,1);
6 H0 = cell(NT,1); C0 = cell(NT,1);
7 for iel = 1:NT
8     % element information
9     index = elem{iel}; Nv = length(index);
10    xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
11    x = node(index,1); y = node(index,2);
12    v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
13    Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors
14    he = sqrt(sum(Ne.^2,2));
15    ne = Ne./repmat(he,1,2); % outer normal vectors
16    te = [-ne(:,2), ne(:,1)];

```

```

17
18 % D
19 m1 = @ (x,y) 1 + 0*x; m2 = @ (x,y) (x-xK)./hK; m3 = @ (x,y) (y-yK)./hK;
20 m = @ (x,y) [m1(x,y), m2(x,y), m3(x,y)];
21 D1 = m(x,y); D1 = blkdiag(D1,D1); D{iel} = D1;
22
23 % H0,CO
24 rotid1 = [Nv,1:Nv-1]; rotid2 = [2:Nv,1]; % ending and starting indices
25 normVec = 0.5*[y(rotid2) - y(rotid1), x(rotid1)-x(rotid2)]'; % a rotation of ...
    edge vector
26 H0{iel} = area(iel); C01 = reshape(normVec',1,[]);
27 C0{iel} = C01;
28
29 % ----- Bs,Gs -----
30 % E = [E11,E12,E21,E22]
31 E = zeros(6,4);
32 E(2,1) = 1/hK; E([3,5],[2,3]) = 1/hK; E(6,4) = 1/hK;
33 % phinx, phiny
34 nx = ne(:,1); ny = ne(:,2);
35 id = [Nv,1:Nv-1];
36 phinx = 0.5*(nx(id).*he(id) + nx.*he)';
37 phiny = 0.5*(ny(id).*he(id) + ny.*he)';
38 % B1
39 B1 = [E(:,1)*phinX+E(:,2)*phiny, E(:,3)*phinX+E(:,4)*phiny];
40 % first constrain (rot)
41 B0 = zeros(3,2*Nv);
42 tx = te(:,1); ty = te(:,2);
43 phitx = 0.5*(tx(id).*he(id) + tx.*he)';
44 phity = 0.5*(ty(id).*he(id) + ty.*he)';
45 B0(1,:) = [phitx, phity];
46 % second constraint
47 phi = 0.5*(he(id) + he)';
48 B0(2,1:Nv) = phi; B0(3,Nv+1:end) = phi;
49 % B{iel} = B1;
50 Bs{iel} = B1; Bs{iel}([1,3,4],:) = B0; Bs{iel} = Bs{iel}*D1;
51 G{iel} = B1*D1; Gs{iel} = Bs{iel}*D1;
52 end

```

7.4.3 边界条件的处理

Neumann 边界条件与有限元的处理完全相同, 如下

```

1 %% Assemble Neumann boundary conditions
2 bdEdgeN = bdStruct.bdEdgeN;
3 if ~isempty(bdEdgeN)
4     g_N = pde.g_N;
5     z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
6     e = z1-z2; % e = z2-z1
7     ne = [-e(:,2),e(:,1)]; % scaled ne
8     Sig1 = g_N(z1); Sig2 = g_N(z2);
9     F11 = sum(ne.*Sig1(:,[1,3]),2)./2; F12 = sum(ne.*Sig2(:,[1,3]),2)./2; % g1
10    F21 = sum(ne.*Sig1(:,[3,2]),2)./2; F22 = sum(ne.*Sig2(:,[3,2]),2)./2;
11    FN = [F11,F12,F21,F22];
12    ff = ff + accumarray([bdEdgeN(:,1); bdEdgeN(:,2)+N], FN(:,[2*N 1]));
13 end

```

Dirichlet 条件的处理同 Navier 形式.

7.4.4 程序整理

两种限制条件的计算结果相当, 这里仅给出第一种. 数值结果与位移型差不多, 这里省略. 主程序如下

CODE 19. main_elasticityVEM_tensor.m

```
1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [200, 400, 600, 800, 1000];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10
11 %% PDE data
12 pde = elasticitydata();
13 bdNeumann = 'abs(x-0)<1e-4 | abs(x-1)<1e-4';
14
15 %% Virtual element method
16 for k = 1:maxIt
17     % load mesh
18     load( ['meshdata', num2str(nameV(k)), '.mat'] );
19     % get boundary information
20     bdStruct = setboundary(node, elem, bdNeumann);
21     % solve
22     [u, info] = elasticityVEM_tensor(node, elem, pde, bdStruct);
23     % record and plot
24     N(k) = length(u); h(k) = 1/sqrt(size(elem,1));
25     if size(elem,1)<2e3
26         figure(1); clf;
27         subplot(1,3,1), showmesh(node, elem);
28         subplot(1,3,2), showsolution(node, elem, u(1:size(node,1)));
29         ue = pde.ueexact(node);
30         subplot(1,3,3), showsolution(node, elem, ue(:,1));
31     end
32     % compute errors in discrete L2 and H1 norms
33     index = info.elem2dof; % elemenwise global index
34     chi = cellfun(@(id) u(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
35     kOrder = 1;
36     Ph = info.Ph;
37     ErrL2(k) = getL2error_vector(node, elem, Ph, chi, pde, kOrder);
38     ErrH1(k) = getH1error_vector(node, elem, Ph, chi, pde, kOrder);
39 end
40
41 %% Plot convergence rates and display error table
42 figure(2);
43 showrateh(h, ErrL2, ErrH1);
44
45 fprintf('\n');
46 disp('Table: Error')
47 colname = {'#Dof', 'h', '|||u-u_h|||', '|||Du-Du_h|||'};
48 disptable(colname, N, [], h, '%0.3e', ErrL2, '%0.5e', ErrH1, '%0.5e');
```

函数文件见 GitHub.

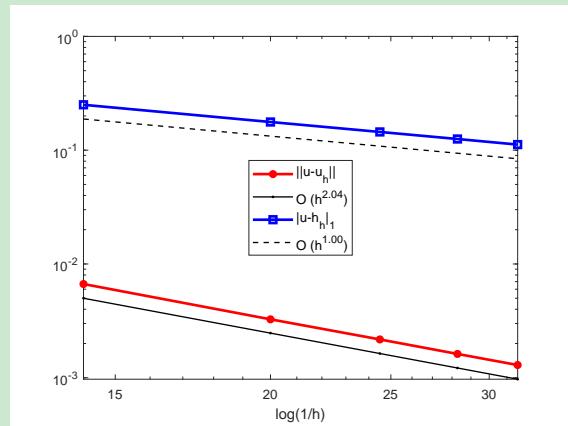


图 15. 线弹性问题 tensor 形式一阶虚拟元方法的误差阶

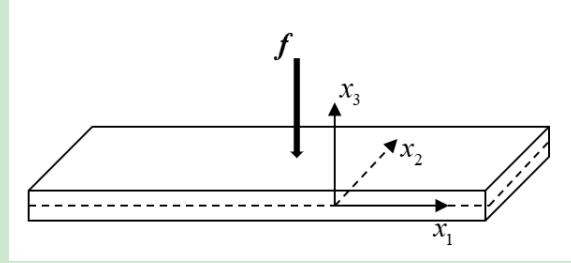
可以看到, L^2 误差阶为 2, H^1 误差阶为 1.

8 Kirchhoff 板弯问题的虚拟元方法

8.1 Kirchhoff 板弯问题

本节回顾一下 Kirchhoff 板弯问题.

8.1.1 模型假设



如图, 给定一个薄板 Ω , 其厚度为 t (远小于其他两个方向的长度). 称垂直于板平面的方向为薄板的横向, 其他两个方向都为纵向 (板平面内). 平分厚度的面称为中面 (或中立面), 以中面为 x_1-x_2 平面, 垂直于中面的方向为 x_3 轴建立直角坐标系, 则上下板面分别位于 $x_3 = \pm t/2$ 的平面内.

当薄板弯曲时, 中面所弯成的曲面, 称为薄板弹性曲面. 注意, 中面是未弯曲时板厚中间的面, 而板弯曲后对应称为弹性曲面 (不过有些书中仍把弹性曲面称为中面, 这里区分开).

称薄板的横向位移为挠度, 有时候直接把中面的横向位移称为挠度, 并记为 w (见后面的假设). 本节考虑的薄板弯曲问题是针对小挠度薄板而言的, 它在变形过程中横向的挠度远小于板的厚度 t .

对薄板弯曲问题, 我们做一些假设, 以简化模型.

假设 1: $\sigma_{31} = \sigma_{32} = \sigma_{33} = 0$

实验结果表明, 应力张量 $\boldsymbol{\sigma} = (\sigma_{ij})$ 的 6 个独立分量 $\sigma_{11}, \sigma_{12}, \sigma_{22}, \sigma_{31}, \sigma_{32}, \sigma_{33}$ 并不是一个量级. $\sigma_{11}, \sigma_{12}, \sigma_{22}$ 是主要应力, σ_{31}, σ_{32} 是次要应力, σ_{33} 是更次要的应力. 所以, 通常假设 $\sigma_{31} = \sigma_{32} = \sigma_{33} = 0$ (即忽略它们).

假设 2: 挠度沿板厚是一致的, 即

$$u_3(x_1, x_2, x_3) = u_3(x_1, x_2) = w(x_1, x_2).$$

正因为如此, 对薄板问题而言, 可以称中面的位移为挠度.

假设 3: 中面上的点没有水平方向的位移, 即

$$u_1^{(0)} = u_2^{(0)} = 0 \quad (u_3^{(0)} = w).$$

对上面的假设, 我们给出一些直观上的理解.

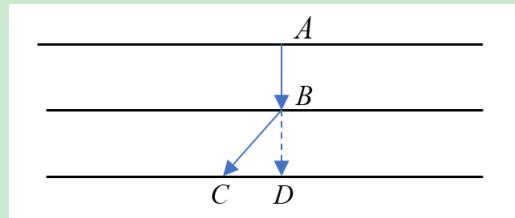
1. 假设 1 的直观理解

- 应力可以理解为对应方向变形能力的衡量. 应力越大, 对应方向阻碍变形的能力越大, 从而越不容易变形.

- 薄板的变形主要是板厚方向的弯曲, 水平方向的伸缩很小, 这意味着水平方向对应的应力很大, 而板厚方向的应力相对较小. 正因为如此, $\sigma_{11}, \sigma_{12}, \sigma_{22}$ 是主要应力.
- $\sigma_{31} = \sigma_{32} = \sigma_{33} = 0$ 的意思是, 这些应力相对很小, 加不加入到方程中影响不大, 从而可以忽略. 而忽略它们的直接办法是令其为零.

2. 假设 2 的直观理解

- 挠度沿板厚一致, 本质是说板的每层之间没有沿板厚方向的挤压, 从而初始法线方向的各点的板厚方向距离保持不变.

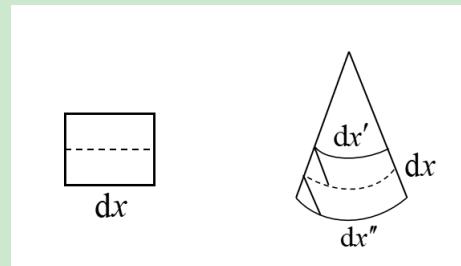


如图, 假设 A 移到 B , B 移到 C , 它们在竖直方向的位移相同 (即只有水平方向的位移).

- 这个假设显然意味着 x_3 方向的应变为零, 即 $\varepsilon_{33} = 0$.

3. 假设 3 的直观理解

- 板中质点发生水平位移, 即有水平方向上的形变 (相对距离变化).



如图, 左侧是一个体积单元, 在弯曲的过程中, 右侧虚线上侧的曲线比虚线要短, 即有压缩, 而下侧的要长, 即有拉伸.

- 薄板弯曲过程中, 在板的向外凸的一面是拉伸, 凹的一侧是压缩. 从拉伸逐步变到压缩, 中间存在一个没有水平伸缩的面 (连续性). 假设这个面就是中位面.
- 为此, 我们假设中面没有发生水平方向的伸缩, 而只有板厚方向上的位移.

注 8.1 $\sigma_{31}, \sigma_{32}, \sigma_{33}$ 所引起的形变效应可以忽略不计. 但对平衡方程, 仍需要考虑它们的作用.

引理 8.1 在以上假设下, 薄板弯曲变形完全取决于中面的横向位移 w , 即挠度, 它只依赖于坐标 x_1, x_2 .

证明 只需要说明位移完全由挠度决定即可 (此时自然有应变和旋转张量由挠度决定). 设 $\varepsilon_{3i} = 0$, 则

$$\partial_3 u_i = -\partial_i u_3 = -\partial_i w, \quad i = 1, 2,$$

积分得

$$u_i(x_1, x_2, x_3) = u_i(x_1, x_2, 0) - x_3 \partial_i w, \quad i = 1, 2.$$

而

$$u_i(x_1, x_2, 0) = u_i^{(0)} = 0,$$

所以

$$u_i(x_1, x_2, x_3) = -x_3 \partial_i w, \quad i = 1, 2.$$

即证. \square

根据上面的讨论, 我们给出一些基本关系式.

1. 薄板的位移 \mathbf{u} 可用中面的横向位移或挠度 w 表示为

$$\begin{cases} u_i(x_1, x_2, x_3) = -x_3 \partial_i w, & i = 1, 2, \\ u_3(x_1, x_2, x_3) = u_3(x_1, x_2) = w(x_1, x_2). \end{cases}$$

2. 应变通过挠度 ω 表示为

$$\varepsilon_{ij} = -x_3 \partial_{ij} w, \quad i, j = 1, 2.$$

3. 记

$$K_{ij}(x_1, x_2) = -\partial_{ij} w, \quad i, j = 1, 2,$$

它是中面经过弯曲后曲率张量的一阶近似, 以下简称为曲率张量. 于是,

$$\varepsilon_{ij} = x_3 K_{ij}, \quad K_{ij} = K_{ji}, \quad i, j = 1, 2. \quad (8.44)$$

注 8.2 从前面的讨论可知, 对薄板弯曲问题, 横向相关的应力分量 $\sigma_{31}, \sigma_{32}, \sigma_{33}$ 可以忽略不计, 且横向正应变 ε_{33} 也忽略不计. 而由 Hooke 定律可知, σ_{31}, σ_{32} 忽略不计的情况下, 必然有 $\varepsilon_{31}, \varepsilon_{32}$ 也忽略不计. 这表明假设 1,2 实际上暗含 Hooke 定律中, 横向相关的应力分量和应变分量都忽略不计, 即令它们为零, 从而只剩下平面相关的应力分量和应变分量, 这就化为平面问题. 而假设 3 进一步把它归结为中面上的问题.

8.1.2 薄板弯曲的 Hooke 定律

1. 常规表达

- 一般的 Hooke 定律为

$$\varepsilon_{ij} = \frac{1+\nu}{E} \sigma_{ij} - \frac{\nu}{E} \sigma_{kk} \delta_{ij}, \quad i, j = 1, 2, 3. \quad (8.45)$$

忽略横向相关的应变 (和应力) 分量后, 只有三个本质的应变分量 $\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{12} = \varepsilon_{21}$, 满足

$$\varepsilon_{ij} = \frac{1+\nu}{E} \sigma_{ij} - \frac{\nu}{E} \sigma_{kk} \delta_{ij}, \quad i, j = 1, 2, \quad (8.46)$$

其中, k 对 1,2 求和. 这就相当于二维问题. 设

$$\boldsymbol{\varepsilon}' = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} \\ \varepsilon_{21} & \varepsilon_{22} \end{bmatrix}, \quad \boldsymbol{\sigma}' = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix},$$

则 Hooke 定律也可写为

$$\boldsymbol{\varepsilon}' = \frac{1+\nu}{E} \boldsymbol{\sigma}' - \frac{\nu}{E} (\sigma_{11} + \sigma_{22}) \mathbf{I} \quad (8.47)$$

- 类似三维情形, 可以给出 (8.46) 的相反结果. 在 (8.46) 中令 $i = j$ 并求和, 有

$$\varepsilon_{11} + \varepsilon_{22} = \frac{1-\nu}{E}(\sigma_{11} + \sigma_{22}).$$

代入 (8.47) 并简单整理有

$$\boldsymbol{\sigma}' = \frac{E}{1-\nu^2} ((1-\nu)\boldsymbol{\varepsilon}' + \nu\varepsilon_{kk}\mathbf{I})$$

或写为分量形式

$$\sigma_{ij} = \frac{E}{1-\nu^2} ((1-\nu)\varepsilon_{ij} + \nu\varepsilon_{kk}), \quad i, j = 1, 2, \quad k \in \{1, 2\}.$$

2. 曲率表达

把 (8.44) 代入前面的式子中, 有

$$\sigma_{ij} = \frac{Ex_3}{1-\nu^2} ((1-\nu)K_{ij} + \nu K_{kk}\delta_{ij}), \quad i, j = 1, 2, \quad k \in \{1, 2\}. \quad (8.48)$$

3. 弯矩表达

为了消去 (8.48) 中的 x_3 (弄成二维问题), 定义弯矩 (不能直接对 (8.48) 求积分, 否则因 x_3 是奇函数, 结果为零)

$$M_{ij}(x_1, x_2) = \int_{-t/2}^{t/2} x_3 \sigma_{ij} dx_3, \quad i, j = 1, 2,$$

则薄板弯曲的 Hooke 定律也可表为

$$M_{ij} = D ((1-\nu)K_{ij} + \nu K_{kk}\delta_{ij}), \quad i, j = 1, 2, \quad k \in \{1, 2\},$$

其中,

$$D = \frac{Et^3}{12(1-\nu^2)}$$

称为抗弯刚度, 而曲率 K_{ij} 相当于应变, 弯矩 M_{ij} 相当于应力.

8.1.3 变分原理

1. 应变能

- 用曲率表达的应变能体密度为

$$W = \frac{1}{2} \frac{Ex_3}{1-\nu^2} \left((1-\nu)K_{ij}^2 + \nu(K_{kk})^2 \right), \quad i, j = 1, 2, \quad k \in \{1, 2\},$$

从而有应变能面密度

$$\overline{W}(x_1, x_2) = \int_{-t/2}^{t/2} W dx_3 = \frac{1}{2} D \left((1-\nu)K_{ij}^2 + \nu(K_{kk})^2 \right).$$

- 可以看到,

$$M_{ij} = \frac{\partial \overline{W}}{\partial K_{ij}}, \quad \overline{W} = \frac{1}{2} M_{ij} K_{ij}, \quad i, j \in \{1, 2\}.$$

- 板的弯曲应变能为

$$P(w) = \int_{\Omega} \overline{W}(w) dx_1 dx_2 = \frac{1}{2} \int_{\Omega} M_{ij}(w) K_{ij}(w) dx_1 dx_2,$$

这里 Ω 可认为是板的中面 (不是三维的薄板).

2. 外功势能

- 暂时不考虑几何约束以及弹性支承.
- 板面 Ω 上的横向载荷为 P_3 , 边界 $\partial\Omega$ 上的横向载荷为 q_3 , 边界上弯矩载荷为 m_t .
- 外功势能为

$$-F(w) = - \left\{ \int_{\Omega} P_3 w dx_1 dx_2 + \oint_{\partial\Omega} q_3 w dl + \oint_{\partial\Omega} m_t w_t dl \right\},$$

其中, w_t 表示弯曲变形时边界的切向转角.

- 设 $\mathbf{n} = (n_1, n_2, 0)^T$ 是边界的外方向, $\mathbf{e}_3 = (0, 0, 1)^T$ 是正 x_3 轴的单位向量, 边界的正切向 $\mathbf{t} = (t_1, t_2, 0)^T$ 使得 $\{\mathbf{n}, \mathbf{t}, \mathbf{e}_3\}$ 形成右手系.

在边界上, 弯曲后的板相对于弯曲前的平板有无穷小旋转, 其绕 x_1, x_2 轴的分量分别为

$$w_1 = \partial_2 w, \quad w_2 = -\partial_1 w,$$

所以转角的切向分量或切向转角为

$$w_t = t_1 w_1 + t_2 w_2 = -n_2 \partial_2 w + n_1 (-\partial_1 w) = -\partial_{\mathbf{n}} w.$$

于是,

$$-F(w) = - \left\{ \int_{\Omega} P_3 w dx_1 dx_2 + \oint_{\partial\Omega} q_3 w dl - \oint_{\partial\Omega} m_t \partial_{\mathbf{n}} w dl \right\}.$$

3. 变分形式

令

$$\begin{aligned} D(w, v) &= \int_{\Omega} M_{ij}(w) K_{ij}(v) dx_1 dx_2, \\ F(v) &= \int_{\Omega} P_3 v dx_1 dx_2 + \oint_{\partial\Omega} q_3 v dl - \oint_{\partial\Omega} m_t \partial_{\mathbf{n}} v dl, \end{aligned}$$

则变分形式为

$$D(w, v) = F(v).$$

8.1.4 平衡方程

从变分形式导出平衡方程的思路就是, 运用分部积分公式, 把 $D(w, v)$ 中检验函数 v 的导数迁移到 w 上 (Ω 上的积分), 从而可以合并, 由 v 的任意性 (变分学基本引理) 获得平衡方程及边界条件.

分部积分有

$$\begin{aligned} D(w, v) &= \int_{\Omega} M_{ij}(w) K_{ij}(v) dx_1 dx_2 = - \int_{\Omega} M_{ij}(w) \partial_{ij} v dx_1 dx_2 \\ &= \int_{\Omega} \partial_j M_{ij}(w) \partial_i v dx_1 dx_2 - \oint_{\partial\Omega} M_{ij}(w) n_j \partial_i v dl \end{aligned}$$

注意到

$$\partial_i v = n_i \partial_{\mathbf{n}} v + t_i \partial_l v,$$

并令

$$Q_{3i} = \partial_j M_{ij} \quad (i = 1, 2), \quad M_{\mathbf{nn}} = M_{ij} n_i n_j, \quad M_{t\mathbf{n}} = M_{ij} t_i n_j,$$

我们有

$$D(w, v) = \int_{\Omega} Q_{3i}(w) \partial_i v dx_1 dx_2 - \oint_{\partial\Omega} M_{\mathbf{nn}}(w) \partial_{\mathbf{n}} v dl - \oint_{\partial\Omega} M_{t\mathbf{n}}(w) \partial_l v dl.$$

将右端第一项继续分部积分有

$$\int_{\Omega} Q_{3i}(w) \partial_i v dx_1 dx_2 = - \int_{\Omega} \partial_i Q_{3i}(w) v dx_1 dx_2 + \oint_{\partial\Omega} Q_{3i}(w) n_i v dl.$$

令

$$Q_{3\mathbf{n}} = Q_{3i} n_i,$$

则

$$\begin{aligned} D(w, v) &= - \int_{\Omega} \partial_i Q_{3i}(w) v dx_1 dx_2 + \oint_{\partial\Omega} Q_{3\mathbf{n}}(w) v dl \\ &\quad - \oint_{\partial\Omega} M_{\mathbf{nn}}(w) \partial_{\mathbf{n}} v dl - \oint_{\partial\Omega} M_{t\mathbf{n}}(w) \partial_l v dl. \end{aligned}$$

在 $\partial\Omega$ 上, $\partial_l v$ 完全取决于其上的 v , 而 $\partial_{\mathbf{n}} v$ 则不同, 它还涉及到内部值. 所以最后一项可继续进行分部积分以消去 $\partial_l v$. 设边界 $\partial\Omega$ 上有角点 P_1, \dots, P_m (即切线跳跃之点), 分段分部积分可知 (看成一维普通积分)

$$\oint_{\partial\Omega} M_{t\mathbf{n}}(w) \partial_l v dl = - \oint_{\partial\Omega} \partial_l M_{t\mathbf{n}}(w) v dl - \sum_{i=1}^m [M_{t\mathbf{n}}(w)]|_{P_i^-}^{P_i^+} v(P_i).$$

定义跳量

$$[M_{t\mathbf{n}}(w)](P_i) = M_{t\mathbf{n}}(w)|_{P_i^-}^{P_i^+},$$

则

$$\begin{aligned} D(w, v) &= - \int_{\Omega} \partial_i Q_{3i}(w) v dx_1 dx_2 + \oint_{\partial\Omega} (Q_{3\mathbf{n}}(w) + \partial_l M_{t\mathbf{n}}(w)) v dl \\ &\quad - \oint_{\partial\Omega} M_{\mathbf{nn}}(w) \partial_{\mathbf{n}} v dl + \sum_{i=1}^m [M_{t\mathbf{n}}(w)](P_i) v(P_i). \end{aligned}$$

于是,

$$\begin{aligned} D(w, v) - F(v) &= - \int_{\Omega} (\partial_i Q_{3i}(w) + P_3) v dx_1 dx_2 + \oint_{\partial\Omega} (Q_{3\mathbf{n}}(w) + \partial_l M_{t\mathbf{n}}(w)) v dl \\ &\quad + \oint_{\partial\Omega} (-M_{\mathbf{nn}}(w) + m_t) \partial_{\mathbf{n}} v dl + \sum_{i=1}^m [M_{t\mathbf{n}}(w)](P_i) v(P_i) = 0. \end{aligned}$$

由此可得平衡方程和边界条件为

$$\Omega : -\partial_i Q_{3i}(w) = P_3, \tag{8.49}$$

$$\partial\Omega : \begin{cases} Q_{3\mathbf{n}}(w) + \partial_l M_{t\mathbf{n}}(w) = q_3, \\ M_{\mathbf{nn}}(w) = m_t, \\ [M_{t\mathbf{n}}(w)](P_i) = 0, \quad i = 1, \dots, m. \end{cases}$$

代入前面的一些记号, 平衡方程用挠度表示为

$$\Omega : -\partial_{ij}M_{ij}(w) = P_3,$$

式中,

$$M_{ij} = D((1-\nu)K_{ij} + \nu K_{kk}\delta_{ij}) = -D((1-\nu)\partial_{ij}w + \nu\partial_{kk}w\delta_{ij}),$$

指标范围为 $i, j = 1, 2, k \in \{1, 2\}$. 该方程是挠度 w 的四阶椭圆型偏微分方程.

按习惯, 后面记 $f = P_3$. 若板面与地基有弹性耦合, 则平衡方程可改写为

$$\Omega : -\partial_{ij}M_{ij}(w) + cw = f, \quad (8.50)$$

其中 c 为弹性耦合常数.

当 D, ν 为常数时, 含有 ν 的项会抵消, 上述方程简化为双调和方程

$$D\Delta^2 w = f.$$

注意, 对任意的 ν 都是如此, 特别地, 取 $\nu = 0$ (工程中是不允许的), 原来的平衡方程就是 $D\partial_{ij}\partial_{ij}w = f$, 显然 $\partial_{ij}\partial_{ij} = \Delta^2$. 为了方便, 以下将坐标 (x_1, x_2) 改记为 (x, y) .

本文考虑强加边界条件

$$w = \partial_n w = 0, \quad (x, y) \in \partial\Omega.$$

8.1.5 分部积分公式

在平衡方程 (8.50) 两边乘以检验函数 v 并分部积分后可获得变分问题, 此时会出现复杂的边界项. 但当 $v \in H_0^2(\Omega)$ 时, 这些边界项都会消失 (见前面的分部积分计算过程).

变分问题为: 找 $w \in V := H_0^2(\Omega)$ 使得

$$a(w, v) = \ell(v), \quad v \in V,$$

式中,

$$\begin{aligned} a(w, v) &= \int_{\Omega} M_{ij}(w)K_{ij}(v)dxdy + \int_{\Omega} cwvdxdy, \\ \ell(v) &= \int_{\Omega} fv dxdy. \end{aligned}$$

以下设 $c = 0$. 注意,

- 当 $\nu = 0, D = 1$ 时,

$$M_{ij}(v)K_{ij}(w) = D\partial_{ij}v\partial_{ij}w = D\nabla^2 v : \nabla^2 w = \nabla^2 v : \nabla^2 w.$$

- 当 $\nu = 1, D = 1$ 时,

$$M_{ij}(v)K_{ij}(w) = D(\partial_{11}v + \partial_{22}v)(\partial_{11}w + \partial_{22}w) = \Delta v \Delta w.$$

设 $\mathbf{n} = (n_1, n_2, 0)^T$ 是边界的外法向量, $\mathbf{e}_3 = (0, 0, 1)^T$ 是正 x_3 轴的单位向量, 边界的正切向量 $\mathbf{t} = (t_1, t_2, 0)^T$ 使得 $\{\mathbf{n}, \mathbf{t}, \mathbf{e}_3\}$ 形成右手系 (根据旋转关系有 $\mathbf{t} = (t_1, t_2) = (-n_2, n_1)$). 令 (注意求和约定)

$$\begin{aligned} Q_{3i} &= M_{ij,j}, \quad Q_{3n} = Q_{3i} n_i, \\ M_{nn} &= M_{ij} n_i n_j, \quad M_{tn} = M_{ij} t_i n_j, \end{aligned}$$

其中,

$$M_{ij} = D((1 - \nu)K_{ij} + \nu K_{kk}\delta_{ij}), \quad K_{ij} = -\partial_{ij}w,$$

则有

$$\begin{aligned} a^K(p, v) &= - \int_K Q_{3i,i}(p)v dx_1 dx_2 \\ &\quad + \int_{\partial K} (Q_{3n}(p) + \partial_t M_{tn}(p))v ds - \int_{\partial K} M_{nn}(p)\partial_n v ds \\ &\quad + \sum_{i=1}^{N_v} [M_{tn}(p)](z_i)v(z_i) \\ &=: I_1 + I_2 + I_3 + I_4. \end{aligned} \tag{8.51}$$

- 直接计算有

$$I_1 = - \int_K Q_{3i,i}(p)v dx = \int_K D(\Delta^2 p)v dx.$$

当 $p \in \mathbb{P}_2$ 时, $I_1 = I_2 = 0$.

- 当 $p \in \mathbb{P}_2$ 时, $M_{nn}(p)$ 在每条边上是常数.
- I_4 这一项不可去掉, 因为 $M_{tn}(p)(z_i)$ 不连续.

注意,

- 当 $\nu = 0, D = 1$ 时, 分部积分公式可写为

$$\begin{aligned} \int_K \nabla^2 p : \nabla^2 v dx &= \int_K \Delta^2 p v dx \\ &\quad - \int_{\partial K} \partial_n(\Delta p)v ds + \int_{\partial K} (\Delta p - \partial_t^2 p)\partial_n v ds \\ &\quad + \int_{\partial K} \partial_{tn}^2 p \partial_t v ds. \end{aligned} \tag{8.52}$$

- 当 $\nu = 1, D = 1$ 时, 分部积分公式可写为

$$\int_K \Delta p \Delta v dx = \int_K \Delta^2 p v dx - \int_{\partial K} \partial_n(\Delta p)v ds + \int_{\partial K} \Delta p \partial_n v ds. \tag{8.53}$$

- 显然有

$$\int_K \nabla^2 p : \nabla^2 v dx = \int_K \Delta p \Delta v dx - \int_{\partial K} \partial_t^2 p \partial_n v ds + \int_{\partial K} \partial_{tn}^2 p \partial_t v ds.$$

8.2 C^0 连续的非协调 VEM

8.2.1 C^0 连续的非协调虚拟元空间

1. VEM 函数的连续性由边界元空间决定. 考虑问题

$$\begin{cases} \Delta^2 v = f & \text{in } K, \\ v = g_1, \quad \Delta v = g_2, & \text{on } \partial K. \end{cases}$$

由 PDE 理论知, 当 f 和边界函数适当光滑, K 是凸多角形时, 解 v 存在且是 C^1 的. 为此, VEM 函数的光滑性只要考虑边界元即可. 注意, 内部所有点任意光滑的函数, 它在边界也可能不是光滑的. 例如, 三角形上的一次多项式, 沿顶点两侧边的切向导数一般不相等.

2. C^0 连续

- 设 e 是两个单元的公共边, 我们要给定 e 上的自由度, 使得 v 在不同单元考虑时, $v|_e$ 是同一个函数, 即这些自由度能唯一确定 $v|_e$.
- 用自由度确定的函数自然是有限维函数, 最简单的是多项式函数, 为此要求 $v|_e \in \mathbb{P}_k(e)$.
- 设

$$v|_e(s) = c_0 m_0(s) + \cdots + c_k m_k(s), \quad m_i(s) \in \mathbb{M}_k(e),$$

则有自由度

$$\chi_e(v) = \frac{1}{|e|} \int_e m v ds, \quad m \in \mathbb{M}_k(e).$$

- 为了保证顶点处的连续性, 上面的两个自由度用顶点值替换, 从而 C^0 连续给出自由度

$$\chi_a(v) = v(z), \quad z \in \partial e,$$

$$\chi_e(v) = \frac{1}{|e|} \int_e m v ds, \quad m \in \mathbb{M}_{k-2}(e),$$

共有 $k+1$ 个.

3. 可计算性

- 考察分部积分公式 (8.51). 对 I_1 , 由 $\partial_i Q_{3i}(p) \in \mathbb{P}_{k-4}(K)$, 我们给出内部自由度

$$\chi_K(v) = \frac{1}{|K|} \int_K m v dx, \quad m \in \mathbb{M}_{k-4}(K).$$

- 对 I_2 , 注意到 $[Q_{3n}(p) + \partial_t M_{tn}(p)]|_e \in \mathbb{P}_{k-3}(e)$, 从而该项可由自由度表示 (实际上 $v|_e$ 已经能够显式计算出来). 注意, C^0 连续要求有直到 $k-2$ 阶的边界函数矩量, 而这里用不到 $k-2$ 阶的.

- 对 I_3 , 由 $M_{nn}(p)|_e \in \mathbb{P}_{k-2}(e)$, 我们添加自由度

$$\chi_{n_e}(v) = \int_e m \partial_n v ds, \quad m \in \mathbb{M}_{k-2}(e).$$

注意, 这里并未要求 $\partial_n v|_e$ 是多项式, 只是纯粹从可计算性角度来添加自由度.

4. 唯一可解性与虚拟元空间的构造

- 前面给出的自由度保证了 C^0 连续性与可计算性, 现在来确定相应的虚拟元空间. 设虚拟元空间为 $V_k(K)$, 它要由自由度唯一确定. 也就是说, 设 $v \in V_k(K)$ 且它的所有自由度值为零, 我们要保证 $v \equiv 0$.
- 由连续性的自由度为零可得 $v|_{\partial K} = 0$.
- 注意到 (8.52), 即

$$\int_K |\Delta v|^2 dx = \int_K v \Delta^2 v dx + \int_{\partial K} \partial_n v \Delta v ds - \int_{\partial K} v \partial_n (\Delta v) ds, \quad (8.54)$$

若要求

$$\Delta^2 v \in \mathbb{P}_{k-4}(K), \quad \Delta v|_e \in \mathbb{P}_{k-2}(e), \quad e \subset \partial K,$$

则右端前两项化为自由度, 最后一项为零, 因为 $v|_{\partial K} = 0$. 于是

$$\int_K |\Delta v|^2 dx = 0 \quad \Rightarrow \quad \Delta v = 0.$$

这样, 我们有边值问题

$$\begin{cases} \Delta v = 0, & x \in K, \\ v|_{\partial K} = 0, \end{cases}$$

由此可知 $v \equiv 0$.

上面的分析表明, 虚拟元空间可取为

$$V_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K), \quad v|_{\partial K} \in \mathbb{B}_k(\partial K)\},$$

其中, 边界元空间为

$$\mathbb{B}_k(\partial K) = \{v \in C^0(\partial K) : v|_e \in \mathbb{P}_k(e), \quad \Delta v|_e \in \mathbb{P}_{k-2}(e), \quad e \subset \partial K\}.$$

自由度包括

- 顶点处的函数值,

$$\chi_a(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 边上 v 的直到 $k-2$ 阶的矩量,

$$\chi_e(v) = \frac{1}{|e|} \int_e mv ds, \quad v \in \mathbb{M}_{k-2}(e).$$

- 边上 $\partial_n v$ 的直到 $k-2$ 阶的矩量,

$$\chi_{n_e}(v) = \int_e m \partial_n v ds, \quad m \in \mathbb{M}_{k-2}(e).$$

- 单元 K 上直到 $k-4$ 阶的矩量,

$$\chi_K(v) = \frac{1}{|K|} \int_K mv dx, \quad v \in \mathbb{M}_{k-4}(K).$$

注 8.3 前两种自由度决定了 $v|_{\partial K}$; 第三种自由度决定了 $\Pi_{k-2,e}^0(v|_e)$, $e \subset \partial K$; 最后一种自由度决定了 $\Pi_{k-4}^0 v$. 而且, 我们有

$$h_K^{-1} (\|\chi_1(v)\|_{l^2}^2 + \|\chi_2(v)\|_{l^2}^2)^{1/2} \asymp h_K^{-3/2} \left(\sum_{e \subset \partial K} \|\Pi_{k,e}^0 v\|_{0,e}^2 \right)^{1/2}, \quad (8.55)$$

$$h_K^{-1} \|\chi_3(v)\|_{l^2} \asymp h_K^{-1/2} \left(\sum_{e \subset \partial K} \|\Pi_{k-2,e}^0 \partial_n v\|_{0,e}^2 \right)^{1/2}, \quad (8.56)$$

$$h_K^{-1} \|\chi_4(v)\|_{l^2} \asymp h_K^{-2} \|\Pi_{k-4}^0 v\|_{0,K}. \quad (8.57)$$

本文仅考虑最低阶, 即 $k = 2$ 的情形 ($k \leq 1$ 椭圆投影定义的右端为零, 没有意义). 自由度包含顶点值泛函和边界的两种矩量

$$\chi_e(v) = \frac{1}{|e|} \int_e v \mathrm{d}s, \quad \chi_{n_e}(v) = \int_e \partial_n v \mathrm{d}s,$$

如下图所示

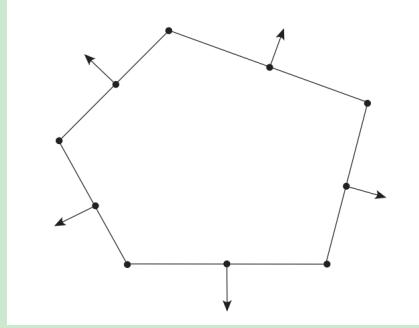


图 16. C^0 连续虚拟元空间的局部自由度 ($k = 2$)

注意边界的第一种矩量也可换成边界中点值, 而第二种矩量是带方向的. 设单元顶点个数为 N_v , 则每个单元有 $N_k = 3N_v$ 个自由度, 排列为

- 顶点值泛函

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 边中点值泛函

$$\chi_{i+N_v}(v) = v(m_i), \quad i = 1, \dots, N_v.$$

- 边中点法向导数的矩量

$$\chi_{2N_v+i}(v) = \int_{e_i} \partial_n v \mathrm{d}s, \quad i = 1, \dots, N_v.$$

注 8.4 文献 [3] 提出了三角形上九个自由度的改进 Morley 元 (见那里的 Figure 2, 称为 NTW 元), 可以看到那里的自由度与 C^0 元一致 ($k = 2$). 正因为如此, C^0 虚拟元可看作改进 Morley 元的推广. 需要注意的是, 改进 Morley 元的有限元空间与 C^0 元在三角形情形时并不相同. 正因为如此, 虚拟元方法的相同自由度对应的虚拟元空间不唯一. 虚拟元方法的核心是自由度, 虚拟元空间只是用来进行理论分析.

8.2.2 过渡矩阵 D

设 $V_k(K)$ 为 C^0 连续的非协调虚拟元空间, 基函数排成行向量, 记为

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

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

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

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

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

根据自由度的定义,

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

且 D 是 $N_k \times N_p$ 的矩阵.

$\mathbb{P}_2(K)$ 的尺度单项式集合 m^T 由以下元素组成

$$\begin{aligned} m_1(x, y) &= 1, \quad m_2(x, y) = \frac{x - x_K}{h_K}, \quad m_3(x, y) = \frac{y - y_K}{h_K}, \\ m_4(x, y) &= \frac{(x - x_K)^2}{h_K^2}, \quad m_5(x, y) = \frac{(x - x_K)(y - y_K)}{h_K^2}, \quad m_6(x, y) = \frac{(y - y_K)^2}{h_K^2}, \end{aligned}$$

从而 ∇m^T 为

$$\begin{aligned} \nabla m_1 &= [0, 0], \quad \nabla m_2 = [\frac{1}{h_K}, 0], \quad \nabla m_3 = [0, \frac{1}{h_K}], \\ \nabla m_4 &= [\frac{2(x - x_K)}{h_K^2}, 0], \quad \nabla m_5 = [\frac{y - y_K}{h_K^2}, \frac{x - x_K}{h_K^2}], \quad \nabla m_6(x, y) = [0, \frac{2(y - y_K)}{h_K^2}]. \end{aligned}$$

现在来计算过渡矩阵. 顶点值和边中点值比较容易, 如下计算

```

1 % element information
2 index = elem{iel};      Nv = length(index);      Ndof = 3*Nv;
3 xK = centroid(iel,1);  yK = centroid(iel,2);  hK = diameter(iel);
4 x = node(index,1);    y = node(index,2); % vertices
5 v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
6 xe = (x(v1)+x(v2))/2; ye = (y(v1)+y(v2))/2; % mid-edge
7 Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors
8 he = sqrt(sum(Ne.^2,2));
9 ne = Ne./repmat(he,1,2); % outer normal vectors
10 te = [-ne(:,2), ne(:,1)];
11 % scaled monomials
12 m = @ (x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK.^2, ...
13     (x-xK).*(y-yK)/hK.^2, (y-yK).^2/hK.^2]; % m1, ..., m6
14 Gradm = @ (x,y) [[0,0], [1, 0]/hK, [0, 1]/hK, [2*(x-xK), 0]/hK.^2;
15     [(y-yK), (x-xK)]/hK.^2, [0, 2*(y-yK)]/hK.^2]; % grad m
16 % D
17 D1 = zeros(Ndof,Nm);
18 D1(1:2*Nv,:) = [m(x,y); m(xe,ye)]; % vertices and mid-edge points

```

再考虑边的法向矩量. 注意到 ∇m_α 次数不超过一次, 可用梯形公式计算, 即

$$\chi_{2N_v+i}(m_\alpha) = \int_e \nabla m_\alpha \cdot \mathbf{n}_e ds = \frac{1}{2} (\nabla m_\alpha(z_i) + \nabla m_\alpha(z_{i+1})) \cdot \mathbf{n}_e |e|, \quad e = e_i.$$

这里, $\mathbf{n}_e |e|$ 恰是定向边的反向逆时针旋转 90 度所得. 该部分如下计算

```

1   for i = 1:Nv % normal moments on edges
2       gradi = 0.5*(Gradm(x(v1(i)),y(v1(i)))+Gradm(x(v2(i)),y(v2(i))));
3       D1(2*Nv+i,:) = Ne(i,:)*gradi';
4   end
5   D{iel} = D1;

```

8.2.3 椭圆投影

椭圆投影 Π^K 满足

$$\begin{cases} a^K(\Pi^K \phi_i, m_\alpha) = a^K(\phi_i, m_\alpha) \\ P_0(\Pi^K \phi_i) = P_0(\phi_i) \quad , \quad i = 1, \dots, N_k, \quad \alpha = 1, \dots, N_p, \\ P_0(\nabla \Pi^K \phi_i) = P_0(\nabla \phi_i) \end{cases}$$

写为向量形式为

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

注意, 第二个限制条件实际上有两个限制, 即 x 和 y 方向的两个导数. 对 $p \in \mathbb{P}_2$, 有

$$a^K(p, v) = - \int_{\partial K} M_{nn}(p) \partial_n v \, ds + \sum_{i=1}^{N_v} [M_{tn}(p)](z_i) v(z_i),$$

而

$$P_0 v = \frac{1}{N_v} \sum_{i=1}^{N_v} v(z_i) \quad \text{或} \quad P_0(v) = \int_{\partial K} v \, ds,$$

前者对应第一个限制条件, 后者对应第二个限制条件.

设 Π^K 在节点基下的矩阵记为 $\boldsymbol{\Pi}^K$, 即

$$\Pi^K(\phi_1, \phi_2, \dots, \phi_{N_k}) = (\phi_1, \phi_2, \dots, \phi_{N_k}) \boldsymbol{\Pi}^K \quad \text{或} \quad \Pi^K \phi^T = \phi^T \boldsymbol{\Pi}^K.$$

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

$$\boldsymbol{\Pi}^K = (\chi_i(\Pi^K \phi_j)).$$

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

$$\Pi^K \phi^T = m^T \boldsymbol{\Pi}_*^K.$$

两组不同基下的矩阵满足

$$\boldsymbol{\Pi}^K = D \boldsymbol{\Pi}_*^K.$$

令

$$G = a^K(m, m^T), \quad B = a^K(m, \phi^T),$$

则

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

其中, G 是 $N_p \times N_p$ 的方阵, B 是 $N_k \times N_p$ 的矩阵. 而 $P_0(m^T)$ 是一行向量, $P_0(\nabla m^T)$ 是两行向量. 显然 G 的前三行为零, 限制条件的三个行向量替换前三行后得

$$\tilde{G}\Pi_*^K = \tilde{B},$$

式中,

$$\tilde{G}(i,:) = \begin{cases} P_0(m^T), & i = 1 \\ P_0(\partial_x m^T), & i = 2 \\ P_0(\partial_y m^T), & i = 3 \\ G(i,:), & i \geq 4 \end{cases}, \quad \tilde{B} = \begin{cases} P_0(\phi^T), & i = 1 \\ P_0(\partial_x \phi^T), & i = 2 \\ P_0(\partial_y \phi^T), & i = 3 \\ B(i,:), & i \geq 4 \end{cases},$$

且

$$\Pi_*^K = \tilde{G}^{-1}\tilde{B}, \quad \Pi_k^K = D\Pi_*^K.$$

\tilde{G} 或 G 不需要直接计算, 有如下的一致性关系

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

注 8.5 椭圆投影不能对 $\nu = 1$ 定义, 容易发现, 这是因为

$$\Delta m_i = 0, \quad i = 1, 2, 3, 5,$$

从而 $G = (\Delta m, \Delta m^T)_K$ 至少有 4 行为零, 但是现在只有三个限制条件, 它不足以保证 \tilde{G} 可逆.

现在来计算 B , 其元素为

$$\begin{aligned} B_{\alpha j} &= a^K(m_\alpha, \phi_j) = - \sum_{e \subset \partial K} \int_e M_{nn}(m_\alpha) \partial_n \phi_j ds + \sum_{i=1}^{N_v} [M_{tn}(m_\alpha)](z_i) \phi_j(z_i) \\ &=: J_1(\alpha, j) + J_2(\alpha, j), \end{aligned} \tag{8.58}$$

其中,

$$M_{nn}(p) = M_{ij}(p)n_i n_j, \quad M_{tn}(p) = M_{ij}(p)t_i n_j,$$

$$M_{ij}(p) = -D((1-\nu)\partial_{ij}p + \nu(\partial_{11}p + \partial_{22}p)\delta_{ij}).$$

注意到

| | $m_\alpha(\alpha = 1, 2, 3)$ | m_4 | m_5 | m_6 |
|-----------------|------------------------------|-----------|-----------|-----------|
| ∂_{11} | 0 | $2/h_K^2$ | 0 | 0 |
| ∂_{12} | 0 | 0 | $1/h_K^2$ | 0 |
| ∂_{22} | 0 | 0 | 0 | $2/h_K^2$ |

由此可得 $M_{ij}(m_\alpha)$, 进而获得 M_{nn}, M_{tn} , 如下计算.

```

1 % \partial_ij (m)
2 D11 = zeros(Nm,1); D11(4) = 2/hK^2;
3 D12 = zeros(Nm,1); D12(5) = 1/hK^2;
4 D22 = zeros(Nm,1); D22(6) = 2/hK^2;
5 % Mij (m)
6 M11 = -para.D*((1-para.nu)*D11 + para.nu*(D11+D22));

```

```

7      M12 = -para.D*(1-para.nu)*D12;
8      M22 = -para.D*((1-para.nu)*D22 + para.nu*(D11+D22));
9      % Mnn(m) on e1,...,eNv
10     n1 = ne(:,1); n2 = ne(:,2);
11     Mnn = M11*(n1.*n1)' + M12*(n1.*n2+n2.*n1)' + M22*(n2.*n2)';
12     % Mtn(m) on e1,...,eNv
13     t1 = te(:,1); t2 = te(:,2);
14     Mtn = M11*(t1.*n1)' + M12*(t1.*n2+t2.*n1)' + M22*(t2.*n2)';

```

这里, M_{nn} 和 M_{tn} 行对应 m_α , 列对应 e_j .

对 J_1 , 因 $M_{nn}(m_\alpha)$ 在每条边上为常数, 故

$$J_1(\alpha, j) = - \sum_{e \subset \partial K} \int_e M_{nn}(m_\alpha) \partial_n \phi_j ds = - \sum_{e \subset \partial K} M_{nn}(m_\alpha) \int_e \partial_n \phi_j ds. \quad (8.59)$$

对固定的 $e = e_j$, 由基函数的 Kronecker 性质,

$$\int_e \partial_n \phi^T ds = [\mathbf{0}, \mathbf{0}, \mathbf{e}_j],$$

其中, $\mathbf{0}$ 是 N_v 长度的全零行向量, \mathbf{e}_j 是 N_v 长度的自然基向量. 显然用 M_{nn} 的第 j 列乘以上面的行向量, 所得为 J_1 在边 e_j 上的结果, 循环求和即得 J_1 .

对 J_2 , 即

$$J_2(\alpha, j) = \sum_{i=1}^{N_v} [M_{tn}(m_\alpha)](z_i) \phi_j(z_i),$$

由跳跃定义,

$$[M_{tn}(m_\alpha)](z_i) = M_{tn}(m_\alpha)|_{z_i^-}^{z_i^+} = M_{tn}(m_\alpha)(z_i^+) - M_{tn}(m_\alpha)(z_i^-),$$

这里, $M_{tn}(m_\alpha)(z_i^+)$ 是 z_i 右侧边 e_i 上的结果, $M_{tn}(m_\alpha)(z_i^-)$ 是左侧边 e_{i-1} 上的结果. 而

$$\phi^T(z_i) = [\mathbf{e}_i, \mathbf{0}, \mathbf{0}].$$

综上, B 可如下计算

```

1      B1 = zeros(Nm,Ndof);
2      p1 = [Nv,1:Nv-1]; p2 = 1:Nv;
3      for j = 1:Nv % loop of edges
4          % nphi on ej
5          nphi = zeros(1,Ndof); nphi(2*Nv+j) = 1;
6          % Jump at zj
7          Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
8          % phi at zj
9          phi = zeros(1,Ndof); phi(j) = 1;
10         % B1
11         B1 = B1 - Mnn(:,j)*nphi + Jump*phi;
12     end

```

再计算 B_s . 它的第一行容易获得, 显然为

$$\tilde{B}(1,:) = P_0(\phi^T) = \frac{1}{N_v} [\mathbf{1}, \mathbf{0}, \mathbf{0}]. \quad (8.60)$$

对剩下的限制条件, 注意到

$$\begin{aligned}
\int_{\partial K} \nabla v ds &= \sum_{e \subset \partial K} \int_e \partial_{n_e} v n_e ds + \int_e \partial_{t_e} v t_e ds \\
&= \sum_{e \subset \partial K} n_e \int_e \partial_n v ds + \sum_{i=1}^{N_v} t_{e_i} (v(z_{i+1}) - v(z_i)),
\end{aligned} \quad (8.61)$$

我们有

$$\begin{cases} \int_{\partial K} \nabla \phi_j \cdot \mathbf{n} ds = t_{e_{j-1}} - t_{e_j}, \\ \int_{\partial K} \nabla \phi_{N_v+j} \cdot \mathbf{n} ds = \mathbf{0}, \\ \int_{\partial K} \nabla \phi_{2N_v+j} \cdot \mathbf{n} ds = n_{e_j}. \end{cases}$$

```

1 % Bs
2 te = [-ne(:,2), ne(:,1)];
3 B1s = B1;
4 B1s(1,1:Nv) = 1/Nv;
5 B1s(2:3,1:Nv) = te([Nv,1:Nv-1],:) - te';
6 B1s(2:3,2*Nv+1:end) = ne';

```

有了 B, B_s, D , 我们就可用一致性关系计算 G, G_s .

```

1 % G, Gs
2 G{iel} = B1*D1;      Gs{iel} = B1s*D1;

```

8.2.4 单元刚度矩阵与载荷向量的计算

令

$$\begin{aligned} A_K^1(i, j) &= a^K(\Pi^K \phi_j, \Pi^K \phi_i), \\ A_K^2(i, j) &= S^K(\phi_j - \Pi^K \phi_j, \phi_i - \Pi^K \phi_i) \\ &= h_K^{-2} \sum_{r=1}^{N_k} \chi_r(\phi_j - \Pi^K \phi_j) \chi_r(\phi_i - \Pi^K \phi_i), \end{aligned}$$

则

$$\begin{aligned} A_K^1 &= a^K(\phi, \phi^T) = (\Pi_*^K)^T a^K(m, m^T) \Pi_*^K = (\Pi_*^K)^T G \Pi_*^K, \\ A_K^2 &= h_K^{-2} (\mathbf{I} - \Pi^K)^T (\mathbf{I} - \Pi^K). \end{aligned}$$

单元刚度矩阵如下计算

```

1 Aelem = cell(NT,1);
2 % Projection
3 Pis = Gs{iel}\Bs{iel};    Pi = D{iel}*Pis;    I = eye(size(Pi));
4 % Stiffness matrix
5 AK = Pis'*G{iel}*Pis + hK^(-2)*(I-Pi)'*(I-Pi);
6 Aelem{iel} = reshape(AK',1,[]); % straighten as row vector for easy assembly

```

现在考虑单元载荷向量. 右端的

$$F_h^K(v) = \int_K \Pi_{k-2}^0 f v dx,$$

当 $k = 2$ 时, $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (5.14) 定义, 即单元顶点值的平均. 注意到

$$F_h^K(v) = \int_K \Pi_0^0 f v dx = \int_K \Pi_0^0 f \Pi_0^0 v dx,$$

我们有单元载荷向量为

$$F_K = \int_K P_0 f P_0 \phi dx = f(x_K, y_K) |K| P_0 \phi,$$

这里为了方便, $P_0 f$ 用中点值近似, 而 $P_0 \phi$ 由 (8.64) 给出.

```

1 P0 = zeros(Ndof,1); P0(1:Nv) = 1/Nv;
2 fK = pde.f(cK)*areaK*P0;

```

8.2.5 符号刚度矩阵和载荷向量

局部自由度的后 N_v 个为边法向导数的矩量, 它是有方向的. 我们要取定一个方向, 从而相应修改刚度矩阵元素的符号. 注意到

$$a^K(\pm\varphi_i, \pm\varphi_j) = \pm \cdot \pm a^K(\varphi_i, \varphi_j),$$

为此可对应给出一个符号矩阵, 它记录 (i, j) 位置的符号.

1. 边的定向: 对单元的边 e , 规定逆时针方向为其在单元中的正向, 可用局部编号来确定.
2. 边的符号:
 - 对内部定向边, 称终点与起点整体编号差的符号为该边在单元中的符号.
 - 对边界边, 规定符号为正 (即 +1).
 - 显然对内部边, 限制在相邻两个单元上的符号相异, 因而可作为法向的符号.

下面说明如何获得固定单元边的符号, 用 `sgnelem` 存储.

- 执行如下语句可获得第 `iel` 个单元所有边的符号

```

1 index = elem{iel}; Nv = length(index);
2 v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
3 sgnelem = sign(index(v2)-index(v1));

```

这里边界边的符号可能不正确.

- 利用逻辑运算可修改 `sgnelem` 中边界边的符号差 (边界边符号差为 1).

```

1 sgnelem = sign(diff(index([1:Nv,1])));
2 id = elem2edge{iel}; sgnbd = E(id); sgnelem(sgnbd) = 1;
3 sgnbase = ones(Ndof,1); sgnbase(2*Nv+1:end) = sgnelem;
4 sgnK = sgnbase*sgnbase'; sgnF = sgnbase;

```

这里, `bdIndex` 是边界边的自然序号, `E` 内部边对应 `false`, 边界边对应 `true`.

上面的过程在单元刚度矩阵循环计算的过程中同步处理, 不用存储为元胞数组. 最终的单元刚度矩阵和载荷向量为

```

1 AK = AK.*sgnK; fK = fK.*sgnF;

```

8.2.6 边界条件的处理

我们考虑的是强加边界条件

$$w = \partial_n w = 0, \quad (x, y) \in \partial\Omega,$$

这里可以是非齐次的 (这两个条件都是 Dirichlet 边界条件).

$w|_{\partial\Omega} = g$ 与常规情形一样处理. 现考虑 $\partial_n w|_{\partial\Omega} = g_n$.

1. 第三种自由度为

$$\chi(v) = \int_e \partial_n v ds,$$

因 $\partial_n w$ 在边界线段 e 上的值已知, 故可用数值积分计算边界上的该类自由度.

2. 在 setboundary.m 中我们给出了 Dirichlet 点的编号 `bdEdgeIdxD` 以及边界边的编号 `bdEdgeIdx`, 这样约束点的编号为

```
1 bdNodeIdx = bdStruct.bdNodeIdx;
2 bdEdgeD = bdStruct.bdEdgeD;
3 g_D = pde.g_D; Dw = pde.Du;
4 id = [bdNodeIdx; bdEdgeIdx+N; bdEdgeIdx+N+NE];
```

由此给出自由点编号

```
1 isBdNode = false(NN dof, 1); isBdNode(id) = true;
2 bdDof = (isBdNode); freeDof = (~isBdNode);
```

3. 边界顶点值和边界边中点值为

```
1 % vertices on the boundary
2 nodeD = node(bdNodeIdx,:); wD = g_D(nodeD);
3 % mid-edge on the boundary
4 z1 = node(bdEdgeD(:,1,:)); z2 = node(bdEdgeD(:,2,:)); zc = (z1+z2)./2;
5 wDc = g_D(zc);
```

边界边法向矩量如下计算

```
1 % moments on the boundary
2 eb = z1-z2; % e = z2-z1
3 neb = [-eb(:,2), eb(:,1)]; % scaled ne
4 wnD = sum(1/6*(Dw(z1)+4*Dw(zc)+Dw(z2)).*neb, 2);
```

这里 Dw 是梯度 ∇w , 而 $\partial_n w = \nabla w \cdot \vec{n}$ 中的 \vec{n} 通过边界边的旋转获得 (并进行单位化), 且用 Simpson 公式近似积分.

4. 这样, 我们如下求解

```
1 w = zeros(NN dof, 1); w(bdDof) = [wD; wDc; wnD];
2 ff = ff - kk*w;
3 w(freeDof) = kk(freeDof, freeDof)\ff(freeDof);
```

8.2.7 程序整理

主程序如下

```
1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [200, 400, 600, 800, 1000];
6 maxIt = length(nameV);
7 h = zeros(maxIt, 1); N = zeros(maxIt, 1);
8 ErrL2 = zeros(maxIt, 1);
9 ErrH1 = zeros(maxIt, 1);
10 ErrH2 = zeros(maxIt, 1);
```

```

11 ErrI = zeros(maxIt,1);
12
13 %% PDE data
14 pde = PlateBendingData();
15
16 %% Virtual element method
17 for k = 1:maxIt
18     % load mesh
19     load( ['meshdata', num2str(nameV(k)), '.mat'] );
20     % get boundary information
21     bdStruct = setboundary(node,elem);
22     % solve
23     [w,info] = PlateBending_COVEM(node,elem,pde,bdStruct);
24     % record and plot
25     N(k) = length(w); h(k) = 1/sqrt(size(elem,1));
26     if size(elem,1)<2e3
27         figure(1); clf;
28         subplot(1,3,1), showmesh(node,elem);
29         subplot(1,3,2), showsolution(node,elem,w(1:size(node,1)));
30         subplot(1,3,3), showsolution(node,elem,pde.ueexact(node));
31     end
32     % compute errors in discrete L2 and H1 norms
33     index = info.elem2dof; % elemenwise global index
34     chi = cellfun(@(id) w(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
35     kOrder = 2;
36     Ph = info.Ph;
37     ErrL2(k) = getL2error(node,elem,Ph,chi,pde,kOrder);
38     ErrH1(k) = getH1error(node,elem,Ph,chi,pde,kOrder);
39     ErrH2(k) = getH2error(node,elem,Ph,chi,pde,kOrder);
40     % compute errors in discrete energy norm
41     % auxstructure
42     auxT = auxstructure(node,elem);
43     edge = auxT.edge;
44     % chi1: evaluation at all vertices
45     ueexact = pde.ueexact;
46     chi1 = ueexact(node);
47     % chi2: evaluation at all mid-edge points
48     z1 = node(edge(:,1,:)); z2 = node(edge(:,2,:)); zc = (z1+z2)/2;
49     chi2 = ueexact(zc);
50     % chi3: moments of \partial_n v on edges with given orientation
51     e = z1-z2; % e = z2-z1
52     Ne = [-e(:,2),e(:,1)]; % scaled ne
53     Dw = pde.Du;
54     Dw1 = Dw(z1); Dwc = Dw(zc); Dw2 = Dw(z2);
55     wnD = sum(1/6*(Dw1+4*Dwc+Dw2).*Ne,2);
56     chi3 = wnD;
57     % chi
58     chi = w;
59     chie = [chi1; chi2; chi3];
60     % Relative error
61     kk = info.kk; freeDof = info.freeDof;
62     kk = kk(freeDof,freeDof); chie = chie(freeDof); chi = chi(freeDof);
63     ErrI(k) = sqrt(abs((chi-chie)'*kk*(chi-chie))/abs(chie'*kk*chie));
64 end
65
66 %% Plot convergence rates and display error table
67 figure(2);

```

```

68 showrateh(h,ErrL2,ErrH1,ErrH2,ErrI);
69
70 fprintf('\n');
71 disp('Table: Error')
72 colname = {'#Dof','h','||u-u_h||','|u-u_h|_1','|u-u_h|_2','||u_I-u_h||_A'};
73 disptable(colname,N,[],h,'%0.3e',ErrL2,'%0.5e',ErrH1,'%0.5e',ErrH2,'%0.5e',ErrI,'%0.5e');

```

函数文件见 GitHub.

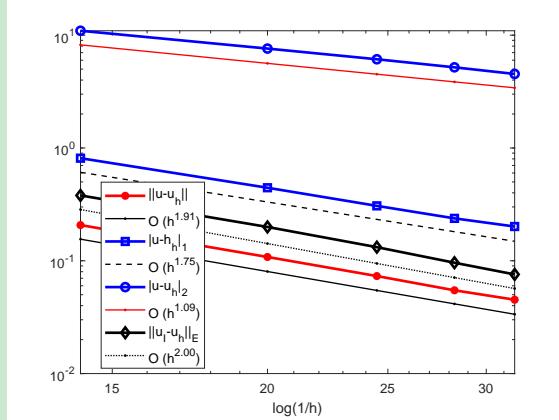


图 17. 板弯问题 C^0 虚拟元方法的误差阶

可以看出, H^2 和 L^2 误差都达到最优阶 (分别为 1, 2 阶); H^1 误差也近乎最优阶 2; 能量范数误差为 2 阶, 它有超收敛现象.

8.2.8 C^0 -强化处理

为了保证有内部自由度, 我们可引入提升空间

$$\tilde{V}_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-2}(K), v|_{\partial K} \in \mathbb{B}_k(\partial K)\},$$

其中, 边界元空间为

$$\mathbb{B}_k(\partial K) = \{v \in C^0(\partial K) : v|_e \in \mathbb{P}_k(e), \Delta v|_e \in \mathbb{P}_{k-2}(e), e \subset \partial K\}.$$

在该空间中可定义 L^2 投影 Π_{k-2}^0 . 注意到椭圆投影的计算用不到 $k-3$ 和 $k-2$ 阶的内部矩量, 为此我们引入强化空间

$$W_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-2}(K), \int_K \Pi^K v m dx = \int_K v m dx, m \in \mathbb{M}_{k-2}(K) \setminus \mathbb{M}_{k-4}(K), v|_{\partial K} \in \mathbb{B}_k(\partial K)\}.$$

该空间的自由度与 C^0 元的一致. 注意引入这个空间的目的就是为了计算 L^2 投影 Π_{k-2}^0 .

对最低阶 $k=2$, 显然有 $\Pi_{k-2}^0 = \Pi_0^0 = \Pi^K$. 椭圆投影的计算同 C^0 元. 右端采用 L^2 投影近似

$$\ell_h^K(v) = \int_K f \Pi_{k-2}^0 v dx.$$

单元载荷向量为

$$F_K = \int_K f \Pi_{k-2}^0 \phi dx = (\boldsymbol{\Pi}_{k-2,*}^0)^T \int_K f m^{k-2} dx,$$

式中, m^{k-2} 为直到 $k-2$ 阶的尺度单项式向量.

注 8.6 强化技巧的计算结果与前面的差不多, 1.75 可近似认为是二阶. 事实上, 如果使用单元个数为 100, 200, 300, 400, 500 的网格进行试验, 那么 H^1 的误差阶变为 1.87. 由于当 $k = 2$ 时, L^2 投影就是椭圆投影, 为此后面不需要再考虑这种强化技巧.

8.3 Morley 型非协调 VEM

前面已经提到, 从可计算的角度来说, 在分部积分公式

$$\begin{aligned} a^K(p, v) &= - \int_K Q_{3i,i}(p) v dx_1 dx_2 \\ &\quad + \int_{\partial K} (Q_{3n}(p) + \partial_t M_{tn}(p)) v ds - \int_{\partial K} M_{nn}(p) \partial_n v ds \\ &\quad + \sum_{i=1}^m [M_{tn}(p)](z_i) v(z_i) \\ &=: I_1 + I_2 + I_3 + I_4. \end{aligned}$$

中, 我们用不到 I_2 边界函数的 $k-2$ 阶矩量. 为此, 去掉相应的自由度, C^0 的编程过程稍加改动即可获得 Morley 型自由度的计算结果.

剩下的就是用自由度定义函数的延拓方式.

- 注意到 $\Pi^K v$ 的计算用不到 $k-2$ 阶矩量, 正因为如此, $\Pi^K v$ 的 $k-2$ 阶矩量 $\int_e m_{k-2}^e \Pi^K v ds$ 也无需 v 的 $k-2$ 阶矩量就可计算. 为此, 我们用 $\Pi^K v$ 的 $k-2$ 阶矩量替代 v 的 $k-2$ 阶矩量, 即定义

$$\int_e m_{k-2}^e v ds := \int_e m_{k-2}^e \Pi^K v ds.$$

也就是说, 对 $v \in V_k(K)$ ($k = 2$), 在展开式

$$v = \sum_{i=1}^{N_v} \chi_i(v) \phi_i + \chi_{N_v+i}(v) \phi_{N_v+i} + \chi_{2N_v+i}(v) \phi_{2N_v+i}$$

中, 我们将第二部分基函数对应的自由度替换为 $\chi_{N_v+i}(\Pi^K v)$, 即定义新函数

$$v^M = \sum_{i=1}^{N_v} \chi_i(v) \phi_i + \chi_{N_v+i}(\Pi^K v) \phi_{N_v+i} + \chi_{2N_v+i}(v) \phi_{2N_v+i},$$

这就是我们给出的延拓方式.

- 记 C^0 连续的虚拟元空间为 $\tilde{V}_k(K)$, 上面的延拓方式表明, 我们定义的局部 Morley 元空间为

$$V_k(K) = \left\{ v \in \tilde{V}_k(K) : \int_e m_{k-2}^e \Pi^K v ds = \int_e m_{k-2}^e v ds \right\}.$$

注意, 尽管 $V_k(K)$ 是 C^0 连续空间的子空间, 但这只是局部来说的, 整体上并不连续.

本文仅考虑最低阶, 即 $k = 2$ 的情形. 自由度包含顶点值泛函和边界的法向导数矩量

$$\chi_{n_e}(v) = \int_e \partial_n v ds,$$

如下图所示

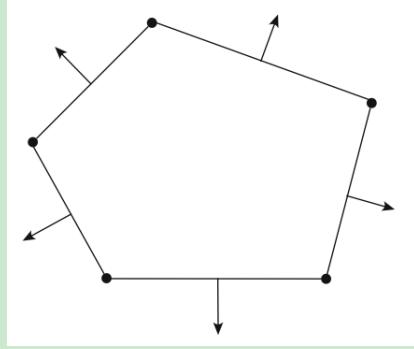


图 18. Morley 型虚拟元空间的局部自由度 ($k = 2$)

设单元顶点个数为 N_v , 则每个单元有 $N_k = 2N_v$ 个自由度, 排列为

- 顶点值泛函

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 边中点法向导数的矩量

$$\chi_{N_v+i}(v) = \int_{e_i} \partial_n v \, ds, \quad i = 1, \dots, N_v.$$

注 8.7 上面给出的自由度可以有不同的延拓方式.

后面重复一下 C^0 的过程.

8.3.1 过渡矩阵 D

设 $V_k(K)$ 为 C^0 连续的非协调虚拟元空间, 基函数排成行向量, 记为

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

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

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

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

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

根据自由度的定义,

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

且 D 是 $N_k \times N_p$ 的矩阵.

$\mathbb{P}_2(K)$ 的尺度单项式集合 m^T 由以下元素组成

$$m_1(x, y) = 1, \quad m_2(x, y) = \frac{x - x_K}{h_K}, \quad m_3(x, y) = \frac{y - y_K}{h_K},$$

$$m_4(x, y) = \frac{(x - x_K)^2}{h_K^2}, \quad m_5(x, y) = \frac{(x - x_K)(y - y_K)}{h_K^2}, \quad m_6(x, y) = \frac{(y - y_K)^2}{h_K^2},$$

从而 ∇m^T 为

$$\begin{aligned}\nabla m_1 &= [0, 0], \quad \nabla m_2 = \left[\frac{1}{h_K}, 0 \right], \quad \nabla m_3 = \left[0, \frac{1}{h_K} \right], \\ \nabla m_4 &= \left[\frac{2(x - x_K)}{h_K^2}, 0 \right], \quad \nabla m_5 = \left[\frac{y - y_K}{h_K^2}, \frac{x - x_K}{h_K^2} \right], \quad \nabla m_6(x, y) = \left[0, \frac{2(y - y_K)}{h_K^2} \right].\end{aligned}$$

现在来计算过渡矩阵. 顶点值和边中点值比较容易, 如下计算

```

1 % element information
2 index = elem{iel};      Nv = length(index);      Ndof = 2*Nv;
3 xK = centroid(iel,1);  yK = centroid(iel,2);  hK = diameter(iel);
4 x = node(index,1);    y = node(index,2);    % vertices
5 v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
6 Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors
7 he = sqrt(sum(Ne.^2,2));
8 ne = Ne./repmat(he,1,2); % outer normal vectors
9 te = [-ne(:,2), ne(:,1)];
10 % scaled monomials
11 m = @(x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK.^2, ...
12 (x-xK).*(y-yK)/hK.^2, (y-yK).^2/hK.^2]; % m1,...,m6
13 Gradm = @(x,y) [[0,0]; [1, 0]/hK; [0, 1]/hK; [2*(x-xK), 0]/hK.^2;
14 [(y-yK), (x-xK)]/hK.^2; [0, 2*(y-yK)]/hK.^2]; % grad m
15 % D
16 D1 = zeros(Ndof,Nm);
17 D1(1:Nv,:) = m(x,y); % vertices

```

再考虑边的法向矩量. 注意到 ∇m_α 次数不超过一次, 可用梯形公式计算, 即

$$\chi_{N_v+i}(m_\alpha) = \int_e \nabla m_\alpha \cdot \mathbf{n}_e ds = \frac{1}{2} (\nabla m_\alpha(z_i) + \nabla m_\alpha(z_{i+1})) \cdot \mathbf{n}_e |e|, \quad e = e_i.$$

这里, $\mathbf{n}_e |e|$ 恰是定向边的反向逆时针旋转 90 度所得. 该部分如下计算

```

1 for i = 1:Nv
2     gradi = 0.5*(Gradm(x(v1(i)),y(v1(i)))+Gradm(x(v2(i)),y(v2(i))));
3     D1(Nv+i,:) = Ne(i,:)*gradi';
4 end
5 D{iel} = D1;

```

8.3.2 椭圆投影的计算

椭圆投影 Π^K 满足

$$\begin{cases} a^K(\Pi^K \phi_i, m_\alpha) = a^K(\phi_i, m_\alpha) \\ P_0(\Pi^K \phi_i) = P_0(\phi_i) \quad , \quad i = 1, \dots, N_k, \quad \alpha = 1, \dots, N_p, \\ P_0(\nabla \Pi^K \phi_i) = P_0(\nabla \phi_i) \end{cases}$$

写为向量形式为

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

注意, 第二个限制条件实际上有两个限制, 即 x 和 y 方向的两个导数. 对 $p \in \mathbb{P}_2$, 有

$$a^K(p, v) = - \int_{\partial K} M_{nn}(p) \partial_n v ds + \sum_{i=1}^{N_v} [M_{tn}(p)](z_i) v(z_i),$$

而

$$P_0 v = \frac{1}{N_v} \sum_{i=1}^{N_v} v(z_i) \text{ 或 } P_0(v) = \int_{\partial K} v \, ds,$$

前者对应第一个限制条件, 后者对应第二个限制条件.

设 Π^K 在节点基下的矩阵记为 $\mathbf{\Pi}^K$, 即

$$\Pi^K(\phi_1, \phi_2, \dots, \phi_{N_k}) = (\phi_1, \phi_2, \dots, \phi_{N_k}) \mathbf{\Pi}^K \quad \text{或} \quad \Pi^K \phi^T = \phi^T \mathbf{\Pi}^K.$$

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

$$\mathbf{\Pi}^K = (\chi_i(\Pi^K \phi_j)).$$

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

$$\Pi^K \phi^T = m^T \mathbf{\Pi}_*^K.$$

两组不同基下的矩阵满足

$$\mathbf{\Pi}^K = D \mathbf{\Pi}_*^K.$$

令

$$G = a^K(m, m^T), \quad B = a^K(m, \phi^T),$$

则

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

其中, G 是 $N_p \times N_p$ 的方阵, B 是 $N_k \times N_p$ 的矩阵. 而 $P_0(m^T)$ 是一行向量, $P_0(\nabla m^T)$ 是两行向量. 显然 G 的前三行为零, 限制条件的三个行向量替换前三行后得

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

式中,

$$\tilde{G}(i,:) = \begin{cases} P_0(m^T), & i = 1 \\ P_0(\partial_x m^T), & i = 2 \\ P_0(\partial_y m^T), & i = 3 \\ G(i,:), & i \geq 4 \end{cases}, \quad \tilde{B} = \begin{cases} P_0(\phi^T), & i = 1 \\ P_0(\partial_x \phi^T), & i = 2 \\ P_0(\partial_y \phi^T), & i = 3 \\ B(i,:), & i \geq 4 \end{cases},$$

且

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

\tilde{G} 或 G 不需要直接计算, 有如下的一致性关系

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

现在来计算 B , 其元素为

$$\begin{aligned} B_{\alpha j} &= a^K(m_\alpha, \phi_j) = - \sum_{e \subset \partial K} \int_e M_{nn}(m_\alpha) \partial_n \phi_j \, ds + \sum_{i=1}^{N_v} [M_{tn}(m_\alpha)](z_i) \phi_j(z_i) \\ &=: J_1(\alpha, j) + J_2(\alpha, j), \end{aligned} \tag{8.62}$$

其中,

$$M_{nn}(p) = M_{ij}(p)n_i n_j, \quad M_{tn}(p) = M_{ij}(p)t_i n_j,$$

$$M_{ij}(p) = -D ((1-\nu)\partial_{ij}p + \nu(\partial_{11}p + \partial_{22}p)\delta_{ij}).$$

注意到

| | $m_\alpha (\alpha = 1, 2, 3)$ | m_4 | m_5 | m_6 |
|-----------------|-------------------------------|-----------|-----------|-----------|
| ∂_{11} | 0 | $2/h_K^2$ | 0 | 0 |
| ∂_{12} | 0 | 0 | $1/h_K^2$ | 0 |
| ∂_{22} | 0 | 0 | 0 | $2/h_K^2$ |

由此可得 $M_{ij}(m_\alpha)$, 进而获得 M_{nn}, M_{tn} , 如下计算.

```

1      % \partial_ij (m)
2      D11 = zeros(Nm,1); D11(4) = 2/hK^2;
3      D12 = zeros(Nm,1); D12(5) = 1/hK^2;
4      D22 = zeros(Nm,1); D22(6) = 2/hK^2;
5      % Mij(m)
6      M11 = -para.D*((1-para.nu)*D11 + para.nu*(D11+D22));
7      M12 = -para.D*(1-para.nu)*D12;
8      M22 = -para.D*((1-para.nu)*D22 + para.nu*(D11+D22));
9      % Mnn(m) on e1,...,eNv
10     n1 = ne(:,1); n2 = ne(:,2);
11     Mnn = M11*(n1.*n1)' + M12*(n1.*n2+n2.*n1)' + M22*(n2.*n2)';
12     % Mtn(m) on e1,...,eNv
13     t1 = te(:,1); t2 = te(:,2);
14     Mtn = M11*(t1.*n1)' + M12*(t1.*n2+t2.*n1)' + M22*(t2.*n2)';

```

这里, M_{nn} 和 M_{tn} 行对应 m_α , 列对应 e_j .

对 J_1 , 因 $M_{nn}(m_\alpha)$ 在每条边上为常数, 故

$$J_1(\alpha, j) = - \sum_{e \subset \partial K} \int_e M_{nn}(m_\alpha) \partial_n \phi_j ds = - \sum_{e \subset \partial K} M_{nn}(m_\alpha) \int_e \partial_n \phi_j ds. \quad (8.63)$$

对固定的 $e = e_j$, 由基函数的 Kronecker 性质,

$$\int_e \partial_n \phi^T ds = [\mathbf{0}, \mathbf{e}_j],$$

其中, $\mathbf{0}$ 是 N_v 长度的全零行向量, \mathbf{e}_j 是 N_v 长度的自然基向量. 显然用 M_{nn} 的第 j 列乘以上面的行向量, 所得为 J_1 在边 e_j 上的结果, 循环求和即得 J_1 .

对 J_2 , 即

$$J_2(\alpha, j) = \sum_{i=1}^{N_v} [M_{tn}(m_\alpha)](z_i) \phi_j(z_i),$$

由跳量定义,

$$[M_{tn}(m_\alpha)](z_i) = M_{tn}(m_\alpha) \Big|_{z_i^-}^{z_i^+} = M_{tn}(m_\alpha)(z_i^+) - M_{tn}(m_\alpha)(z_i^-),$$

这里, $M_{tn}(m_\alpha)(z_i^+)$ 是 z_i 右侧边 e_i 上的结果, $M_{tn}(m_\alpha)(z_i^-)$ 是左侧边 e_{i-1} 上的结果. 而

$$\phi^T(z_i) = [\mathbf{e}_i, \mathbf{0}].$$

综上, B 可如下计算

```

1      B1 = zeros(Nm,Ndof);
2      p1 = [Nv,1:Nv-1]; p2 = 1:Nv;
3      for j = 1:Nv % loop of edges
4          % nphi on ej
5          nphi = zeros(1,Ndof); nphi(2*Nv+j) = 1;
6          % Jump at zj
7          Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
8          % phi at zj
9          phi = zeros(1,Ndof); phi(j) = 1;
10         % B1
11         B1 = B1 - Mnn(:,j)*nphi + Jump*phi;
12     end

```

再计算 B_s . 它的第一行容易获得, 显然为

$$\tilde{B}(1,:) = P_0(\phi^T) = \frac{1}{N_v}[\mathbf{1}, \mathbf{0}]. \quad (8.64)$$

对剩下的限制条件, 注意到

$$\begin{aligned} \int_{\partial K} \nabla v \, ds &= \sum_{e \subset \partial K} \int_e \partial_{n_e} v n_e \, ds + \int_e \partial_{t_e} v t_e \, ds \\ &= \sum_{e \subset \partial K} n_e \int_e \partial_n v \, ds + \sum_{i=1}^{N_v} t_{e_i} (v(z_{i+1}) - v(z_i)), \end{aligned} \quad (8.65)$$

我们有

$$\begin{cases} \int_{\partial K} \nabla \phi_j \, ds = t_{e_{j-1}} - t_{e_j}, \\ \int_{\partial K} \nabla \phi_{N_v+j} \, ds = n_{e_j}. \end{cases}$$

```

1      % Bs
2      B1s = B1;
3      % first constraint
4      B1s(1,1:Nv) = 1/Nv;
5      % second constraint
6      B1s(2:3,1:Nv) = te([Nv,1:Nv-1],:) - te';
7      B1s(2:3,Nv+1:end) = ne';
8      Bs{iel} = B1s;

```

有了 B, B_s, D , 我们就可用一致性关系计算 G, G_s .

```

1      % G, Gs
2      G{iel} = B1*D1;       Gs{iel} = B1s*D1;

```

8.3.3 单元刚度矩阵与载荷向量的计算

令

$$A_K^1(i,j) = a^K(\Pi^K \phi_j, \Pi^K \phi_i),$$

$$\begin{aligned} A_K^2(i,j) &= S^K(\phi_j - \Pi^K \phi_j, \phi_i - \Pi^K \phi_i) \\ &= h_K^{-2} \sum_{r=1}^{N_k} \chi_r(\phi_j - \Pi^K \phi_j) \chi_r(\phi_i - \Pi^K \phi_i), \end{aligned}$$

则

$$A_K^1 = a^K(\phi, \phi^T) = (\boldsymbol{\Pi}_*^K)^T a^K(m, m^T) \boldsymbol{\Pi}_*^K = (\boldsymbol{\Pi}_*^K)^T G \boldsymbol{\Pi}_*^K,$$

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

单元刚度矩阵如下计算

```

1 Aelem = cell(NT,1);
2 % Projection
3 Pis = Gs{iel}\Bs{iel}; Pi = D{iel}*Pis; I = eye(size(Pi));
4 % Stiffness matrix
5 AK = Pis'*G{iel}*Pis + hK^(-2)*(I-Pi)'*(I-Pi);
6 Aelem{iel} = reshape(AK',1,[]); % straighten as row vector for easy assembly

```

现在考虑单元载荷向量. 右端的

$$F_h^K(v) = \int_K \Pi_{k-2}^0 f v dx,$$

当 $k = 2$ 时, $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (5.14) 定义, 即单元顶点值的平均. 注意到

$$F_h^K(v) = \int_K \Pi_0^0 f v dx = \int_K \Pi_0^0 f \Pi_0^0 v dx,$$

我们有单元载荷向量为

$$F_K = \int_K P_0 f P_0 \phi dx = f(x_K, y_K) |K| P_0 \phi,$$

这里为了方便, $P_0 f$ 用中点值近似, 而 $P_0 \phi$ 由 (8.64) 给出.

```

1 PO = zeros(Ndof,1); PO(1:Nv) = 1/Nv;
2 fK = pde.f(cK)*areaK*PO;

```

8.3.4 符号刚度矩阵和载荷向量

局部自由度的后 N_v 个为边法向导数的矩量, 它是有方向的. 我们要取定一个方向, 从而相应修改刚度矩阵元素的符号. 注意到

$$a^K(\pm \varphi_i, \pm \varphi_j) = \pm \cdot \pm a^K(\varphi_i, \varphi_j),$$

为此可对应给出一个符号矩阵, 它记录 (i, j) 位置的符号.

1. 边的定向: 对单元的边 e , 规定逆时针方向为其在单元中的正向, 可用局部编号来确定.
2. 边的符号:
 - 对内部定向边, 称终点与起点整体编号差的符号为该边在单元中的符号.
 - 对边界边, 规定符号为正 (即 +1).
 - 显然对内部边, 限制在相邻两个单元上的符号相异, 因而可作为法向的符号.

下面说明如何获得固定单元边的符号, 用 `sgnElem` 存储.

- 执行如下语句可获得第 `iel` 个单元所有边的符号

```

1 index = elem{iel}; Nv = length(index); Ndof = 2*Nv;
2 v1 = 1:Nv; v2 = [2:Nv, 1]; % edge index
3 sgnElem = sign(index(v2)-index(v1));

```

这里边界边的符号可能不正确.

- 利用逻辑运算可修改 `sgnelem` 中边界边的符号差 (边界边符号差为 1).

```
1     id = elem2edge{iel}; sgnbd = E(id); sgnelem(sgnbd) = 1;
2     sgnbase = ones(Ndof,1); sgnbase(2*Nv+1:end) = sgnelem;
3     sgnK = sgnbase*sgnbase'; sgnF = sgnbase;
```

这里, `bdIndex` 是边界边的自然序号, `E` 内部边对应 `false`, 边界边对应 `true`.

上面的过程在单元刚度矩阵循环计算的过程中同步处理, 不用存储为元胞数组. 最终的单元刚度矩阵和载荷向量为

```
1     AK = AK.*sgnK; fK = fK.*sgnF;
```

8.3.5 边界条件的处理

我们考虑的是强加边界条件

$$w = \partial_n w = 0, \quad (x, y) \in \partial\Omega,$$

这里可以是非齐次的 (这两个条件都是 Dirichlet 边界条件).

$w|_{\partial\Omega} = g$ 与常规情形一样处理. 现考虑 $\partial_n w|_{\partial\Omega} = g_n$.

- 第三种自由度为

$$\chi(v) = \int_e \partial_n v ds,$$

因 $\partial_n w$ 在边界线段 e 上的值已知, 故可用数值积分计算边界上的该类自由度.

- 在 `setboundary.m` 中我们给出了 Dirichlet 点的编号 `eD` 以及边界边的编号 `bdIndex`, 这样约束点的编号为

```
1 bdNodeIdx = bdStruct.bdNodeIdx;
2 bdEdgeD = bdStruct.bdEdgeD;
3 id = [bdNodeIdx; bdEdgeIdx+N];
```

由此给出自由点编号

```
1 isBdNode = false(NNdoF,1); isBdNode(id) = true;
2 bdDof = (isBdNode); freeDof = (~isBdNode);
```

- 边界顶点值为

```
1 % vertices on the boundary
2 nodeD = node(bdNodeIdx,:); wD = g_D(nodeD);
```

边界边法向矩量如下计算

```
1 % moments on the boundary
2 z1 = node(bdEdgeD(:,1),:); z2 = node(bdEdgeD(:,2),:); zc = (z1+z2)./2;
3 eb = z1-z2; % e = z2-z1
4 neb = [-eb(:,2),eb(:,1)]; %scaled ne
5 wnD = sum(1/6*(Dw(z1)+4*Dw(zc)+Dw(z2)).*neb,2);
```

这里 ∇w 是梯度 ∇w , 而 $\partial_n w = \nabla w \cdot \vec{n}$ 中的 \vec{n} 通过边界边的旋转获得(并进行单位化), 且用 Simpson 公式近似积分.

4. 这样, 我们如下求解

```

1 w = zeros(NNdof,1); w(bdDof) = [wD; wnD];
2 ff = ff - kk*w;
3 w(freeDof) = kk(freeDof,freeDof)\ff(freeDof);

```

8.3.6 程序整理

主程序如下

```

1 clc; close all;
2 clear variables;
3
4 %% Parameters
5 nameV = [200, 400, 600, 800, 1000];
6 maxIt = length(nameV);
7 h = zeros(maxIt,1); N = zeros(maxIt,1);
8 ErrL2 = zeros(maxIt,1);
9 ErrH1 = zeros(maxIt,1);
10 ErrH2 = zeros(maxIt,1);
11
12 %% PDE data
13 pde = PlateBendingData();
14
15 %% Virtual element method
16 for k = 1:maxIt
17     % load mesh
18     load(['meshdata', num2str(nameV(k)), '.mat']);
19     % get boundary information
20     bdStruct = setboundary(node, elem);
21     % solve
22     [w, info] = PlateBending_MorleyVEM(node, elem, pde, bdStruct);
23     % record and plot
24     N(k) = length(w); h(k) = 1/sqrt(size(elem,1));
25     if size(elem,1)<2e3
26         figure(1); clf;
27         subplot(1,3,1), showmesh(node, elem);
28         subplot(1,3,2), showsolution(node, elem, w(1:size(node,1)));
29         subplot(1,3,3), showsolution(node, elem, pde.uexact(node));
30     end
31     % compute errors in discrete L2 and H1 norms
32     index = info.elem2dof; % elemenwise global index
33     chi = cellfun(@(id) w(id), index, 'UniformOutput', false); % elementwise ...
            numerical d.o.f.s
34     kOrder = 2;
35     Ph = info.Ph;
36     ErrL2(k) = getL2error(node, elem, Ph, chi, pde, kOrder);
37     ErrH1(k) = getH1error(node, elem, Ph, chi, pde, kOrder);
38     ErrH2(k) = getH2error(node, elem, Ph, chi, pde, kOrder);
39 end
40
41 %% Plot convergence rates and display error table
42 figure(2);
43 showrateh(h, ErrL2, ErrH1, ErrH2);

```

```

44
45 fprintf ('\n');
46 disp('Table: Error')
47 colname = {'#Dof','h','||u-u_h||','|u-u_h|_1','|u-u_h|_2'};
48 disptable(colname,N,[],h,'%0.3e',ErrL2,'%0.5e',ErrH1,'%0.5e',ErrH2,'%0.5e');

```

函数文件见 GitHub.

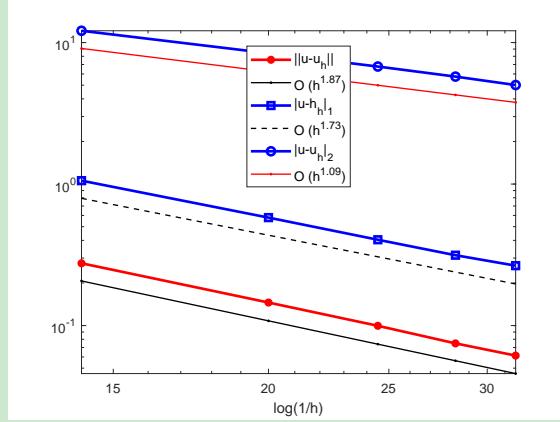


图 19. 板弯问题 Morley 型虚拟元方法的误差阶

可以看出, H^2 , H^1 和 L^2 误差都达到最优阶 (分别为 1, 2, 2 阶).

8.4 C^1 协调 VEM

8.4.1 虚拟元空间

1. C^1 连续性仍然取决于边界, 单元的内部连续性是由延拓导致的.
2. 先考虑边 e 内部, 这里假设边 e 已定向.

- 对 C^0 连续, 我们要求 $v|_e \in \mathbb{P}_k(e)$. 设

$$v|_e(s) = c_0 m_0(s) + \cdots + c_k m_k(s), \quad m_i(s) \in \mathbb{M}_k(e), \quad (8.66)$$

从而给出边上函数直到 k 阶的矩量

$$\chi_e(v) = |e|^{-1} (m_e, v)_e, \quad m_e \in \mathbb{M}_{k-2}(e).$$

- 对 C^1 连续, 我们要求 $\nabla v|_e \in \mathbb{P}_k(e)^2$. 显然有

$$\nabla v|_e = (\nabla v \cdot \boldsymbol{\tau}) \boldsymbol{\tau} + (\nabla v \cdot \mathbf{n}) \mathbf{n} = \partial_{\boldsymbol{\tau}} v \boldsymbol{\tau} + \partial_{\mathbf{n}} v \mathbf{n},$$

式中, $\boldsymbol{\tau}$ 和 \mathbf{n} 分别为定向边 e 对应的切向量和法向量. 由于 $v|_e \in \mathbb{P}_k(e)$ 自然蕴含 $\partial_{\boldsymbol{\tau}} v|_e \in \mathbb{P}_{k-1}(e)$, 为此只需要 $\partial_{\mathbf{n}} v|_e \in \mathbb{P}_{k-1}(e)$. 设

$$\partial_{\mathbf{n}} v|_e(s) = d_0 m_0(s) + \cdots + d_{k-1} m_{k-1}(s), \quad m_i(s) \in \mathbb{M}_{k-1}(e), \quad (8.67)$$

则有边上法向导数直到 $k-1$ 阶的矩量

$$\chi_{n_e}(v) = (m_e, \partial_{\mathbf{n}} v)_e, \quad m_e \in \mathbb{M}_{k-1}(e).$$

3. 再考虑顶点的 C^0 连续性, 这要求给定 e 两个顶点处的值. 顶点值自由度无法用来求解 (8.67), 只能用来求解 (8.66), 从而边上函数的矩量要去掉 2 个.
4. 最后考虑顶点的 C^1 连续性, 这要求给定 e 两个顶点处的 $\partial_x v, \partial_y v$ 的值. 偏导数值既可用来求解 (8.66) (对它求切向导数即知), 也可用来求解 (8.67).

 - 因为 $\partial_\tau v = \nabla v \cdot \tau$, 所以两个点处的偏导数值提供了切向导数的两个值, 从而边上矩量还要再去掉两个.
 - 同理, 由 $\partial_n v = \nabla v \cdot n$, 偏导数值提供两个法向导数值, 从而边上法向矩量也要去掉两个.

5. 综上, C^1 连续性给出单元 K 的如下自由度

- 顶点处的函数值,

$$\chi_a(v) = v(z_i), \quad i = 1, \dots, N_v;$$

- 顶点处的导数值,

$$\chi_{a1}(v) = \partial_x v(z_i), \quad \chi_{a2}(v) = \partial_y v(z_i), \quad i = 1, \dots, N_v;$$

- 边上函数直到 $k - 4$ 阶的矩量,

$$\chi_e(v) = |e|^{-1} (m_e, v)_e, \quad m_e \in \mathbb{M}_{k-4}(e);$$

- 边上法向导数直到 $k - 3$ 阶的矩量,

$$\chi_{n_e}(v) = (m_e, \partial_n v)_e, \quad m_e \in \mathbb{M}_{k-3}(e).$$

6. 可计算性

- 考察分部积分公式 (8.51). 对 I_1 , 由 $\partial_i Q_{3i}(p) \in \mathbb{P}_{k-4}(K)$, 我们给出内部自由度

$$\chi_K(v) = |K|^{-1} (m_K, v)_K, \quad m_K \in \mathbb{M}_{k-4}(K).$$

- 对 $I_2, v|_e$ 能够显式计算出来.
- 对 $I_3, \partial_n v|_e$ 能够显式计算出来.
- 对 I_4 , 顶点值是自由度.

7. 唯一可解性与虚拟元空间的构造

- 由自由度为零, 立即得到 $v|_{\partial K} = \partial_n v|_{\partial K} = 0$.
- 注意到

$$\int_K |\Delta v|^2 dx = \int_K v \Delta^2 v dx + \int_{\partial K} \partial_n v \Delta v ds - \int_{\partial K} v \partial_n (\Delta v) ds,$$

再要求

$$\Delta^2 v \in \mathbb{P}_{k-4}(K),$$

则

$$\int_K |\Delta v|^2 dx = 0 \quad \Rightarrow \quad \Delta v = 0.$$

这样, 我们有边值问题

$$\begin{cases} \Delta v = 0, & x \in K, \\ v|_{\partial K} = 0, \end{cases}$$

由此可知 $v \equiv 0$.

- 上面的分析表明, 虚拟元空间可取为 ($k \geq 3$)

$$V_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K), v|_e \in \mathbb{P}_k(e), \partial_n v|_e \in \mathbb{P}_{k-1}(e), e \subset \partial K\},$$

且自由度为自由度包括

- $\chi_1(v)$: 顶点处的函数值,

$$\chi_a(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- $\chi_2(v)$: 顶点处的导数值,

$$\chi_{a1}(v) = \partial_x v(z_i), \quad \chi_{a2}(v) = \partial_y v(z_i), \quad i = 1, \dots, N_v.$$

- $\chi_3(v)$: 边上 v 的直到 $k-4$ 阶的矩量,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-4}(e).$$

- $\chi_4(v)$: 边上 $\partial_n v$ 的直到 $k-3$ 阶的矩量,

$$\chi_{ne}(v) = (m_e, \partial_n v)_e, \quad m_e \in \mathbb{M}_{k-3}(e).$$

- $\chi_5(v)$: 单元 K 上直到 $k-4$ 阶的矩量,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-4}(K).$$

注 8.8 对最低次, 即 $k=2$ 情形, 我们要做一定修改.

(a) C^1 连续性要求在边 e 的每个顶点给定 3 个自由度, 即顶点值和两个偏导数值.

(b) 2 个顶点值和 4 个偏导数值, 可提供 2 个顶点值和 2 个切向导数值, 它们确定 $v|_e \in \mathbb{P}_3(e)$.

4 个偏导数值可提供 2 个法向导数值, 它们确定 $\partial_n v|_e \in \mathbb{P}_1(e)$.

(c) 上面的分析表明,

$$V_2(K) = \{v \in H^2(K) : \Delta^2 v = 0, v|_e \in \mathbb{P}_3(e), \partial_n v|_e \in \mathbb{P}_1(e), e \subset \partial K\},$$

且自由度为

- 顶点处的函数值,

$$\chi_a(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 顶点处的偏导数值,

$$\chi_{a1}(v) = \partial_x v(z_i), \quad \chi_{a2}(v) = \partial_y v(z_i), \quad i = 1, \dots, N_v.$$

注 8.9 以下编程中只考虑 $k = 2$ 的情形, 且自由度排列顺序为

$$\begin{aligned}\chi_i(v) &= v(z_i), \quad i = 1, \dots, N_v, \\ \chi_{N_v+i}(v) &= h_\xi \partial_x v(z_i), \quad i = 1, \dots, N_v, \\ \chi_{2N_v+i}(v) &= h_\xi \partial_y v(z_i), \quad i = 1, \dots, N_v.\end{aligned}$$

通过 Scaling 可知, 导数自由度是有尺度的, 这里 h_ξ 是特征尺度, 如取所有连接该顶点的单元直径的平均. 注意, 不能直接取 $h_\xi = h_K$, 因为要保证自由度一致.

注 8.10 当 $K = T$ 为三角形时, 前面已经知道, 最低阶 C^1 协调元的自由度有 9 个. 而有限元方法的自由度则要多出很多. 根据前面 C^1 连续的讨论, 单元的每个顶点最起码要有 3 个自由度. 因有限元单元上整体是多项式, 且边上 6 个自由度决定 5 次多项式, 所以最低次的 C^1 协调有限元单元上至少为 5 次多项式, 从而有 21 个自由度, 即所谓的 Argyris 元. 造成这种差别的根本原因在于, 虚拟元方法是在边界流形上考虑的.

注 8.11 有意思的是, 对 $k = 2$ 的情形, C^0 元和 C^1 元的自由度个数都为 $3N_v$; 而对 $k \geq 3$, 不考虑内部矩量时, C^0 元的自由度个数为 $(2k - 1)N_v$, C^1 元的自由度个数为 $(2k - 2)N_v$. 这样看来, C^1 元的自由度个数反而少一点.

注 8.12 当 $k = 3$ 时, C^1 元的自由度还要添上边中点处的法向导数矩量. 此时的自由度与文献 [3] 中的 $\tilde{W}(T)$ 是一致的, 而且那里的边界上也是 3 次多项式. 有意思的是, 该文中给出的改进 Morley 元空间 $W(T)$ 对应前面的 C^0 元, 而 $\tilde{W}(T)$ 构造的初衷就是为了减少自由度. 这与注 8.11 的观察是一致的. 文献 [3] 中有限元的构造思路是用 bubble 函数分离边界与内部, 这恰好符合虚拟元的特点.

注 8.13 三角形上的 Zienkiewicz 元的自由度与这里的一致, 但它不是 C^1 连续的. 这表明, 相同自由度可匹配不同连续性的空间.

8.4.2 计算说明

先考虑过渡矩阵, 这里要注意导数自由度的特征尺度. 在辅助数据结构中, 我们给出了 `node2elem`, 它给出每个顶点周围的单元序号, 由此可获得每个顶点的特征尺度.

```
1 % characteristic length
2 hxi = cellfun(@(id) mean(diameter(id)), node2elem);
3 index = elem{iel};
4 hxiK = hxi(index);
```

过渡矩阵如下获得

```
1 % m'
2 m = @(x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK.^2, ...
3 (x-xK).*(y-yK)/hK.^2, (y-yK).^2/hK.^2]; % m1, ..., m6
4 % Dx(m'), Dy(m')
5 Dxm = @(x,y) [0*x, 1/hK+0*x, 0*x, 2*(x-xK)/hK.^2, (y-yK)/hK.^2, 0*x];
6 Dym = @(x,y) [0*x, 0*x, 1/hK+0*x, 0*x, (x-xK)/hK.^2, 2*(y-yK)/hK.^2];
7 % D
8 D1 = zeros(Ndof,Nm);
9 D1(1:Nv,:) = m(x,y);
10 D1(Nv+1:2*Nv,:) = repmat(hxiK,1,Nm).*Dxm(x,y);
11 D1(2*Nv+1:end,:) = repmat(hxiK,1,Nm).*Dym(x,y);
```

再考虑椭圆投影的计算. 注意到式 (8.63) 中的 $\partial_n \phi_j \in \mathbb{P}_1$, 所以可用梯形公式计算积分, 如下

$$-\sum_{e \subset \partial K} M_{nn}(m_\alpha) \int_e \partial_n \phi_j ds = -\sum_{i=1}^{N_v} M_{nn}(m_\alpha)|_{e_i} \frac{|e_i|}{2} (\partial_n \phi_j(z_i) + \partial_n \phi_j(z_{i+1})). \quad (8.68)$$

注意到

$$\chi_{a1,i}(v) = h_\xi \partial_x v(z_i) \Rightarrow \partial_x v(z_i) = \frac{1}{h_\xi} \chi_{a1,i}(v), \quad i = 1, \dots, N_v,$$

从而基函数的偏导数如下

```

1 % Dx(phi'), Dy(phi') at z1, ..., zNv (each row)
2 Dxphi = zeros(Nv, Ndof); Dyphi = zeros(Nv, Ndof);
3 Dxphi(:, Nv+1:2*Nv) = eye(Nv) ./ repmat(hxiK, 1, Nv);
4 Dyphi(:, 2*Nv+1:end) = eye(Nv) ./ repmat(hxiK, 1, Nv);

```

这样, B 可如下计算

```

1 % B
2 B1 = zeros(Nm, Ndof);
3 p1 = [Nv, 1:Nv-1]; p2 = 1:Nv;
4 for j = 1:Nv % loop of edges
    % int[\partial_n \phi_i]
    Dnphi1 = Dxphi(v1(j), :)*n1(j) + Dyphi(v1(j), :)*n2(j); % zj
    Dnphi2 = Dxphi(v2(j), :)*n1(j) + Dyphi(v2(j), :)*n2(j); % z_{j+1}
    nphi = 0.5*he(j)*(Dnphi1+Dnphi2);
    % Jump(m) at zj
    Jump = Mtn(:, p2(j))-Mtn(:, p1(j));
    % phi' at zj
    phi = zeros(1, Ndof); phi(j) = 1;
    % B1 on e and at zj
    B1 = B1 - Mnn(:, j)*nphi + Jump*phi;
5 end

```

B_s 的第一个限制条件同前处理, 即

$$\tilde{B}(1, :) = P_0(\phi^T) = \frac{1}{N_v} [\mathbf{1}, \mathbf{0}, \mathbf{0}].$$

对第二个限制条件, 仍使用

$$\begin{aligned} \int_{\partial K} \nabla v ds &= \sum_{e \subset \partial K} \int_e \partial_{n_e} v n_e ds + \int_e \partial_{t_e} v t_e ds \\ &= \sum_{e \subset \partial K} n_e \int_e \partial_n v ds + \sum_{i=1}^{N_v} t_{e_i} (v(z_{i+1}) - v(z_i)), \end{aligned}$$

我们有

$$\int_{\partial K} \nabla \phi_j ds = t_{e_{j-1}} - t_{e_j},$$

其他分量类似 (8.68) 计算.

```

1 B1s = B1;
2 % first constraint
3 B1s(1, 1:Nv) = 1/Nv;
4 % second constraint
5 for j = 1:Nv % loop of edges
    B1s(2:3, 1:Nv) = te([Nv, 1:Nv-1], :)' - te';
    Dnphi1 = Dxphi(v1(j), Nv+1:end)*n1(j) + Dyphi(v1(j), Nv+1:end)*n2(j); % zj

```

```

8      Dnphi2 = Dxphi(v2(j),Nv+1:end)*n1(j) + Dyphi(v2(j),Nv+1:end)*n2(j); % z_{j+1}
9      Nphi = 0.5*Ne(j,:)'*(Dnphi1+Dnphi2); % scaled
10     B1s(2:3,Nv+1:end) = B1s(2:3,Nv+1:end) + Nphi;
11 end

```

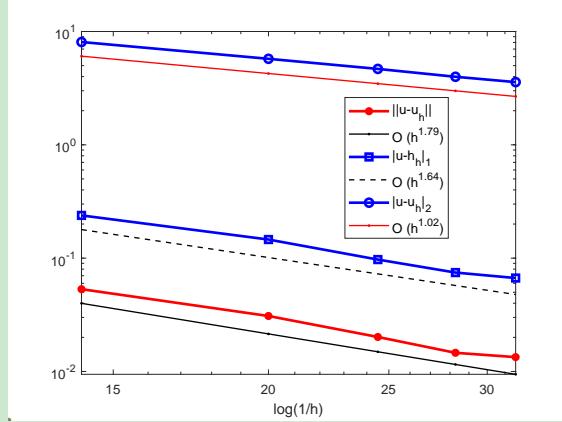


图 20. 板弯问题 C^1 -元的误差阶

相比 C^0 和 Morley 元, C^1 元的低模误差阶偏离 2 要多一些. 如果采用强化技巧, 对 $k = 2$ 即相当于用椭圆投影逼近, 此时的结果稍微好一点, 如下图

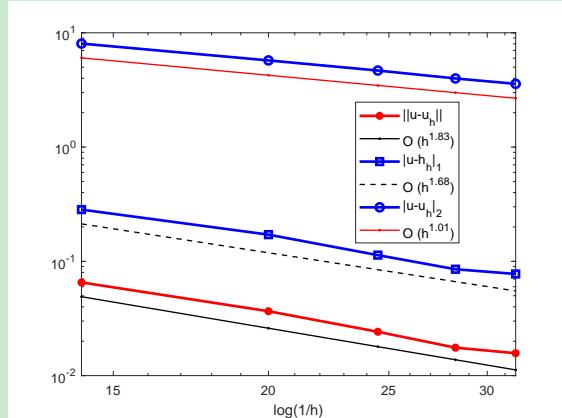


图 21. 板弯问题 C^1 -元的误差阶 (强化技巧)

注 8.14 程序中, 稳定项 S^K 中的尺度不再取为 h_K^{-2} , 而是采用特征尺度. 计算发现, 对前者, 若仍使用 C^0 元那里的简单近似, 则低模误差会丢失半阶, 而采用特征尺度, 结果与前面图中的类似. 若使用 100-500 的网格, H^1 和 L^2 误差阶分别为 1.84 和 1.70. 若用标准三角剖分, H^1 和 L^2 误差阶都达到最优.

Part III

奇异摄动问题的虚拟元方法

9 四阶奇异摄动问题的 C^0 元方法

9.1 一些说明

9.1.1 数学模型

考虑问题

$$\begin{cases} \varepsilon^2 \Delta^2 u - \Delta u = f & \text{in } \Omega, \\ u = \partial_n u = 0 & \text{on } \partial\Omega, \end{cases}$$

为了使用板弯问题的程序, 方程改写为

$$\begin{cases} -\varepsilon^2 \partial_{ij} M_{ij}(w) - \Delta w = f & \text{in } \Omega, \\ w = \partial_n w = 0 & \text{on } \partial\Omega, \end{cases}$$

其中,

$$M_{ij} = D((1-\nu)K_{ij} + \nu K_{kk}\delta_{ij}), \quad K_{ij} = -\partial_{ij}w.$$

注意, 当 D 和 ν 为常数时,

$$-\partial_{ij} M_{ij}(w) = D\Delta^2 w.$$

连续变分问题为: 找 $w \in V := H_0^2(\Omega)$ 使得

$$a(w, v) = \ell(v), \quad v \in V,$$

式中,

$$\begin{aligned} a(w, v) &= \varepsilon^2 \int_{\Omega} M_{ij}(w) K_{ij}(v) dx dy + \int_{\Omega} \nabla w \cdot \nabla v dx dy, \\ \ell(v) &= \int_{\Omega} f v dx dy. \end{aligned}$$

为了方便, 以下记

$$a_{\Delta}(w, v) = \int_{\Omega} M_{ij}(w) K_{ij}(v) dx dy, \quad a_{\nabla}(w, v) = \int_{\Omega} \nabla w \cdot \nabla v dx dy,$$

则

$$a(w, v) = \varepsilon^2 a_{\Delta}(w, v) + a_{\nabla}(w, v).$$

注意, 当 $D = 1$ 且 $\nu = 0$ 时,

$$a_{\Delta}(w, v) = (\nabla^2 w, \nabla^2 v).$$

为了方便, 以下称 a_{Δ} 为板弯项, 而 a_{∇} 为 Poisson 项.

9.1.2 分部积分公式

对板弯项, 有

$$\begin{aligned} a_{\Delta}^K(p, v) &= - \int_K Q_{3i,i}(p) v dx_1 dx_2 \\ &\quad + \int_{\partial K} (Q_{3n}(p) + \partial_t M_{tn}(p)) v ds - \int_{\partial K} M_{nn}(p) \partial_n v ds \\ &\quad + \sum_{i=1}^{N_v} [M_{tn}(p)](z_i) v(z_i), \end{aligned}$$

式中,

$$Q_{3i} = M_{ij,j}, \quad Q_{3n} = Q_{3i}n_i, \quad M_{nn} = M_{ij}n_i n_j, \quad M_{tn} = M_{ij}t_i n_j.$$

对 Poisson 项, 有

$$a_{\nabla}^K(p, v) = - \int_K \Delta p v dx + \int_{\partial K} \partial_n p v ds.$$

9.2 C^0 连续的非协调虚拟元空间

当 ε 很小时, 方程趋向于 Poisson 方程. 为此, 需要有限元具有 C^0 连续性以保证 Poisson 方程可以求好. 板弯问题 C^0 连续的虚拟元空间为

$$V_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K), \quad v|_e \in \mathbb{P}_k(e), \quad \Delta v|_e \in \mathbb{P}_{k-2}(e), \quad e \subset \partial K\},$$

规定: $\mathbb{P}_{-1} = \mathbb{P}_{-2} = \{0\}$. 该空间对应的自由度可取为

- 顶点值泛函

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 边上函数直到 $k-2$ 阶的矩量

$$\chi_e(v) = \frac{1}{|e|} \int_e m v ds, \quad m \in \mathbb{M}_{k-2}(e).$$

- 边上法向导数直到 $k-2$ 阶的矩量

$$\chi_{n_e}(v) = \int_e m \partial_n v ds, \quad m \in \mathbb{M}_{k-2}(e).$$

- 单元上函数直到 $k-4$ 阶的矩量

$$\chi_K(v) = \frac{1}{|K|} \int_K m v dx, \quad v \in \mathbb{M}_{k-4}(K).$$

注意到线弹性问题单元上函数的矩量直到 $k-2$ 阶, 为此引入提升空间

$$\tilde{V}_k(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-2}(K), \quad v|_e \in \mathbb{P}_k(e), \quad \Delta v|_e \in \mathbb{P}_{k-2}(e), \quad e \subset \partial K\},$$

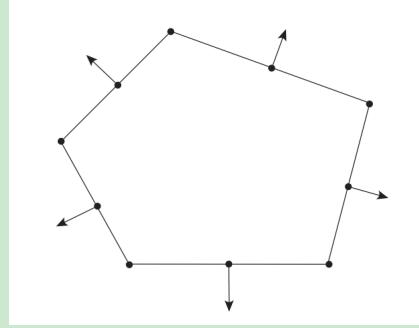
它的内部自由度是单元上函数直到 $k-2$ 阶的矩量, 其他都不变. 需要注意的是, $\tilde{V}_k(K)$ 上定义的椭圆投影用不到 $k-3$ 和 $k-2$ 阶的内部矩量. 类似 Morley 元的构造, 设 $\tilde{V}_k(K)$ 上的椭圆投影为 $\tilde{\Pi}_{\Delta}^K$, 则 $k-3$ 和 $k-2$ 阶的矩量用 $\tilde{\Pi}_{\Delta}^K v$ 的替代:

$$W_k(K) = \left\{ v \in \tilde{V}_k(K) : (v, m)_K = (\tilde{\Pi}_{\Delta}^K v, m)_K, \quad m \in \mathbb{M}_{k-2}(e) \setminus \mathbb{M}_{k-4}(e) \right\}.$$

注 9.1 上面的 enhancement 处理, 在 Poisson 方程的二阶及三阶虚拟元方法中已经说明过.

这样, 四阶奇异摄动问题的虚拟元空间为

$$V_h(K) = W_k(K).$$



后面计算过程中仅考虑最低阶, 即 $k = 2$ 的情形. 分量空间的自由度排列为

- 顶点值泛函

$$\chi_i(v) = v(z_i), \quad i = 1, \dots, N_v.$$

- 边中点值泛函

$$\chi_{i+N_v}(v) = v(m_i), \quad i = 1, \dots, N_v.$$

- 边中点法向导数的矩量

$$\chi_{2N_v+i}(v) = \int_{e_i} \partial_n v ds, \quad i = 1, \dots, N_v.$$

9.3 Poisson 项的处理

为了验证计算的正确性, 我们先在 C^0 板弯元空间中计算 Poisson 问题.

9.3.1 过渡矩阵

过渡矩阵与板弯问题 C^0 元的一致.

```

1 % element information
2 index = elem{iel};      Nv = length(index);      Ndof = 3*Nv;
3 xK = centroid(iel,1);  yK = centroid(iel,2);  hK = diameter(iel);
4 x = node(index,1);    y = node(index,2); % vertices
5 v1 = 1:Nv; v2 = [2:Nv,1]; % edge index
6 xe = (x(v1)+x(v2))/2; ye = (y(v1)+y(v2))/2; % mid-edge
7 Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors
8 he = sqrt(sum(Ne.^2,2));
9 ne = Ne./repmat(he,1,2); % outer normal vectors
10 te = [-ne(:,2), ne(:,1)];
11 % scaled monomials
12 m = @ (x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK^2, ...
13 (x-xK).*(y-yK)/hK^2, (y-yK).^2/hK^2]; % m1, ..., m6
14 Gradm = @ (x,y) [[0,0]; [1, 0]/hK; [0, 1]/hK; [2*(x-xK), 0]/hK^2;
15 [(y-yK), (x-xK)]/hK^2; [0, 2*(y-yK)]/hK^2]; % grad m
16 % D
17 D1 = zeros(Ndof,6);
18 D1(1:2*Nv,:) = [m(x,y); m(xe,ye)]; % vertices and mid-edge points
19 for i = 1:Nv
20     gradi = 0.5*(Gradm(x(v1(i)),y(v1(i)))+Gradm(x(v2(i)),y(v2(i))));
21     D1(2*Nv+i,:) = Ne(i,:)*gradi';
22 end
23 D{iel} = D1;

```

9.3.2 椭圆投影 Π_{Δ}^K

对 $k = 2$, Poisson 部分涉及到单元 K 上的零阶矩量

$$\chi_K(v) = \frac{1}{|K|} \int_K v dx,$$

具体来说就是需要 $W_k(K)$ 基函数的内部矩量

$$\chi_K(\phi^T) = \frac{1}{|K|} \int_K \phi^T dx.$$

根据规定, 这部分矩量用 $\tilde{\Pi}_{\Delta}^K \phi^T$ 的矩量代替, 即

$$\chi_K(\phi^T) = \frac{1}{|K|} \int_K \phi^T dx = \frac{1}{|K|} \int_K \tilde{\Pi}_{\Delta}^K \phi^T dx = \chi_K(\tilde{\Pi}_{\Delta}^K \phi^T).$$

为此, 我们必须计算一个子问题, 即计算椭圆投影 $\tilde{\Pi}_{\Delta}^K$ 的矩阵 $\tilde{\Pi}_{\Delta}^K$, 它的第 j 列就是第 j 个基函数投影的自由度向量, 而最后一行就是所有基函数投影的内部矩量.

注 9.2 $\tilde{\Pi}_{\Delta}^K$ 是在 $\tilde{V}_k(K)$ 中计算的. 要注意的是, $W_k(K)$ 和 $\tilde{V}_k(K)$ 相同自由度对应的基函数不一定相同 (毕竟不唯一), 但是基函数对应的自由度值相同, 因而不会影响计算.

现在我们来计算 $\tilde{\Pi}_{\Delta}^K$, 为此要定义椭圆投影

$$\tilde{\Pi}_{\Delta}^K : \tilde{V}_k(K) \rightarrow \mathbb{P}_k(K), \quad v \mapsto \tilde{\Pi}_{\Delta}^K v,$$

满足

$$\begin{cases} a_{\Delta}^K(\tilde{\Pi}_{\Delta}^K v, p) = a_{\Delta}^K(v, p), & p \in \mathbb{P}_k(K), \\ \int_{\partial K} \tilde{\Pi}_{\Delta}^K v ds = \int_{\partial K} v ds, \\ \int_{\partial K} \nabla \tilde{\Pi}_{\Delta}^K v ds = \int_{\partial K} \nabla v ds. \end{cases}$$

先计算过渡矩阵, 它比 C^0 元那里多一行, 对应内部矩量.

```

1  Di = zeros(Ndof+1,6);
2  Di(1:Ndof,:) = D1;
3  nodeT = [node(index,:);aux.centroid(iel,:)];
4  elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
5  Di(end,:) = 1/area(iel)*integralTri(m,2,nodeT,elemT); % inner d.o.f

```

接着计算 B , 它比板弯问题多出一列, 对应最后一个自由度. 由于 $B_{\alpha j}$ 的右端不涉及到该矩量 (见式 (8.62)), 因此最后一列为零, 其他列不变. 限制条件也是如此. 这表明, 此时的 B_s 也只比原来的多出一列零向量, 因而 C^0 程序的过程几乎不变.

投影矩阵最后一行的前 $3N_v$ 个元素组成的行向量就是 $\chi_K(\tilde{\Pi}_{\Delta}^K \phi^T)$, 从而可得 $\chi_K(\phi^T)$. 程序中记为 Dof , 如下计算

```

1  % \partial_ij (m)
2  D11 = zeros(6,1); D11(4) = 2/hK^2;
3  D12 = zeros(6,1); D12(5) = 1/hK^2;
4  D22 = zeros(6,1); D22(6) = 2/hK^2;
5  % Mij (m)
6  nu = 0.3;
7  M11 = -((1-nu)*D11 + nu*(D11+D22));

```

```

8      M12 = -(1-nu)*D12;
9      M22 = -((1-nu)*D22 + nu*(D11+D22));
10     % Mnn(m) on e1,...,eNv
11     Mnn = M11*(n1.*n1) + M12*(n1.*n2+n2.*n1) + M22*(n2.*n2);
12     % Mtn(m) on e1,...,eNv
13     Mtn = M11*(t1.*n1) + M12*(t1.*n2+t2.*n1) + M22*(t2.*n2);
14     % B, Bs, G, Gs
15     B1 = zeros(6,Ndof+1);
16     for j = 1:Nv % loop of edges
17         % nphi on ej
18         nphi = zeros(1,Ndof+1); nphi(2*Nv+j) = 1;
19         % Jump at zj
20         Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
21         % phi at zj
22         phi = zeros(1,Ndof+1); phi(j) = 1;
23         % B1
24         B1 = B1 - Mnn(:,j)*nphi + Jump*phi;
25     end
26     B1s = B1;
27     % first constraint
28     B1s(1,1:Nv) = 1/Nv;
29     % second constraint
30     B1s(2:3,1:Nv) = te([Nv,1:Nv-1],:) ' - te';
31     B1s(2:3,2*Nv+1:Ndof) = ne';
32     % G, Gs
33     G1s = B1s*Di;
34     % Pi, Pis
35     Pis = G1s\B1s; Pi = Di*Pis;
36     Dof = Pi(end,1:end-1);

```

9.3.3 a_{∇}^K 的椭圆投影

定义椭圆投影

$$\Pi_{\nabla}^K : V_h(K) \rightarrow \mathbb{P}_k(K), \quad v \mapsto \Pi_{\nabla}^K v,$$

满足

$$a_{\nabla}^K(\Pi_{\nabla}^K u, p) = a_{\nabla}^K(u, p), \quad p \in \mathbb{P}_k(K),$$

限制条件为

$$\int_{\partial K} \Pi_{\nabla}^K u \, ds = \int_{\partial K} u \, ds.$$

注意, 此时要计算 $k = 2$ 的问题.

上面的定义等价于如下的向量形式

$$a_{\nabla}^K(m, \Pi_{\nabla}^K \phi^T) = a_{\nabla}^K(m, \phi^T)$$

或

$$\int_K \nabla m \cdot \nabla \Pi_{\nabla}^K \phi^T \, dx = \int_K \nabla m \cdot \nabla \phi^T \, dx,$$

限制条件为

$$\sum_{i=1}^{N_v} \Pi_{\nabla}^K \phi^T(z_i) = \sum_{i=1}^{N_v} \phi^T(z_i).$$

设 Π_{∇}^K 在基 ϕ^T 下的矩阵为 $\mathbf{\Pi}_{\nabla}$, 即

$$\Pi_{\nabla}^K \phi^T = \phi^T \mathbf{\Pi}_{\nabla},$$

而 $\Pi_{\nabla}^K \phi^T$ 在多项式基 m^T 下的矩阵为 $\mathbf{\Pi}_{\nabla}^*$, 即

$$\Pi_{\nabla}^K \phi^T = m^T \mathbf{\Pi}_{\nabla}^*,$$

易知有

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

将以上表达式代入向量形式, 有

$$\begin{cases} G_{\nabla} \mathbf{\Pi}_{\nabla}^* = B_{\nabla}, \\ \frac{1}{N_v} \sum_{i=1}^{N_v} \mathbf{m}^T(z_i) \mathbf{\Pi}_{\nabla}^* = \frac{1}{N_v} \sum_{i=1}^{N_v} \phi^T(z_i) \end{cases}$$

式中,

$$G_{\nabla} = a_{\nabla}^K(m, m^T) = \int_K \nabla m \cdot \nabla m^T dx,$$

$$B_{\nabla} = a_{\nabla}^K(m, \phi^T) = \int_K \nabla m \cdot \nabla \phi^T dx.$$

而 G_{∇} 和 B_{∇} 的某些行用限制条件替换后记为 \tilde{G}_{∇} 和 \tilde{B}_{∇} . 我们有一致性关系

$$G_{\nabla} = B_{\nabla} D, \quad \tilde{G}_{\nabla} = \tilde{B}_{\nabla} D.$$

现在计算 B_{∇} , 即

$$B_{\nabla} = \int_K \nabla m \cdot \nabla \phi^T dx = - \int_K \Delta m \phi^T dx + \int_{\partial K} \partial_n m \phi^T ds.$$

为了方便, 记

$$I_1 = \int_K \Delta m \phi^T dx, \quad I_2 = \int_{\partial K} \partial_n m \phi^T ds.$$

先计算 I_1 . 因为 m 的分量是二次多项式, 所以 Δm 是常向量, 从而

$$I_1 = \Delta m \int_K \phi^T dx.$$

直接计算可知,

$$\Delta m = [0, 0, 0, \frac{2}{h_K^2}, 0, \frac{2}{h_K^2}]^T.$$

根据规定,

$$\int_K \phi^T dx = |K| \cdot \text{Dof}.$$

由此可知第一项如下获得

```

1      % first term
2      Lapm = zeros(6,1); Lapm([4,6]) = 2/hK^2;
3      Dof1 = area(iel)*Dof;
4      I1 = Lapm*Dof1;

```

再计算 I_2 , 即

$$I_2 = \int_{\partial K} \partial_n m \phi^T ds = \sum_{i=1}^{N_v} \int_{e_i} \partial_n m \phi^T ds.$$

注意到 $\partial_n m \phi^T$ 在每条边上是三次多项式, 右端的积分可用 Simpson 公式计算. 记 $\mathbf{f} = \partial_n m \phi^T$, 则

$$I_2 = \frac{1}{6} \sum_{i=1}^{N_v} h_{e_i} [\mathbf{f}(z_i) + 4\mathbf{f}(z_m) + \mathbf{f}(z_{i+1})],$$

其中,

$$\begin{aligned}\mathbf{f}(z_i) &= \partial_n m(z_i)[\mathbf{e}_i, \mathbf{0}, \mathbf{0}], \\ \mathbf{f}(z_m) &= \partial_n m(z_m)[\mathbf{0}, \mathbf{e}_i, \mathbf{0}], \\ \mathbf{f}(z_{i+1}) &= \partial_n m(z_{i+1})[\mathbf{e}_{i+1}, \mathbf{0}, \mathbf{0}].\end{aligned}$$

```

1      % second term
2      I2 = 0;
3      for j = 1:Nv % loop of edges
4          % he*\partial_n(m) at x_j, xe, x_j+1
5          DnL = Gradm(x(j),y(j))*Ne(j,:)';
6          Dne = Gradm(xe(j),ye(j))*Ne(j,:)';
7          DnR = Gradm(x(v2(j)),y(v2(j)))*Ne(j,:)';
8          % [ei,0,0]
9          e1 = zeros(1,Ndof); e1(j) = 1;
10         e2 = zeros(1,Ndof); e2(Nv+j) = 1;
11         e3 = zeros(1,Ndof); e3(v2(j)) = 1;
12         % f
13         f1 = DnL*e1; f2 = Dne*e2; f3 = DnR*e3;
14         I2 = I2 + (f1+4*f2+f3);
15     end
16     B2 = -I1 + 1/6*I2;

```

投影的限制条件如下处理.

```

1      % constraint
2      B2s = B2; B2s(1,1:Nv) = 1/Nv;
3      Bps{iel} = B2s;
4      Gp{iel} = B2*D{iel}; Gps{iel} = B2s*D{iel};

```

9.4 板弯项的处理

板弯项与板弯问题 C^0 非协调元的过程完全相同. 下面给出投影矩阵的计算过程.

```

1 %% Compute projection matrices
2 D = cell(NT,1);
3 Bps = cell(NT,1); Gp = cell(NT,1); Gps = cell(NT,1); % p: Poisson
4 Bbs = cell(NT,1); Gb = cell(NT,1); Gbs = cell(NT,1); % b: biharmonic
5 for iel = 1:NT
6     % 0. ----- element information -----
7     index = elem{iel}; Nv = length(index); Ndof = 3*Nv;
8     xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
9     x = node(index,1); y = node(index,2); % vertices
10    v1 = 1:Nv; v2 = [2:Nv,1]; % loop index for vertices or edges
11    p1 = [Nv,1:Nv-1]; p2 = 1:Nv; % jump index
12    xe = (x(v1)+x(v2))/2; ye = (y(v1)+y(v2))/2; % mid-edge points
13    Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % scaled outer normal vectors

```

```

14     he = sqrt(sum(Ne.^2,2));
15     ne = Ne./repmat(he,1,2); % outer normal vectors
16     te = [-ne(:,2), ne(:,1)];
17     n1 = ne(:,1)'; n2 = ne(:,2)';
18     t1 = te(:,1)'; t2 = te(:,2)';
19     % scaled monomials
20     m = @(x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK^2, ...
21                 (x-xK).* (y-yK)/hK^2, (y-yK).^2/hK^2]; % m1,...,m6
22     Gradm = @(x,y) [[0,0]; [1, 0]/hK; [0, 1]/hK; [2*(x-xK), 0]/hK^2;
23                           [(y-yK), (x-xK)]/hK^2; [0, 2*(y-yK)]/hK^2]; % grad m
24
25     % 1. ----- transition matrices -----
26     D1 = zeros(Ndof,6);
27     D1(1:2*Nv,:) = [m(x,y); m(xe,ye)]; % vertices and mid-edge points
28     for i = 1:Nv
29         gradi = 0.5*(Gradm(x(v1(i)),y(v1(i)))+Gradm(x(v2(i)),y(v2(i))));
30         D1(2*Nv+i,:) = Ne(i,:)*gradi';
31     end
32     D{iel} = D1;
33
34     % 2. ----- elliptic projection of plate bending term -----
35     % \partial_ij (m)
36     D11 = zeros(6,1); D11(4) = 2/hK^2;
37     D12 = zeros(6,1); D12(5) = 1/hK^2;
38     D22 = zeros(6,1); D22(6) = 2/hK^2;
39     % Mij(m)
40     nu = 0.3;
41     M11 = -((1-nu)*D11 + nu*(D11+D22));
42     M12 = -(1-nu)*D12;
43     M22 = -((1-nu)*D22 + nu*(D11+D22));
44     % Mnn(m) on e1,...,eNv
45     Mnn = M11*(n1.*n1) + M12*(n1.*n2+n2.*n1) + M22*(n2.*n2);
46     % Mtn(m) on e1,...,eNv
47     Mtn = M11*(t1.*n1) + M12*(t1.*n2+t2.*n1) + M22*(t2.*n2);
48     % B, Bs, G, Gs
49     B1 = zeros(6,Ndof);
50     for j = 1:Nv % loop of edges
51         % nphi on ej
52         nphi = zeros(1,Ndof); nphi(2*Nv+j) = 1;
53         % Jump at zj
54         Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
55         % phi at zj
56         phi = zeros(1,Ndof); phi(j) = 1;
57         % B1
58         B1 = B1 - Mnn(:,j)*nphi + Jump*phi;
59     end
60     B1s = B1;
61     % first constraint
62     B1s(1,1:Nv) = 1/Nv;
63     % second constraint
64     B1s(2:3,1:Nv) = te([Nv,1:Nv-1],:) - te';
65     B1s(2:3,2*Nv+1:end) = ne';
66     Bbs{iel} = B1s;
67     Gb{iel} = B1*D1; Gbs{iel} = B1s*D1;
68
69     % 3. --- Interior d.o.f.s of elliptic projection of basis functions ---
70     Di = zeros(Ndof+1,6);
71     Di(1:Ndof,:) = D1;

```

```

72     nodeT = [node(index,:);aux.centroid(iel,:)];
73     elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]';
74     Di(end,:) = 1/area(iel)*integralTri(m,2,nodeT,elemT); % inner d.o.f
75     Bis = zeros(6,Ndof+1); Bis(:,1:Ndof) = B1s;
76     % G, Gs
77     Gis = Bis*Di;
78     % Pi, Pis
79     Pis = Gis\Bis; Pi = Di*Pis;
80     Dof = Pi(end,1:end-1);
81
82     % 4. ----- elliptic projection for Poisson term -----
83     % first term
84     Lapm = zeros(6,1); Lapm([4,6]) = 2/hK^2;
85     Dof1 = area(iel)*Dof;
86     I1 = Lapm*Dof1;
87     % second term
88     I2 = 0;
89     for j = 1:Nv % loop of edges
90         % he*\partial_n(m) at x_j, xe, x_j+1
91         DnL = Gradm(x(j),y(j))*Ne(j,:)';
92         Dne = Gradm(xe(j),ye(j))*Ne(j,:)';
93         DnR = Gradm(x(v2(j)),y(v2(j)))*Ne(j,:)';
94         % [ei,0,0]
95         e1 = zeros(1,Ndof); e1(j) = 1;
96         e2 = zeros(1,Ndof); e2(Nv+j) = 1;
97         e3 = zeros(1,Ndof); e3(v2(j)) = 1;
98         % f
99         f1 = DnL*e1; f2 = Dne*e2; f3 = DnR*e3;
100        I2 = I2 + (f1+4*f2+f3);
101    end
102    B2 = -I1 + 1/6*I2;
103    % constraint
104    B2s = B2; B2s(1,1:Nv) = 1/Nv;
105    Bps{iel} = B2s;
106    Gp{iel} = B2*D{iel}; Gps{iel} = B2s*D{iel};
107 end

```

9.5 刚度矩阵与载荷向量的计算

Poisson 项椭圆投影对应的矩阵为

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

稳定项为

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

类似可给出板弯项的矩阵.

再考虑右端的计算

$$F_K(\phi) = \int_K \Pi_{k-2}^0 f \phi dx.$$

当 $k = 2$ 时, $\Pi_{k-2}^0 = \Pi_0^0 = P_0$, 即常值投影, 由 (5.14) 定义, 即单元顶点值的平均. 注意到

$$F_K(\phi) = \int_K \Pi_0^0 f \phi dx = \int_K \Pi_0^0 f \Pi_k^0 \phi dx,$$

我们有

$$F_K = \int_K \Pi_0^0 f \Pi_0^0 \phi dx = f(x_K, y_K) |K| \frac{1}{N_v} [\mathbf{1}_{N_v}, \mathbf{0}, \mathbf{0}]^T,$$

这里为了方便, $\Pi_0^0 f$ 用中点值近似.

如下获得刚度矩阵和载荷向量

```

1 %% Get elementwise stiffness matrix and load vector
2 ABelem = cell(NT,1); belem = cell(NT,1);
3 bdIndex = bdStruct.bdIndex; E = false(NE,1); E(bdIndex) = 1;
4 for iel = 1:NT
5     % sign matrix and sign vector
6     index = elem{iel}; Nv = length(index); Ndof = 3*Nv;
7     sgnelem = sign(diff(index([1:Nv,1])));
8     id = elem2edge{iel}; sgnbd = E(id); sgnelem(sgnbd) = 1;
9     sgnbase = ones(Ndof,1); sgnbase(2*Nv+1:3*Nv) = sgnelem;
10    sgnK = sgnbase*sgnbase'; sgnF = sgnbase;
11    % Projection
12    Pips = Gps{iel}\Bps{iel}; Pip = D{iel}*Pips; Ip = eye(size(Pip));
13    Pibs = Gbs{iel}\Bbs{iel}; Pib = D{iel}*Pibs; Ib = eye(size(Pib));
14    % Stiffness matrix
15    AK = Pips'*Gp{iel}*Pips + (Ip-Pip)'*(Ip-Pip);
16    BK = Pibs'*Gb{iel}*Pibs + hK^(-2)*(Ib-Pib)'*(Ib-Pib);
17    ABK = (AK+para.epsilon^2*BK).*sgnK;
18    ABelem{iel} = reshape(ABK',1,[]); % straighten
19    % Load vector
20    fK = zeros(Ndof,1);
21    POf = pde.f(centroid(iel,:))*area(iel)/Nv;
22    fK(1:Nv,:) = repmat(POf,Nv,1);
23    belem{iel} = (fK.*sgnF)'; % straighten
24 end

```

这里, 符号刚度矩阵类似板弯问题获得.

刚度矩阵与载荷向量如下装配

```

1 %% Assemble stiffness matrix and load vector
2 elemLen = cellfun('length',elem); vertNum = unique(elemLen);
3 nnz = sum((3*elemLen).^2);
4 ii = zeros(nnz,1); jj = zeros(nnz,1); ss = zeros(nnz,1);
5 id = 0; ff = zeros(NNdof,1);
6 for Nv = vertNum(:) % only valid for row vector
7
8     % assemble the matrix
9     idNv = find(elemLen == Nv); % find polygons with Nv vertices
10    NTv = length(idNv); % number of elements with Nv vertices
11
12    elemNv = cell2mat(elem(idNv)); % elem
13    elem2edgeNv = cell2mat(elem2edge(idNv));
14    elem2 = [elemNv, elem2edgeNv+N, elem2edgeNv+N+NE];
15    K = cell2mat(ABelem(idNv)); F = cell2mat(belem(idNv));
16    Ndof = 3*Nv;
17    ii(id+1:id+NTv*Ndof^2) = reshape(repmat(elem2, Ndof,1), [], 1);
18    jj(id+1:id+NTv*Ndof^2) = repmat(elem2(:,1), Ndof, 1);
19    ss(id+1:id+NTv*Ndof^2) = K(:,1);
20    id = id + NTv*Ndof^2;
21
22    % assemble the vector
23    ff = ff + accumarray(elem2(:,1),F(:,1),[NNdof 1]);
24 end

```

```
25 kk = sparse(ii,jj,ss,NNdof,NNdof);
```

9.6 边界条件的处理

与板弯问题完全相同, 如下

```
1 %% Apply Dirichlet boundary conditions
2 % boundary information
3 bdNodeIdx = bdStruct.bdNodeIdx; bdEdgeD = bdStruct.bdEdgeD;
4 g_D = pde.g_D; Dw = pde.Du;
5 id = [bdNodeIdx; bdEdgeIdx+N; bdEdgeIdx+N+NE];
6 isBdNode = false(NNdoF,1); isBdNode(id) = true;
7 bdDof = (isBdNode); freeDof = (~isBdNode);
8 % vertices on the boundary
9 pD = node(bdNodeIdx,:); wD = g_D(pD);
10 % mid-edge on the boundary
11 z1 = node(bdEdgeD(:,1),:); z2 = node(bdEdgeD(:,2),:); zc = (z1+z2)./2;
12 wDc = g_D(zc);
13 % moments on the boundary
14 eb = z1-z2; % e = z2-z1
15 neb = [-eb(:,2),eb(:,1)]; % scaled ne
16 wnD = sum(1/6*(Dw(z1)+4*Dw(zc)+Dw(z2)).*neb,2);
17 % rhs
18 w = zeros(NNdoF,1); w(bdDof) = [wD; wDc; wnD];
19 ff = ff - kk*w;
20
21 %% Set solver
22 w(freeDof) = kk(freeDof,freeDof)\ff(freeDof);
```

9.7 误差分析

例 9.1 不含边界层. 设精确解为

$$u(x, y) = (\sin(\pi x) \sin(\pi y))^2.$$

我们计算如下的能量范数误差 $\|u - \Pi_{\Delta}^K u_h\|$, 这里用四阶项的投影算子近似. 使用 meshdata100-500, 拟合得到的误差阶如下

表 3. 不同 ε 对应的能量范数误差阶

| ε | 2^0 | 2^{-2} | 2^{-4} | 2^{-6} | 2^{-8} | 2^{-10} | 0 |
|---------------|-------|----------|----------|----------|----------|-----------|------|
| Err | 1.58 | 1.02 | 1.12 | 1.80 | 2.01 | 2.02 | 2.03 |

可以看到, C^0 元确保了一阶收敛性. 当 ε 趋向于零, 即方程趋向于 Poisson 方程时, 收敛阶达到 2 阶.

例 9.2 带边界层. 精确解为

$$u(x, y) = \varepsilon(e^{-x_1/\varepsilon} + e^{-x_2/\varepsilon}) - x_1^2 x_2.$$

表 4. 不同 ε 对应的能量范数误差阶

| ε | 10^0 | 10^{-1} | 10^{-2} | 10^{-3} | 10^{-4} | 10^{-5} | 10^{-6} |
|---------------|--------|-----------|-----------|-----------|-----------|-----------|-----------|
| Err | 1.20 | 0.49 | 0.65 | 0.48 | 0.46 | 0.46 | 0.46 |

对含边界层的例子, 可以看到, C^0 元确保了 $1/2$ 阶的收敛性.

References

- [1] L. Beirão da Veiga, F. Brezzi, L. D. Marini and A. Russo. The Hitchhiker's Guide to the Virtual Element Method. *Mathematical Models and Methods in Applied Sciences*, 24(8): 1541-1573, 2014.
- [2] J. Zhao, B. Zhang, S. Chen, et al. The Morley-Type Virtual Element for Plate Bending Problems. *Journal of Scientific Computing*, 76: 610 - 629, 2018.
- [3] T. K. Nilssen, X. C. Tai and R. Winther. A robust nonconforming H^2 -element. *Mathematics of Computation*, 70(234): 489 - 505, 2000.