3 PolyMesher: 二维多角形剖分的生成

本章节介绍文 [2] 中的多角形剖分生成算法.

3.1 网格剖分的基本思想

3.1.1 Voronoi 图

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

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

区域 Δ 的 Voronoi 划分定义为

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

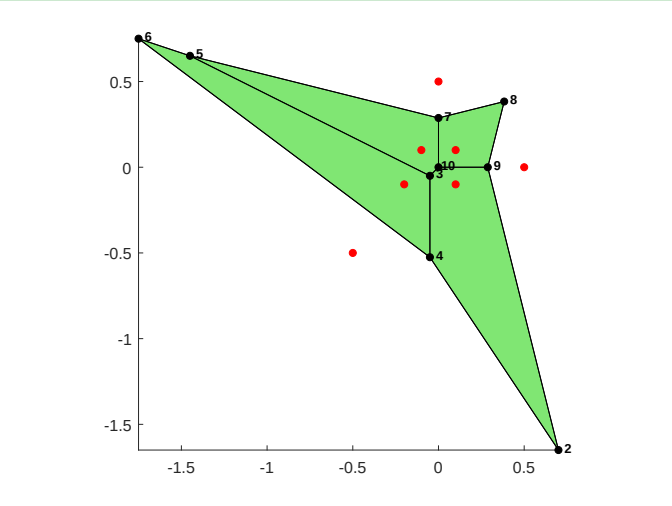
显然区域 Δ 的 Voronoi 划分就是将平面的 Voronoi 划分附加上边界 $\partial\Delta$ 所得.

- 对平面两点, 易知 Voronoi 划分由两点之间的垂直平分线确定.
- 对多个点的 Voronoi 图, 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.

MATLAB 中可用 voronoin.m 函数生成 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



- `node` 的第一个点的坐标都是 `(Inf,Inf)`, 即无穷远点, 且统一用索引 1 标记.
- 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. 可以看到, 多角形的边界是通过种子之间的垂直平分线确定的.

3.1.2 CVTs 与 Lloyd 迭代

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 tessellations, CVTs). 给定密度函数 $\mu(x)$, Voronoi 单元的重心定义为

$$y_c = \frac{\int_{V_y\cap\Delta} x\mu(x)\mathrm{d}x}{\int_{V_y\cap\Delta} \mu(x)\mathrm{d}x}. \tag{3.1}$$

定义 3.2 $\mathcal{T}(P;\Delta)$ 称为一个 CVT, 如果对每个 $y\in P$, 都有

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

当 $\mu(x)\equiv 1$ 时, y_c 就是多角形单元 $V_y\cap\Delta$ 的形心.

定义能量泛函

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

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

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

下面给出计算 CVTs 的 Lloyd 算法.

定义 3.3 定义

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

式中,

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

称其为 Lloyd 映射.

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

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

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

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

因而 Lloyd 算法可视为求能量泛函极值点的下降算法.

多角形或多面体网格剖分的过程可总结如下

Algorithm 1 多角形或多面体网格剖分

Step 1: 给定区域内的若干个种子, 生成对应的 Voronoi 图.

Step 2: 使用 Lloyd 迭代算法更新种子的位置.

Step 3: 回到 Step 1, 直到达到停止准则.

可以看到, 整个算法的核心是给定区域 Voronoi 图的生成. MATLAB 里用 voronoin.m 函数生成 Voronoi 图, 但它没有边界, 仅是空间的划分. 这样, 问题的难点转化为: 如何给无界的 Voronoi 图添加边界. PolyMesher 中采用的边界附近种子的反射方法.

3.2 有界 Voronoi 图的反射法构造

3.2.1 区域边界的近似

给定一个凸区域, 将每一个种子关于最近的边界做对称点, 称该点为反射点, 而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.

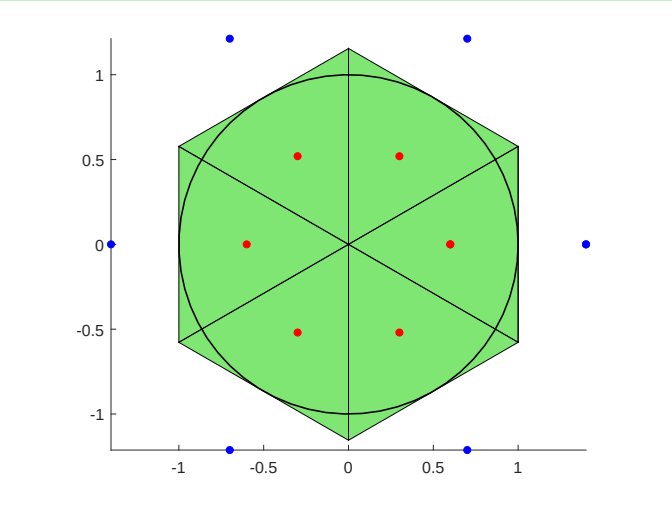
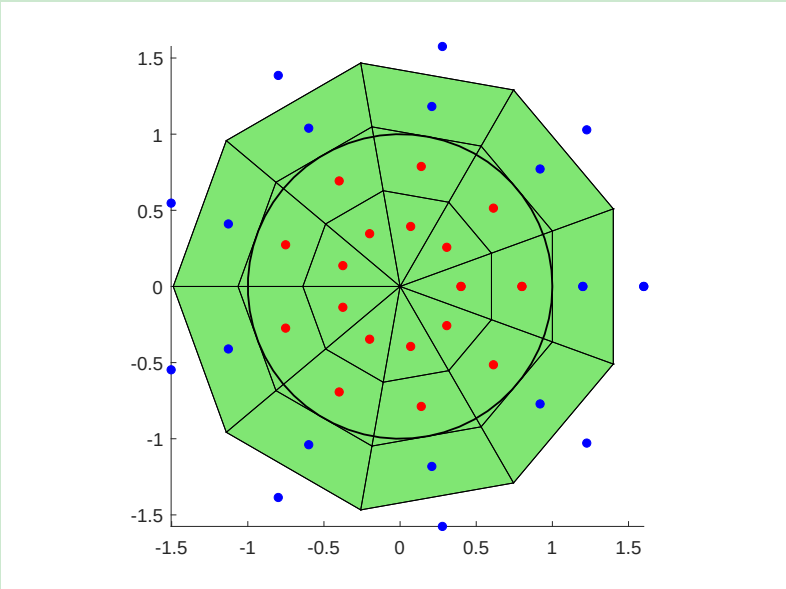


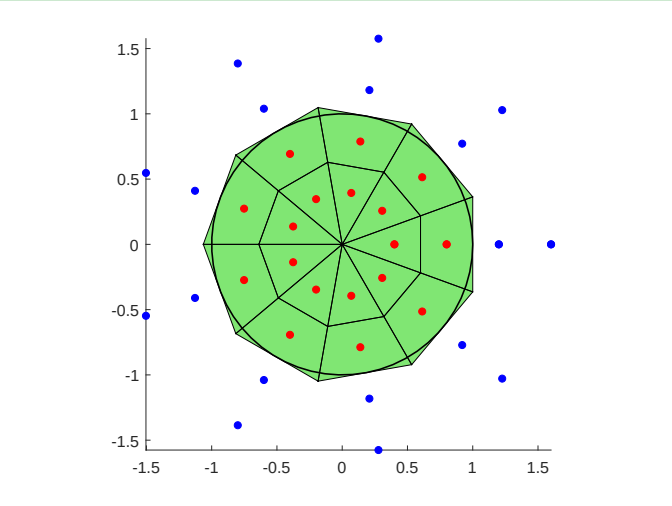
图 10. $\mathcal{T}(\tilde{P};\Delta)$

如图 10 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:

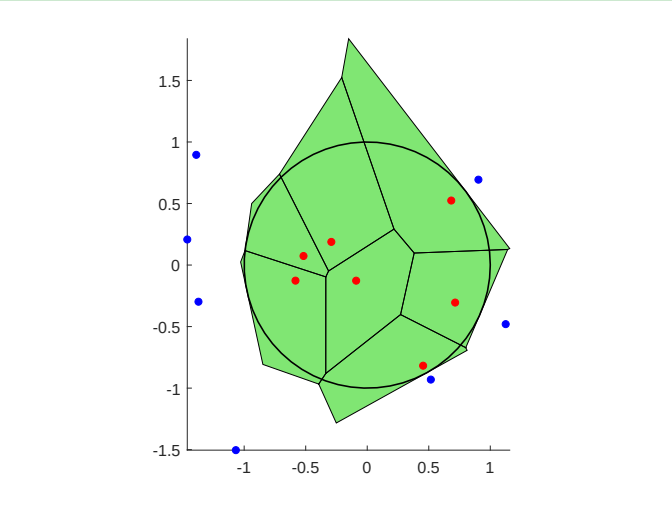
- 反射点 $R(y)$ 的 Voronoi 单元在区域外, 它与种子 y 的单元有公共边界, 且与凸区域的边界相切.
- $\mathcal{T}(\tilde{P};\Delta)$ 有 `2*NT` 个单元, 前 `NT` 个对应区域内的种子, 因而是该区域的一个覆盖.
- 当凸区域内的点充分多且分布质量较高时, 这个覆盖就是一个较好的剖分图.
- 注意上面给出的图外部的单元恰好是空白的, 它们与无穷原点构成“多角形”. 但外部也可能存在可以直接观察到的剖分, 如下图



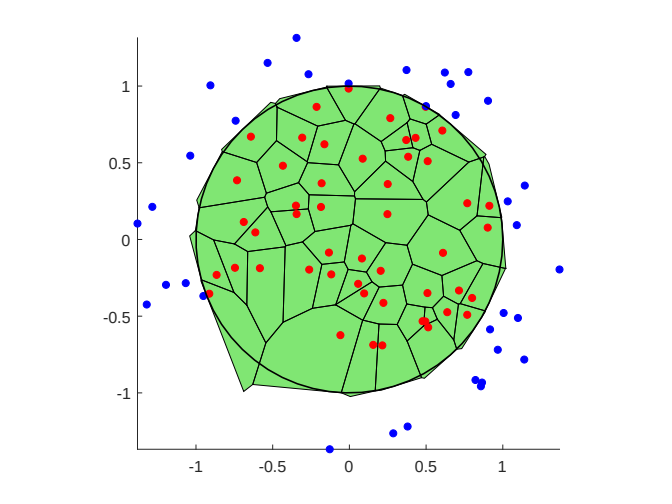
取前 `NT` 个单元的结果如下



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, 可采用前面所述的 CVTs, 获取种

子的高质量分布.

3.2.2 边界反射集的计算

符号距离函数

定义 3.4 设 $\Omega \subset \mathbb{R}^2$, 定义 $d_\Omega : \mathbb{R}^2 \to \mathbb{R}$ 为

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

式中,

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

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

显然有

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

对圆心在 x_0, 半径为 r 的球, 易知

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

当 Ω 光滑时, $\nabla d_\Omega(x)$ 是最近边界点处的单位法向量, 例如对圆, 有 $\nabla d_\Omega(x) = \operatorname{sgn}(x - x_0)$. 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

$$R_\Omega(x) = x - 2 d_\Omega(x) \nabla d_\Omega(x). \tag{3.2}$$

对复合区域, 通常有

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

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

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

考虑圆形区域, 其表示方法如下

```
function x = Circle_Domain(Demand,P)
BdBox = [-1 1 -1 1];
switch(Demand)
    case('Dist');  x = DistFnc(P);
    case('BdBox'); x = BdBox;
end
%--------- the signed distance function ----------
function Dist = DistFnc(P,BdBox)
Dist = dCircle(P,0,0,1);

function Dist = dCircle(P,xc,yc,r)
d = sqrt((P(:,1)-xc).^2+(P(:,2)-yc).^2)-r;
Dist = [d,d];
```

在该段代码中, BdBox 是区域的一个合适的矩形覆盖.

若某个区域的边界由 m 个部分组成, 第 i 个部分的符号距离函数为 d_i, 且整体符号距离函数为 d, 则计算中会如下存储

$$\text{Dist} = [d_1, \cdots, d_m, d]. \tag{3.3}$$

对单个边界, $d_1 = d_m = d$, 因而 Dist $= [d, d]$, 这就是上面程序中 [d,d] 写法的原因.

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. 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. 在矩形内逐一生成随机点, 并用符号距离函数判断是否在 Ω 内, 直至获得 NT 个种子为止.

随机种子如下生成

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

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

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

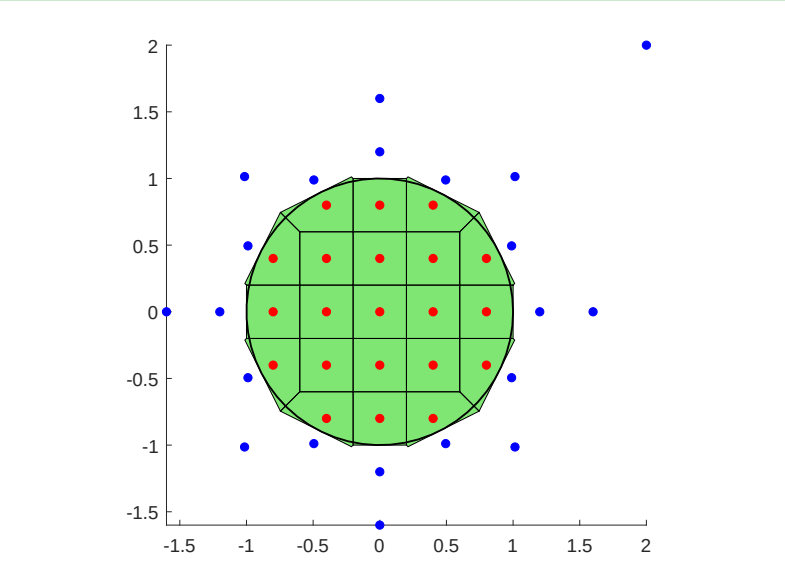
以上过程编写成函数文件

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

BdBox = Domain('BdBox');
nvar = length(varargin);

% ------------ Generate random pointset -----------
if nvar==1
    NT = varargin{1}; P = zeros(NT,2);  s = 0;
```

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

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

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, 它的符号距离函数容易获得. 反射点用公式 (3.2) 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, 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, 为此只对边界附近的点进行反射. 这种做法可避免以上情形的发生, 且大大减少计算量.

给定 n 个种子, 最终剖分的单元平均宽度可用为 $|\Omega|/n$ 度量. 引入参数

$$\alpha(n,\Omega) := c\Big(\frac{|\Omega|}{n}\Big)^{1/2}, \quad c>1,$$

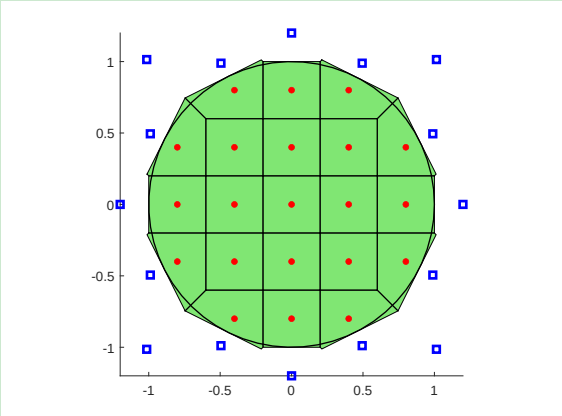
它是比平均宽度略大的数. 小于该数的点都可认为是边界附近的点, 即仅对满足

$$d_\Omega(y) < \alpha(n, \Omega) \tag{3.4}$$

的种子进行反射. 程序如下

```
clc;clear;close all
% ------- Generate intial pointset --------
Domain = @Circle_Domain;
% NT = 50;
% P = PolyMesher_init_Pointset(Domain,NT);
Nx = 5; Ny = 5;
P = PolyMesher_init_Pointset(Domain,Nx,Ny);
NT = size(P,1);

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

d = Domain('Dist',P);
I = abs(d(:,1))<Alpha;  % logical index of seeds near the bdry
n1 = (Domain('Dist',P+[eps,0])-d)/eps;
n2 = (Domain('Dist',P+[0,eps])-d)/eps;
R_P = P(I,:) - 2*[n1(I),n2(I)].*repmat(d(I),1,2);

% ------------ Voronoi -------------------
PP = [P; R_P];
[node,elem] = voronoin(PP);
elem = elem(1:NT);
showmesh(node,elem); %findnode(node);
hold on
plot(P(:,1),P(:,2),'r.','MarkerSize',15);
plot(R_P(:,1),R_P(:,2),'bs','linewidth',2)
t = linspace(0,2*pi,60)';
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 $|\Omega|$.

分段光滑边界凸区域反射集的计算

边界分段光滑的凸区域的典型代表是矩形区域, 此时 `BdBox` 可选为该矩形区域. 设矩形为 $[a_1, b_1] \times [a_2, b_2]$, 对左侧直线, 它的右侧为内部, 符号距离函数为

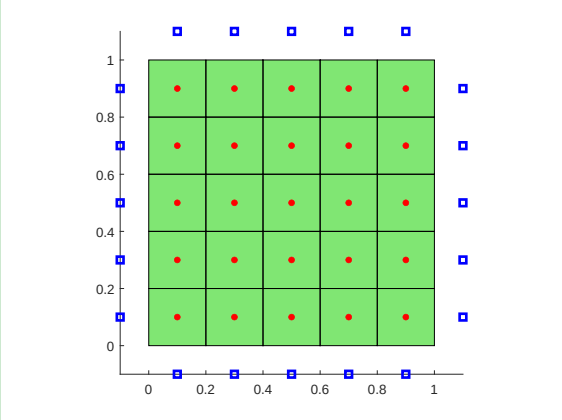
$$d_1(\boldsymbol{x}) = a_1 - \boldsymbol{x}(1),$$

类似下侧、右侧和上侧直线的符号距离函数分别为

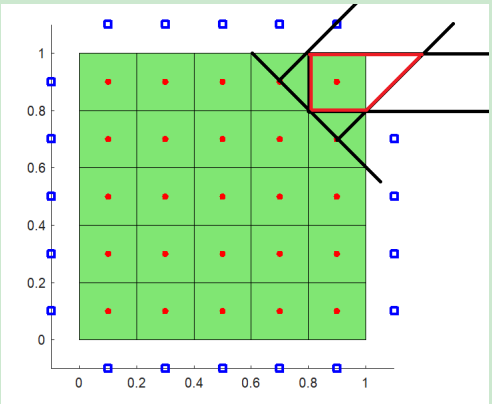
$$d_2(\boldsymbol{x}) = b_1 - \boldsymbol{x}(2), \quad d_3(\boldsymbol{x}) = \boldsymbol{x}(1) - a_2, \quad d_4(\boldsymbol{x}) = \boldsymbol{x}(2) - a_2.$$

整体区域是这些半平面的交, 从而

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



- 对由分段光滑边界组成的区域, 边界附近的点可能要进行多次反射. 例如, 对上图中矩形右上角附近的种子, 若不进行右侧边界反射, 则它对应的 Voronoi 单元应为下图中红色边界的多角形.



  在使用 Lloyd 迭代的过程中, 有时会观察到这种现象. 原因是角点附近的种子可能仅归结为一侧边界的附近点.

- 通过反射两个边界, 上图获得的是标准的四边形剖分. 由于要对多个边进行反射, 因此要保留每个边对应的符号距离函数. 这就是 (3.3) 这种存储方式的原因.

矩形区域如下定义

```
function x = Rectangle_Domain(Demand,P)
  BdBox = [0 1 0 1];
  switch(Demand)
    case('Dist');  x = DistFnc(P,BdBox);
    case('BdBox'); x = BdBox;
  end
%------------ the signed distance function --------------
function Dist = DistFnc(P,BdBox)
  Dist = dRectangle(P,BdBox(1),BdBox(2),BdBox(3),BdBox(4));

function d = dRectangle(P,x1,x2,y1,y2)
 d = [x1-P(:,1), P(:,1)-x2, y1-P(:,2), P(:,2)-y2];
 d = [d,max(d,[],2)];
```

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

反射集如下计算.

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

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

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

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

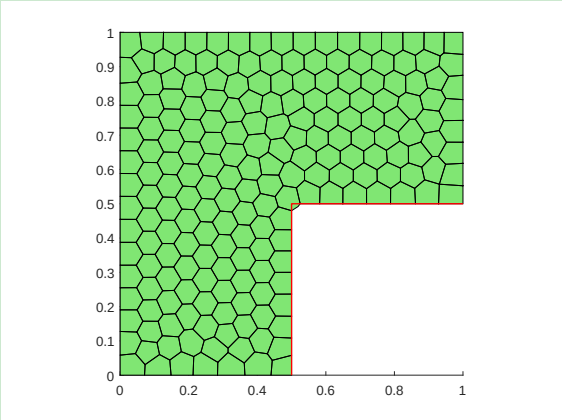
这里,

- 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 则是各条边对应的近边逻辑数组.
- P1 是横坐标, 复制 4 列以应用逻辑数组取值. 注意逻辑数组取值会排成一列, 即 P1(I) 是一个列向量.

- `d(I)` 是各条边对应的近边种子的符号距离.

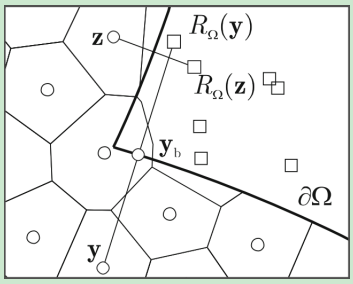
非凸区域反射集的计算

对非凸区域, 考虑 L-型区域. 此时, 采用反射操作不能很好地捕捉 L-型区域的拐点.



- 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在其他种子的反射界面上. 为此需要判断符号距离函数的符号, 排除 $d_{\Omega_i}(\bar{y}) > 0$ 的反射点 $\bar{y} = R_{\Omega_i}(y)$.

- 有时候, 反射点可能比种子距离边界更近.



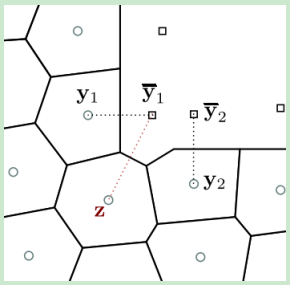
例如, 在该图中, 种子 y 与边界的距离要明显大于反射点 $R_\Omega(y)$ 与边界的距离. 此时, 这个反射点对边界近似没有贡献, 而且可能干扰其他种子的反射. 为此, 要求反射点 $\bar{y}$ 满足

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

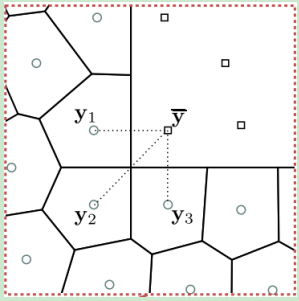
为此, 前面的反射集合后面只需要添加语句

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

对非凸区域的角点, 如 L-型区域的拐点, 前面的标准还不足以保证良好的逼近效果. 这是因为, 当种子位于对角线且在拐点附近时, 其反射可能会干扰其他点的反射, 如下图所示.



对垂直平分简单分析后可知, 一个解决办法是固定拐点附近的种子, 使得它们的反射点相同.



反射集计算程序

计算反射集合的程序如下

```
function RP = PolyMesher_Reflect(P,Domain,Area)

% ---------- Compute the reflection pointset ---------------
eps = 1e-8; eta = 0.9; c = 1.5; NT = size(P,1);
Alpha = c*sqrt(Area/NT);

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

3.3 二维多角形剖分的生成

3.3.1 Lloyd 迭代

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

```
% Construct the Voronoi diagram
 [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.

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

3.3.2 获取网格剖分的基本数据集

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

- 从 elem = elem(1:NT) 中可获取网格节点编号

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

  这里,

  - id 是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

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

  把 totalid 恢复为 elem 的模式就获得新的 elem

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

- 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.

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

上面通过 elem 恢复自然序号的过程可总结为如下函数

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

```
node = node(id,:);
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
for iel = 1:NT
    index = elem{iel};
    z1 = node(index(:,1),:);
    z2 = node(index(:,2),:);
    z3 = node(index(:,3),:);
    e2 = z3-z1; e3 = z1-z2;
    area = 0.5*(-e3(:,1).*e2(:,2)+e3(:,2).*e2(:,1));
    if area<0, elem{iel} = index(end:-1:1); end
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. 现在去除短边.

- 根据辅助数据结构的讨论过程, 可获得所有的一维边集合 edge

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

- 我们将删除如下的边 (PolyMesher 不是这样做的)

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

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

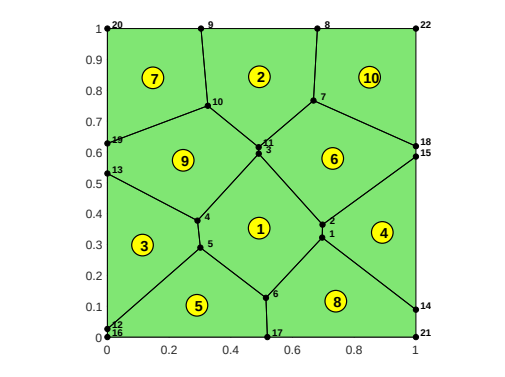
```
z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
he = sqrt(sum((z1-z2).^2,2));
ir = find(he≤0.1*max(he));
nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending numbers
```

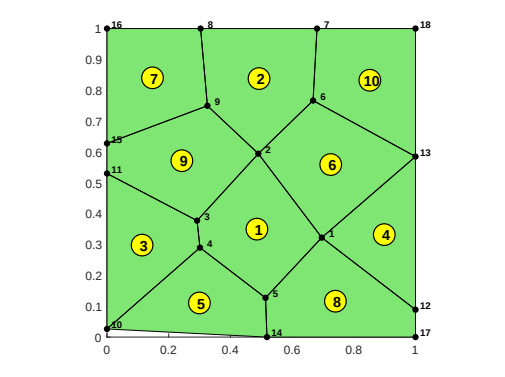
这里 nv1, nv2 是要删除的边的起点和终点编号.

- 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)

```
[id,¬,totalid] = unique(horzcat(elem{:})');
totalid = id(totalid);
for i = 1:length(nv2)
    v1 = nv1(i); v2 = nv2(i);
    totalid(totalid==v2) = v1;
    nv1(nv1==v2) = v1;  nv2(nv2==v2) = v1;
end
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';
```

- 有些时候, 直接保留起点编号可能导致错误. 考察下图





短边 12-16 应保留终点 16, 否则出现右图中的白色空白. 但对反射法来说, 一般不必考虑该问题. 这是因为, 反射法出现的极短边一般位于区域边界外, 直接去除影响不大.

- 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)

```
for iel = 1:NT
    index = elem{iel};
    [¬,i1] = unique(index);
    elem{iel} = index(sort(i1));
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

```
function [node,elem] = PolyMesher_rm_smalledge(NT,node,elem)
[node,elem] = PolyMesher_Reorder(NT,node,elem);
T1 = cellfun(@(verts) [verts(2:end),verts(1)], elem, 'UniformOutput', false);
v0 = horzcat(elem{:})'; v1 = horzcat(T1{:})';
totalEdge = sort([v0,v1],2);
[i,j] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
edge = [j,i];
z1 = node(edge(:,1),:); z2 = node(edge(:,2),:);
he = sqrt(sum((z1-z2).^2,2));
ir = find(he < 0.1*max(he));
nv1 = edge(ir,1); nv2 = edge(ir,2); % starting and ending indices
[id,¬,totalid] = unique(horzcat(elem{:})');
totalid = id(totalid);
for i = 1:length(nv2)
    v1 = nv1(i); v2 = nv2(i);
    totalid(totalid==v2) = v1;
    nv1(nv1==v2) = v1;  nv2(nv2==v2) = v1;
end
elemLen = cellfun('length',elem);
elem = mat2cell(totalid', 1, elemLen)';

for iel = 1:NT
    index = elem{iel};
    [¬,i1] = unique(index);
    elem{iel} = index(sort(i1));
end
[node,elem] = PolyMesher_Reorder(NT,node,elem);
```

References

[1] L. Chen. iFEM: an integrated finite element method package in MATLAB. Technical report, University of California at Irvine, 2009.

[2] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes. Polymesher: a general-purpose mesh generator for polygonal elements written in Matlab. *Struct. Multidiscip. Optim.*, 45(3):309–328, 2012.