# 0 最小二乘法拟合空间圆柱的算法

在三维空间中拟合圆柱的目标是找到一条**轴线**（方向和位置）以及一个**半径R**，使得点云到圆柱面的**距离误差平方和最小**。由于圆柱的参数非线性关联，通常采用**非线性最小二乘法**。下面详细介绍圆柱拟合的**数学模型、算法步骤**及其**MATLAB实现**。

## 圆柱的数学模型

设点云为，圆柱的轴线上某一点为，轴线方向为单位向量，圆柱的半径为。则每个点到圆柱轴线的垂直距离为:

我们希望最小化以下**残差平方和**：

其中是第个点到圆柱轴线的垂直距离。

## 算法步骤

1. **初始化参数**：
   * 圆柱的轴线方向向量：随机初始化，并单位化。
   * 轴线上任意一点：初始化为点云的均值，。
   * 半径：初步估计或随机初始化。
2. **构建目标函数**：
   * 根据所有点计算到圆柱的距离，并最小化误差平方和。
3. **非线性最小二乘优化**：
   * 使用优化算法（如Levenberg-Marquardt或MATLAB的lsqnonlin）求解参数。
4. **输出结果**：
   * 得到最佳拟合的圆柱参数：轴线上一点、轴线方向、半径。

## MATLAB实现代码

function[p0,d,R]=fitCylinderLSQ(points)

%拟合空间圆柱的最小二乘法

%输入:points-nx3点云数据

%输出:p0-轴线上一点,d-轴线方向,R-半径

%初始化参数

p0=mean(points);%初始化轴线通过点云的中心

d=randn(3,1);%随机轴线方向

d=d/norm(d);%单位化方向向量

R=1;%初始化半径

%将参数打包为向量

params0=[p0;d;R];

%优化选项

options=optimoptions('lsqnonlin','Display','iter','TolFun',1e-6);

%非线性最小二乘优化

params\_opt=lsqnonlin(@(params)residuals(params,points),params0,[],[],options);

%解包优化结果

p0=params\_opt(1:3);

d=params\_opt(4:6);

R=params\_opt(7);

d=d/norm(d);%确保方向向量归一化

end

functionres=residuals(params,points)

%计算残差:每个点到圆柱的距离误差

p0=params(1:3);%轴线上的一点

d=params(4:6);%轴线方向

R=params(7);%圆柱半径

d=d/norm(d);%确保单位化

%计算每个点的垂直距离误差

n=size(points,1);

res=zeros(n,1);

fori=1:n

p=points(i,:)';

proj=dot(p-p0,d)\*d;%点到轴线的投影

dist=norm((p-p0)-proj);%点到轴线的垂直距离

res(i)=dist-R;%残差:实际距离-半径

end

end

# 0 非线性最小二乘法拟合空间圆柱的算法

## 圆柱方程及模型特性

在空间圆柱拟合中，我们需要求解**轴线方向**、**圆柱上一点**、以及**半径**等参数。由于圆柱的轴线和这些点的关系一般是**非线性**的，所以在不同情况下可以使用**线性最小二乘法**或**非线性最小二乘法**。

## 线性最小二乘法在圆柱拟合中的应用

* + 线性最小二乘法可以用于近似拟合，比如当已知轴线方向或简化问题时。
  + 将拟合转化为求解垂直于轴线的投影距离，使得模型呈现出线性形式。
  + 问题简化：
  + 如果轴线方向已知，或者通过某种方法（例如PCA）提前确定了方向向量，则只需要求解轴线上一点和半径，这样问题变为线性形式：
  + 缺点：只能解决部分简化问题，不能处理完全未知的轴线方向。

## 非线性最小二乘法在圆柱拟合中的应用

非线性最小二乘法适用于**完全未知轴线方向**的情况。因为**轴线方向与点到轴线的距离是非线性关系**，因此需要通过数值迭代方法（如Levenberg-Marquardt算法）进行优化。

|  |  |  |
| --- | --- | --- |
| 比较项 | 线性最小二乘法 | 非线性最小二乘法 |
| 模型复杂度 | 简单线性模型或已知轴线方向 | 复杂非线性模型，完全未知轴线方向 |
| 计算复杂度 | 快速，解析解 | 需要迭代求解，计算复杂度高 |
| 是否需要初值 | 不需要 | 需要良好的初始值 |
| 适用场景 | 简化问题(如已知轴线方向) | 任意方向圆柱的拟合 |
| 优缺点 | 快速但不适用复杂场景 | 精确但计算耗时 |

在实际应用中，如果**轴线方向不明确且需要精确拟合**，则推荐使用**非线性最小二乘法**。如果可以先简化问题（如使用PCA预估轴线），则**线性最小二乘法**可以作为一种快速替代方案。

## MATLAB实现代码

function[p0,d,R]=fitCylinderNonlinear(points)

%使用非线性最小二乘法拟合空间圆柱

%输入：points-nx3的点云数据

%输出：p0-轴上一点,d-轴线方向,R-圆柱半径

%1.初始化参数

p0=mean(points);%轴线初始点：点云中心

d=randn(3,1);%随机初始化方向向量

d=d/norm(d);%单位化方向向量

R=1;%初始半径估计

%将参数打包为优化向量

params0=[p0;d;R];

%2.优化选项设置

options=optimoptions('lsqnonlin','Display','iter','TolFun',1e-6);

%3.执行非线性最小二乘优化

params\_opt=lsqnonlin(@(params)cylinderResiduals(params,points),...

params0,[],[],options);

%4.解包优化结果

p0=params\_opt(1:3);%轴上一点

d=params\_opt(4:6);%轴线方向

R=params\_opt(7);%半径

d=d/norm(d);%确保方向向量单位化

end

functionresiduals=cylinderResiduals(params,points)

%计算残差函数

p0=params(1:3);%轴线上一点

d=params(4:6);%轴线方向向量

R=params(7);%圆柱半径

d=d/norm(d);%单位化方向向量

%计算每个点的残差

n=size(points,1);

residuals=zeros(n,1);

fori=1:n

p=points(i,:)';%第i个点

proj=dot(p-p0,d)\*d;%点到轴线的投影

dist=norm((p-p0)-proj);%点到轴线的垂直距离

residuals(i)=dist-R;%计算残差

end

end

# 0 RANSAC（随机采样一致性）圆柱拟合算法

RANSAC是一种用于处理含噪声数据和**离群点**的鲁棒拟合算法。它通过随机采样子集、多次迭代找出模型参数，使大多数数据点与模型一致。对于**圆柱拟合**，RANSAC能有效应对数据集中的噪声和异常点。

## 基本思路

* + 随机采样：从点云中随机选择少量点来假设一个初步圆柱模型。
  + 模型估计：利用采样点估计圆柱的参数（如轴线方向、轴线上一点、半径）。
  + 内点集合判断：计算所有数据点与该模型的残差，判断哪些点与模型一致（称为“内点”）。
  + 模型优化：在找到最大内点集合后，对这些内点进行精确的参数拟合（如用最小二乘法）。
  + 重复迭代：不断重复上述步骤，最终选择内点最多的模型作为最佳拟合结果。

## 算法步骤

输入：

点云数据：

误差阈值：

最大迭代次数：

输出：

轴线方向为单位向量

圆柱的轴线上某一点为

圆柱的半径为

步骤：

1.初始化：

* 最大内点数。
* 最优模型参数(初始化为空。

2.循环N次：

* 随机采样：从点云中随机选择3个点，假设它们位于圆柱表面。
* 拟合初始模型：
* 根据3个点，计算圆柱的轴线方向和半径。
* 设3个点为,它们的法向量可以通过叉积求出
* 计算轴线上一点（可设为3点的中心）。
* 计算半径为点到轴线的垂直距离。
* 计算内点：

则记为内点

3.精确拟合：

* 使用找到的内点集合，用最小二乘法进一步优化模型参数。

## MATLAB实现代码

function[p0,d,R,inliers]=fitCylinderRANSAC(points,threshold,maxIter)

%RANSAC圆柱拟合算法

%输入：

%points-nx3的点云数据

%threshold-距离阈值，判断内点的误差范围

%maxIter-最大迭代次数

%输出：

%p0-轴上一点

%d-轴线方向

%R-半径

%inliers-内点集合的索引

numPoints=size(points,1);

maxInliers=0;%初始化最大内点数

%RANSAC迭代

foriter=1:maxIter

%随机选择3个点

sampleIdx=randperm(numPoints,3);

p1=points(sampleIdx(1),:)';

p2=points(sampleIdx(2),:)';

p3=points(sampleIdx(3),:)';

%计算轴线方向

d=cross(p2-p1,p3-p1);

d=d/norm(d);%单位化

%计算轴上一点（3个点的中心）

p0=mean([p1,p2,p3],2);

%计算半径（3个点的平均距离）

R=mean(vecnorm(cross(repmat(d,1,3),[p1-p0,p2-p0,p3-p0])));

%计算所有点的内点

inlierIdx=[];

fori=1:numPoints

p=points(i,:)';

dist=norm((p-p0)-dot(p-p0,d)\*d);%垂直距离

ifabs(dist-R)<threshold

inlierIdx=[inlierIdx;i];%记录内点索引

end

end

%更新最大内点集合

iflength(inlierIdx)>maxInliers

maxInliers=length(inlierIdx);

bestP0=p0;

bestD=d;

bestR=R;

inliers=inlierIdx;

end

end

%返回最佳模型参数

p0=bestP0;

d=bestD;

R=bestR;

end

# 1 生成圆柱面测点算法

## 算法流程

**1 计算法向量起点与终点坐标**

* 根据输入的方位角（azimuth）和仰角（elevation），计算单位球面上的某点坐标(x0,y0,z0)。

**2 计算垂足点**

* 将输入点P1和P2的垂足投影到通过(0,0,0)和(x0,y0,z0)的直线上，得到垂足点(xN1,yN1,zN1)和(xN2,yN2,zN2)。

**3 计算层间插值偏移量**

* 根据给定的层数laynum，按比例计算每层的插值偏移量deltx,delty,deltz，用于控制生成的圆的位置。

**4 计算法向量并归一化**

* 定义法向量为(x0,y0,z0)。如果法向量的长度不是1，则将其归一化。

**5 计算与法向量垂直的单位向量**

* 通过叉积找到一个与法向量垂直的向量v。如果法向量与z轴平行，则选择另一个垂直向量。
* 将v归一化，并通过叉积计算出第二个垂直向量u。

**6 生成单位圆的角度范围**

* 如果向量v的第一个分量大于0，则角度范围为-2pi/3到2pi/3；否则为pi/3到5pi/3。

**7 生成单位圆上的点**

* 使用参数方程计算圆上的点，得到每个点的x,y,z坐标。

**应用层间插值偏移并存储结果**

* 对于每一层，应用对应的插值偏移量deltx,delty,deltz，并将生成的圆点存储在输出矩阵Point\_out中。

## MATLAB实现代码

r = 1;

x0 = r.\*cos(azimuth).\*cos(elevation);

y0 = r.\*cos(azimuth).\*sin(elevation);

z0 = r.\*sin(azimuth);

Pl1 = [0,0,0];

Pl2 = [x0,y0,z0];

[xN1,yN1,zN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(P1,Pl1,Pl2);

[xN2,yN2,zN2] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(P2,Pl1,Pl2);

deltx = zeros(1,laynum);

delty = zeros(1,laynum);

deltz = zeros(1,laynum);

for i = 1:laynum

lamuda = i/(laynum+1-i);

deltx(i) = (xN1+lamuda\*xN2)/(1+lamuda);

delty(i) = (yN1+lamuda\*yN2)/(1+lamuda);

deltz(i) = (zN1+lamuda\*zN2)/(1+lamuda);

end

% 定义法向量

normal\_vector = [x0, y0, z0];

% 检查法向量是否是单位向量，如果不是则归一化

if norm(normal\_vector) ~= 1

normal\_vector = normal\_vector / norm(normal\_vector);

end

v = cross(normal\_vector, [0, 0, 1]); % 找到与法向量垂直的一个向量

if norm(v) < eps % 如果法向量与 [0, 0, 1] 平行，则选择另一个向量

v = cross(normal\_vector, [0, 1, 0]);

end

v = v / norm(v); % 归一化向量

u = cross(v, normal\_vector); % 创建另一个垂直向量

% 使用参数方程生成单位圆上的点

if v(1) > 0

theta = linspace(-2\*pi/3,2/3\*pi,num);

else

theta = linspace(pi/3,5/3\*pi,num);

end

% 单位圆的参数方程

x\_circle = cos(theta) \* v(1) + sin(theta) \* u(1);

y\_circle = cos(theta) \* v(2) + sin(theta) \* u(2);

z\_circle = cos(theta) \* v(3) + sin(theta) \* u(3);

Point\_out = zeros(3,num\*laynum);

for j = 1:laynum

x\_circle1 = x\_circle + deltx(j);

y\_circle1 = y\_circle + delty(j);

z\_circle1 = z\_circle + deltz(j);

Point\_out(:,(j-1)\*num+1:j\*num) = [x\_circle1;y\_circle1;z\_circle1];

end

# 2 更为稳健的拟合方法

## 算法流程

**1 初始化变量和参数**

* 设置角度分割数和遍历的角度范围（0到2π），为180次遍历。
* 初始化最小误差为99999，并准备记录最优圆参数和旋转角度。

**2 遍历所有旋转方向，计算拟合误差**

* 将每对角度转换为球面坐标，生成一个单位向量。
* 计算该单位向量与Z轴之间的旋转角度和法向量，并构建旋转矩阵。
* 使用旋转矩阵将测量点云投影到新的坐标系，求出X-Y平面的投影。
* 对投影后的点拟合圆，并计算每个点到圆的误差，得到误差的均方根值。
* 如果当前误差小于已记录的最小误差，更新最优参数和旋转角度。

**3 根据最优旋转矩阵求解圆柱的参数**

* 使用最优角度生成单位向量，并重新计算旋转矩阵。
* 构建逆旋转矩阵，用于将拟合的圆心还原到原始坐标系中。

**4 计算圆柱的中心和轴线方向**

* 将最优拟合圆的圆心坐标通过逆旋转矩阵转换为原始坐标系的圆柱中心点。
* 圆柱的轴线方向即为最优旋转角度对应的单位向量。

**5 计算圆柱的上下底圆心**

* 根据圆柱的中心和轴线方向，定义上下底圆的线段。
* 利用P\_bound1和P\_bound2，计算测量点到该线段的垂足作为底圆的圆心。

**6 输出最终结果**

* 返回圆柱的中心点、轴线方向向量、半径、误差数组，以及上下底圆的圆心。

## MATLAB实现代码

% points = points';

num = 180;

sp = linspace(0,2\*pi,num);

sz = linspace(0,2\*pi,num);

OptErr = 99999;

OptAngle = zeros(1,2);

OptPara = zeros(1,4);

OptAllErr = zeros(size(points,2),1);

for i = 1:num

for j = 1:num

[x0,y0,z0] = sph2cart(sp(i),sz(j),1);

theta = atan2(norm(cross([0, 0, 1], [x0, y0, z0])), dot([0, 0, 1], [x0, y0, z0]));

v = cross([0, 0, 1], [x0, y0, z0]) / norm(cross([0, 0, 1], [x0, y0, z0]));

rot1 = myvrrotvec2mat([v, theta]);

P = points'\*rot1;

X = P(:,1);

Y = P(:,2);

[center, radius] = fitcircle([X'; Y']);

err = hypot(X-center(1), Y-center(2))-radius;

Merr = sqrt(mean(err.^2));

if Merr < OptErr

OptErr = Merr;

OptPara = [center',radius,Merr];

OptAllErr = err;

OptAngle = [sp(i),sz(j)];

end

end

end

[x1,y1,z1] = sph2cart(OptAngle(1),OptAngle(2),1);

theta = atan2(norm(cross([0, 0, 1], [x1, y1, z1])), dot([0, 0, 1], [x1, y1, z1]));

v = cross([0, 0, 1], [x1, y1, z1]) / norm(cross([0, 0, 1], [x1, y1, z1]));

rot1 = myvrrotvec2mat([v, theta]);

rot2 = pinv(rot1);

Mcenter = [OptPara(1),OptPara(2),0] \* rot2;

MTaon = [x1,y1,z1];

Mradial = OptPara(3);

Err\_every = OptAllErr';

Mu1 = Mcenter;

Mu2 = Mcenter + MTaon;

[xxN1,yyN1,zzN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(P\_bound1,Mu1,Mu2);

[xxN2,yyN2,zzN2] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(P\_bound2,Mu1,Mu2);

Bottom\_round\_center1 = [xxN1,yyN1,zzN1];

Bottom\_round\_center2 = [xxN2,yyN2,zzN2];

# 3 计算圆柱AB面上的点算法

## 算法流程

**1 数据初始化与平移**

* 将输入点Pin,UPP,和PAB转置，并计算相对于参考点Pin的平移量。
* 生成两个新的平移向量：UPPmove和PABmove。

**2 构建旋转矩阵1（将UPPmove对齐至z轴）**

* 计算UPPmove向量与z轴之间的夹角（theta1）。
* 计算旋转轴v1，并使用它创建旋转矩阵rot1。
* 使用该旋转矩阵将Tao向量旋转为TaoRot1。

**3 构建旋转矩阵2（将TaoRot1对齐至x轴）**

* 计算TaoRot1向量与x轴之间的夹角（theta2）。
* 计算旋转轴v2，并使用它生成旋转矩阵rot2。

**4 计算总旋转矩阵ROT**

* 将旋转矩阵rot1和rot2相乘，得到总的旋转矩阵ROT。
* 使用ROT将平移后的PABmove进行旋转，得到PABrot。

**5 计算A点的坐标增量**

* 基于旋转后的PABrot，计算A点的增量坐标：
  + x方向：b\*tan(phi)
  + y方向：b
  + z方向：Ti\*h/2
* 为x方向坐标添加偏移量toff。

**6 生成A点和B点的坐标**

* 使用增量后的坐标创建旋转后的点集PointsArot和PointsBrot。

**7 将点集旋转并平移回初始坐标系**

* 使用总旋转矩阵的逆矩阵，将PointsArot和PointsBrot旋转回原始坐标系，并添加平移量Pin。

**8 输出结果**

* 将旋转平移后的点集结果存储为PointTable\_A\_off和PointTable\_B\_off，作为最终输出。

## MATLAB实现代码

numShengLu = length(Ang)./2;

MTaon = MTaon';

Mcenter = Mcenter';

phi = pi/2-phi;

% 计算phi面的法向量D

A = [0,0,1];

B = MTaon;

C = cross(A,B);

C = C./norm(C);

D = B+tan(phi)\*C;

D = D./norm(D);

% 计算0度线向量E

E = cross(cross(B,C),D);

E = E./norm(E);

% 圆心点

[xN1,yN1,zN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(PAB,Mcenter,Mcenter+MTaon);

% 面法向量第二个点（与圆心点构成phi面的法向量）

Tao2D = [xN1,yN1,zN1] + D;

% 起始点 （测点半圆的中点）

QiShi = [xN1,yN1,zN1] + E;

% 构建旋转矩阵

theta = atan2(norm(cross([0, 0, 1], MTaon)), dot([0, 0, 1], MTaon));

v = cross([0, 0, 1],MTaon) / norm(cross([0, 0, 1], MTaon));

rot1 = myvrrotvec2mat([v, theta]);

Prot = zeros(3,3);

% 选择旋转点集合

Prot(1:3,1) = [xN1;yN1;zN1]; % 圆心点

Prot(1:3,2) = Tao2D'; % 面法向量第二个点

Prot(1:3,3) = QiShi'; % 起始点 （测点半圆的中点，0度线）

% 旋转至【0，0，1】的点集合

P2D = Prot'\*rot1;

P2DT = P2D - repmat([P2D(1,1),P2D(1,2),0],size(P2D,1),1);

% 旋转、平移后 法向量

Tao3 = P2DT(2,:) - P2DT(1,:);

% 法平面参数 aa,bb,cc,dd 过点 P2DT(1,:) ，法向量Tao3

aa = Tao3(1);

bb = Tao3(2);

cc = Tao3(3);

dd = -(aa\*P2DT(1,1)+bb\*P2DT(1,2)+cc\*P2DT(1,3));

% 起始角度

AngOring = atan(P2DT(3,2)/P2DT(3,1));

% !!!!!!!后面的点以这个为基础

% 第一步修正

Ang = Ang+roff./Mradial;

% 第二步处理

AngProcess = zeros(1,length(Ang));

AngProcess(1:numShengLu) = AngOring - Ang(1:numShengLu);

AngProcess(1+numShengLu:2\*numShengLu) = pi+AngOring+Ang(1+numShengLu:2\*numShengLu);

% 调用计算坐标

x = Mradial\*cos(AngProcess);

y = Mradial\*sin(AngProcess);

z = -(dd+aa\*x+bb\*y)./cc;

PointTable2DT\_A = ones(length(Ang),3);

PointTable2DT\_A = [x',y',z'];

for iii = 1:size(PointTable2DT\_A,1)

if PointTable2DT\_A(iii,3)>P2DT(1,3)

PointTable2DT\_A(iii,3) = PointTable2DT\_A(iii,3)-toff(iii);

else

PointTable2DT\_A(iii,3) = PointTable2DT\_A(iii,3)+toff(iii);

end

end

% 2D A面点

PointTable2DT\_A = [PointTable2DT\_A(:,2),-PointTable2DT\_A(:,1),PointTable2DT\_A(:,3)];

%2D B面点

Zcen2D = P2DT(1,3);

tempBZ = 2\*Zcen2D - PointTable2DT\_A(:,3);

PointTable2DT\_B = [PointTable2DT\_A(:,1),PointTable2DT\_A(:,2),tempBZ];

%2D A面测点 转3D

PointTable\_A1 = (PointTable2DT\_A + repmat([P2D(1,1),P2D(1,2),0],size(PointTable2DT\_A,1),1))\*pinv(rot1);

PointTable\_B1 = (PointTable2DT\_B + repmat([P2D(1,1),P2D(1,2),0],size(PointTable2DT\_B,1),1))\*pinv(rot1);

PointTable\_A\_off = zeros(3,length(Ang));

PointTable\_A\_off = PointTable\_A1';

PointTable\_B\_off = zeros(3,length(Ang));

PointTable\_B\_off = PointTable\_B1';

# 4 圆柱面复测算法

## 功能描述

核心目的是分析测量点对的几何关系，计算距离、角度、相对高度、以及两个权重集，并通过这些结果优化几何结构或模型的性能。以下是功能与实现流程的详细解析。

## 算法流程

**1 计算测量点对之间的距离**

功能

计算每对测量点之间的距离，以了解它们在空间中的间隔。

实现思路

遍历所有测量点，每对点的距离由两点之间的欧氏距离决定。

对于点对 (A, B)，计算出它们之间的距离并存储在结果数组中。

**2 计算点对连线与圆柱轴线之间的夹角**

功能

* 计算每对测量点的连线与圆柱轴线之间的夹角，以分析其在空间中的方向关系。

实现思路

* 对于每对点，计算它们之间的连线方向向量。
* 使用该向量与圆柱轴向量的夹角公式，判断两者的方向差异。
* 如果角度大于90度（π/2），将其调整到 [0, π/2] 范围内。

**3 计算点对的相对高度**

功能

* 计算每对测量点的中点到圆柱轴线的垂直距离，并转换为相对高度（归一化处理）。

实现思路

* 对每对测量点，求得它们的中点。
* 将该中点投影到圆柱轴线上，得到中点与轴线之间的垂直距离。
* 使用圆柱的半径进行归一化，将距离转换为相对高度。

**4 计算 LT 参数**

功能

* 计算一个描述点对几何关系的参数 LT，该参数基于夹角和点对距离。

实现思路

* 对于每对点，根据其夹角和距离计算 LT。
* 该参数反映了点对在不同方向和位置上的综合几何关系。

**5 计算权重 1 和权重 2**

功能

* 分别使用不同的权重计算方法，给每对点分配两种权重，衡量它们在结构分析中的重要性。

实现思路

* 遍历每个测量点：对每个点依次排除一个，计算剩余点之间的组合关系。
* 计算组合的乘积和：使用 nchoosek 函数计算所有可能的组合，并对每个组合求乘积和。
* 归一化处理：根据参数和系数，将乘积和结果进行归一化。
* 存储两种权重：权重 1 使用第一组系数，权重 2 使用第二组系数，分别得到不同的结果。

## MATLAB实现代码

%% 距离计算

Distance = zeros(1,shenglunum);

for i = 1:shenglunum

Distance(i) = norm(PointIn(:,2\*i-1)-PointIn(:,2\*i));

end

%% 计算声道角度

% 计算水平方向向量

theta = zeros(1,shenglunum);

for i = 1:shenglunum

shengdao1 = PointIn(:,2\*i-1);

shengdao2 = PointIn(:,2\*i);

SPXL = shengdao2 - shengdao1;

theta(i) = acos(dot(MTaon, SPXL) / (norm(MTaon) \* norm(SPXL)));

end

theta(theta>pi./2) = pi- theta(theta>pi./2);

% rad2deg(theta)

%% 相对高度计算

TiC = zeros(1,shenglunum);

for i = 1:shenglunum

shengdao1 = PointIn(:,2\*i-1);

shengdao2 = PointIn(:,2\*i);

MidP = (shengdao1+shengdao2)./2;

Pt1 = Mcenter;

Pt2 = Mcenter+MTaon;

[xN1,yN1,zN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(MidP,Pt1,Pt2);

temp = [xN1;yN1;zN1];

TiC(i) = norm(MidP-temp)./Mradial;

end

alphaA = asin(TiC);

%% 计算LT

LTPY = zeros(1,shenglunum);

for i = 1:shenglunum

aa = Mradial.\*cos(alphaA(i))./cos(theta(i));

bb = Distance(i)./2.;

LTPY(i) = aa-bb;

end

%% 计算权重 1

k1 = 0.5;

k2 = 0.6;

k3 = 0;

k4 = 0.15;

gk1 = [1.570796,0.392699,0.19635,0.122718,0.085903];

gk2 = [1.513365,0.360325,0.174351,0.106311,0.072959];

gk3 = [2,2/3,2/5,2/7,2/9];

gk4 = [1.838286,0.556753,0.315143,0.215852,0.162469];

gk = gk1;

k = k1;

TiYiCe = TiC;

if mod(shenglunum,2)==1 %奇数

temp6 = floor(shenglunum./2);

TiYiCe(temp6+1) = 0;

end

t\_k = zeros(shenglunum-1,1);

w = zeros(shenglunum,1);

f = zeros(shenglunum,1);

for i=1:shenglunum

count\_tk = 1;

for m=1:shenglunum

if(m~=i)

t\_k(count\_tk) = TiYiCe(m);

count\_tk = count\_tk + 1;

end

end

for j=1:(shenglunum+1)/2

f(j) = -sum(prod(nchoosek(t\_k,(shenglunum+1-2\*j)),2));

w(i) = w(i) + gk(j)\*f(j);

end

mul = 1;

for m=1:shenglunum

if(m ~= i)

mul = mul\*(TiYiCe(i) - TiYiCe(m));

end

end

w(i)=w(i)/((1-TiYiCe(i)\*TiYiCe(i))^k \* mul);

end

Wquanzhong1 = abs(repelem(w,2));

%% 计算权重 2

k1 = 0.5;

k2 = 0.6;

k3 = 0;

k4 = 0.15;

gk1 = [1.570796,0.392699,0.19635,0.122718,0.085903];

gk2 = [1.513365,0.360325,0.174351,0.106311,0.072959];

gk3 = [2,2/3,2/5,2/7,2/9];

gk4 = [1.838286,0.556753,0.315143,0.215852,0.162469];

gk = gk2;

k = k2;

TiYiCe2 = TiC;

if mod(shenglunum,2)==1 %奇数

temp6 = floor(shenglunum./2);

TiYiCe2(temp6+1) = 0;

end

t\_k = zeros(shenglunum-1,1);

w = zeros(shenglunum,1);

f = zeros(shenglunum,1);

for i=1:shenglunum

count\_tk = 1;

for m=1:shenglunum

if(m~=i)

t\_k(count\_tk) = TiYiCe2(m);

count\_tk = count\_tk + 1;

end

end

for j=1:(shenglunum+1)/2

f(j) = -sum(prod(nchoosek(t\_k,(shenglunum+1-2\*j)),2));

w(i) = w(i) + gk(j)\*f(j);

end

mul = 1;

for m=1:shenglunum

if(m ~= i)

mul = mul\*(TiYiCe2(i) - TiYiCe2(m));

end

end

w(i)=w(i)/((1-TiYiCe2(i)\*TiYiCe2(i))^k \* mul);

end

Wquanzhong2 = abs(repelem(w,2));

# 5 四面管路的平面拟合算法

## 核心目的

1. **拟合四个平面**，每个平面由输入的3D点拟合而成。
2. **计算相邻平面之间的交线**，并通过这些交线计算边界附近的三角形点。
3. **判断平面是否接近平行于z轴**，并选择不同的拟合策略。
4. **输出平面参数和三角形点的投影结果**。

## 算法流程及实现

**1. 初始化变量**

PointAll = {Points1, Points2, Points3, Points4};  
PlaneParaOut = zeros(4, 4);

* 将四组点存入PointAll，用于循环遍历。
* 初始化PlaneParaOut，存储四个平面的参数（每列表示一个平面：）。

**2. RANSAC算法拟合平面**

对于每组点，使用\*\*随机采样一致性算法（RANSAC）\*\*拟合平面：

1. **从点集中任选3个点**来构建平面。
2. **计算平面方程**：
   * 假设平面方程为 。
   * 根据3个点的坐标 构建如下矩阵：
   * 求解：
   * 结果为：
3. **计算所有点到平面的距离**：
   * 对于任意点，其到拟合平面的距离为：
   * 找到**所有距离小于**distanceThreshold的内点，更新最优平面模型。

**3. 根据内点重新拟合平面**

1. 如果找到的平面内点数量足够，用**所有内点重新计算平面**。
2. 使用**SVD分解**确定平面的法向量。
3. 计算**法向量与z轴的夹角**：
   * 夹角：
   * 如果夹角接近0°，说明平面接近平行于z轴。

**4. 不同条件下拟合平面模型**

* 如果平面平行于z轴，则使用polyfit拟合线性模型：转换为平面形式：
* 如果不平行于z轴，则使用所有内点重新计算平面系数。

**5. 计算平面的交线**

使用CrossLine函数计算**两个平面之间的交线**：

1. 假设两个平面的参数为 和 。
2. 交线的方向向量为这两个平面的法向量的叉乘：
3. 选择一个适当的值，求解对应的交线点。

**6. 生成三角形点**

调用GenerateTrianglePoints函数：

* 在交线上计算边界点（BoundPoint1和BoundPoint2）的投影点。

**7. 计算点到直线的垂足**

* 给定点到直线的垂足：

**8. 输出结果**

1. PlaneParaOut：存储四个平面的参数。
2. TrianglePoints：所有计算出的三角形点。
3. MaxDis：每个平面到其内点的最大距离。
4. distancesFianal：所有点到各自平面的距离。

## MATLAB实现代码

PointAll = {Points1,Points2,Points3,Points4};

coder.varsize('PlaneParaOut',[4 3000],[0 1]);

PlaneParaOut = zeros(4,4);

inlierIdxFinal = {1,2,3,4};

distancesFianal1 = {1,2,3,4};

for i =1:4

Points = PointAll{i};

% 平面拟合

pointss = Points';

Cnum = 1:length(pointss);

bestDist = 99999; % 最优内点距离

C = nchoosek(Cnum,3);

for j = 1:length(C)

% 随机选择三个点

sampleIdx = C(j,:);

samplePoints = pointss(sampleIdx, :);

% 计算平面模型

A = [samplePoints(:,1), samplePoints(:,2), ones(3,1)];

coefficients = A \ samplePoints(:,3);

a = coefficients(1);

b = coefficients(2);

d = coefficients(3);

c=-1;

distances=abs([a,b,c,d]\*[pointss,ones(size(pointss,1),1)]')/sqrt(a\*a+b\*b+c\*c);

% 确定内点

inlierIdx = find(distances < distanceThreshold);

% 更新最优平面模型

if mean(distances) < bestDist

bestDist = mean(distances);

inlierIdxFinal{i} = inlierIdx;

distancesFianal1{i} = distances;

end

end

% 重新计算最终平面模型，使用所有内点

inlierPoints = pointss(inlierIdxFinal{i}, :);

x = inlierPoints(:,1);

y = inlierPoints(:,2);

z = inlierPoints(:,3);

PIner = [x,y,z];

threshold = 0.5;

% Fit a plane through the points

[~, ~, V] = svd(PIner - mean(PIner, 1));

normal = V(:, 3); % Normal vector of the plane

% Calculate angle between normal vector and z-axis

angle = acosd(abs(dot(normal, [0; 0; 1])));

angle = abs(90-angle);

% Check if angle is below threshold

if angle < threshold

c = 0;

bb=polyfit(x,y,1);

% 拟合，其实是线性回归，但可以用来拟合平面

a = bb(1);

b = -1;

d = bb(2);

else

A = [inlierPoints(:,1), inlierPoints(:,2), ones(size(inlierPoints(:,1)))];

coefficients = A \ inlierPoints(:,3);

a = coefficients(1);

b = coefficients(2);

d = coefficients(3);

c=-1;

end

PlaneParaOut(:,i) = [a;b;c;d];

end

id1 = inlierIdxFinal{1};

id2 = inlierIdxFinal{2};

id3 = inlierIdxFinal{3};

id4 = inlierIdxFinal{4};

tt1 = distancesFianal1{1};

tt2 = distancesFianal1{2};

tt3 = distancesFianal1{3};

tt4 = distancesFianal1{4};

T1 = max(tt1(id1));

T2 = max(tt2(id2));

T3 = max(tt3(id3));

T4 = max(tt4(id4));

MaxDis = [T1,T2,T3,T4];

distancesFianal = [distancesFianal1{1},distancesFianal1{2},distancesFianal1{3},distancesFianal1{4}];

% 平面方程的系数输出

xfit = zeros(1,8);

yfit = zeros(1,8);

zfit = zeros(1,8);

%%%%%% 前2个面的交点 %%%%

% 计算交线

x\_val = (max(x)+min(x))./2;

[P0,D] = CrossLine(PlaneParaOut(:,1),PlaneParaOut(:,2),x\_val);

% 找到边界，确定三角点

[PointTri] = GenerateTrianglePoints(PlaneParaOut(:,1),BoundPoint1,P0,D);

xfit(1) = PointTri(1,1);

yfit(1) = PointTri(2,1);

zfit(1) = PointTri(3,1);

[PointTri2] = GenerateTrianglePoints(PlaneParaOut(:,1),BoundPoint2,P0,D);

xfit(3) = PointTri2(1,1);

yfit(3) = PointTri2(2,1);

zfit(3) = PointTri2(3,1);

%%%%%% 第2、3个面的交点 %%%%

[P0,D] = CrossLine(PlaneParaOut(:,2),PlaneParaOut(:,3),x\_val);

% 找到边界，确定三角点

[PointTri3] = GenerateTrianglePoints(PlaneParaOut(:,3),BoundPoint1,P0,D);

xfit(5) = PointTri3(1,1);

yfit(5) = PointTri3(2,1);

zfit(5) = PointTri3(3,1);

[PointTri4] = GenerateTrianglePoints(PlaneParaOut(:,3),BoundPoint2,P0,D);

xfit(6) = PointTri4(1,1);

yfit(6) = PointTri4(2,1);

zfit(6) = PointTri4(3,1);

%%%%%% 第3、4个面的交点 %%%%

[P0,D] = CrossLine(PlaneParaOut(:,3),PlaneParaOut(:,4),x\_val);

[PointTri5] = GenerateTrianglePoints(PlaneParaOut(:,4),BoundPoint1,P0,D);

xfit(7) = PointTri5(1,1);

yfit(7) = PointTri5(2,1);

zfit(7) = PointTri5(3,1);

[PointTri6] = GenerateTrianglePoints(PlaneParaOut(:,4),BoundPoint2,P0,D);

xfit(8) = PointTri6(1,1);

yfit(8) = PointTri6(2,1);

zfit(8) = PointTri6(3,1);

%%%%%% 第1、4个面的交点 %%%%

[P0,D] = CrossLine(PlaneParaOut(:,1),PlaneParaOut(:,4),x\_val);

[PointTri7] = GenerateTrianglePoints(PlaneParaOut(:,4),BoundPoint1,P0,D);

xfit(2) = PointTri7(1,1);

yfit(2) = PointTri7(2,1);

zfit(2) = PointTri7(3,1);

[PointTri8] = GenerateTrianglePoints(PlaneParaOut(:,4),BoundPoint2,P0,D);

xfit(4) = PointTri8(1,1);

yfit(4) = PointTri8(2,1);

zfit(4) = PointTri8(3,1);

BJD = [xfit([1:3,2:4,[1,3,5],[3,5,6],5:7,6:8,[2,4,7],[4,7,8]]);...

yfit([1:3,2:4,[1,3,5],[3,5,6],5:7,6:8,[2,4,7],[4,7,8]]);...

zfit([1:3,2:4,[1,3,5],[3,5,6],5:7,6:8,[2,4,7],[4,7,8]])];

% 取八个点

Pdd = BJD(:,[1:3,6,9,16:18]);

P1 = Pdd(:,1);

P2 = Pdd(:,2);

P3 = Pdd(:,3);

P4 = Pdd(:,4);

P5 = Pdd(:,5);

P6 = Pdd(:,6);

P7 = Pdd(:,7);

P8 = Pdd(:,8);

% 计算投影点

[xN1,yN1,zN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint1,P1,P3);

PP1 = [xN1;yN1;zN1];

[xN2,yN2,zN2] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint1,P2,P4);

PP2 = [xN2;yN2;zN2];

[xN3,yN3,zN3] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint2,P1,P3);

PP3 = [xN3;yN3;zN3];

[xN4,yN4,zN4] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint2,P2,P4);

PP4 = [xN4;yN4;zN4];

[xN5,yN5,zN5] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint1,P5,P6);

PP5 = [xN5;yN5;zN5];

[xN6,yN6,zN6] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint2,P5,P6);

PP6 = [xN6;yN6;zN6];

[xN7,yN7,zN7] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint1,P7,P8);

PP7 = [xN7;yN7;zN7];

[xN8,yN8,zN8] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(BoundPoint2,P7,P8);

PP8 = [xN8;yN8;zN8];

TrianglePoints = [PP1,PP2,PP3,PP2,PP3,PP4,PP1,PP3,PP5,PP3,PP5,PP6,PP5,PP6,PP7,PP6,PP7,PP8,PP2,PP4,PP7,PP4,PP7,PP8];

coder.varsize('TrianglePoints',[3 3000],[0 1]);

# 6 八面管路拟合算法

## 功能描述

该函数 planefit8 主要实现了以下功能：

1. **拟合 8 个点云的平面模型**：每组点云使用随机采样算法选择三个点拟合平面，并根据距离阈值筛选内点。
2. **计算每组平面的法向量**：用于判断平面方向与 Z 轴的夹角，进而选择合适的拟合模型。
3. **计算相邻平面之间的交线方向**：使用叉乘法向量，并确保方向一致性。
4. **计算顶部和底部的 8 个交点**：通过求解相邻侧面和顶/底面之间的方程组得到交点。
5. **输出每组平面的最大点距及顶面/底面三角形点序列**。

## 关键算法与公式说明

**2.1 平面拟合**

给定 3 个点 ，平面方程为：

使用最小二乘法拟合平面时，系数通过求解下式获得：

其中：

求解：

**2.2 点到平面的距离公式**

对于给定点 到平面 的距离：

**2.3 叉乘计算相邻平面的交线方向**

对于两个平面法向量 和 ，其交线方向为：

展开得：

方向向量归一化：

**2.4 顶部和底部面方程的计算**

设平面法向量为 ，与边界点 的点积为零：

**2.5 顶部和底部顶点求解**

对于相邻的两个平面 和 ，与顶部面：

解得：

**2.6 三角形点序列的生成**

使用交点生成的矩阵 生成顶面和底面三角形的顶点序列。

## 输出说明

* **PlaneParaOut**：8 个平面的参数矩阵 。
* **TrianglePoints**：三角形顶点序列，用于绘制顶面和底面。
* **MaxDis**：每组平面内的最大点到平面距离。
* **distancesFianal**：所有点到平面的距离结果。

## MATLAB实现代码

PointAll = {Points1,Points2,Points3,Points4,Points5,Points6,Points7,Points8};

coder.varsize('PlaneParaOut',[4 3000],[0 1]);

PlaneParaOut = zeros(4,8);

inlierIdxFinal = {1,2,3,4,5,6,7,8};

distancesFianal1 = {1,2,3,4,5,6,7,8};

for i =1:8

Points = PointAll{i};

% 平面拟合

pointss = Points';

Cnum = 1:length(pointss);

bestDist = 99999; % 最优内点距离

C = nchoosek(Cnum,3);

for j = 1:length(C)

% 随机选择三个点

sampleIdx = C(j,:);

samplePoints = pointss(sampleIdx, :);

% 计算平面模型

A = [samplePoints(:,1), samplePoints(:,2), ones(3,1)];

coefficients = A \ samplePoints(:,3);

a = coefficients(1);

b = coefficients(2);

d = coefficients(3);

c=-1;

distances=abs([a,b,c,d]\*[pointss,ones(size(pointss,1),1)]')/sqrt(a\*a+b\*b+c\*c);

% 确定内点

inlierIdx = find(distances < distanceThreshold);

% 更新最优平面模型

if mean(distances) < bestDist

bestDist = mean(distances);

inlierIdxFinal{i} = inlierIdx;

distancesFianal1{i} = distances;

end

end

% 重新计算最终平面模型，使用所有内点

inlierPoints = pointss(inlierIdxFinal{i}, :);

x = inlierPoints(:,1);

y = inlierPoints(:,2);

z = inlierPoints(:,3);

PIner = [x,y,z];

threshold = 0.5;

% Fit a plane through the points

[~, ~, V] = svd(PIner - mean(PIner, 1));

normal = V(:, 3); % Normal vector of the plane

% Calculate angle between normal vector and z-axis

angle = acosd(abs(dot(normal, [0; 0; 1])));

angle = abs(90-angle);

% Check if angle is below threshold

if angle < threshold

c = 0;

bb=polyfit(x,y,1);

% 拟合，其实是线性回归，但可以用来拟合平面

a = bb(1);

b = -1;

d = bb(2);

else

A = [inlierPoints(:,1), inlierPoints(:,2), ones(size(inlierPoints(:,1)))];

coefficients = A \ inlierPoints(:,3);

a = coefficients(1);

b = coefficients(2);

d = coefficients(3);

c=-1;

end

PlaneParaOut(:,i) = [a;b;c;d];

end

id1 = inlierIdxFinal{1};

id2 = inlierIdxFinal{2};

id3 = inlierIdxFinal{3};

id4 = inlierIdxFinal{4};

id5 = inlierIdxFinal{5};

id6 = inlierIdxFinal{6};

id7 = inlierIdxFinal{7};

id8 = inlierIdxFinal{8};

tt1 = distancesFianal1{1};

tt2 = distancesFianal1{2};

tt3 = distancesFianal1{3};

tt4 = distancesFianal1{4};

tt5 = distancesFianal1{5};

tt6 = distancesFianal1{6};

tt7 = distancesFianal1{7};

tt8 = distancesFianal1{8};

T1 = max(tt1(id1));

T2 = max(tt2(id2));

T3 = max(tt3(id3));

T4 = max(tt4(id4));

T5 = max(tt5(id5));

T6 = max(tt6(id6));

T7 = max(tt7(id7));

T8 = max(tt8(id8));

MaxDis = [T1,T2,T3,T4,T5,T6,T7,T8];

distancesFianal = [tt1,tt2,tt3,tt4,tt5,tt6,tt7,tt8];

PlaneParaOutP = PlaneParaOut';

% 初始化一个矩阵存储叉乘结果

cross\_vectors = zeros(8, 3);

% 计算每对相邻侧面法向量的叉乘

for i = 1:8

if i < 8

cross\_vectors(i, :) = cross(PlaneParaOutP(i, 1:3), PlaneParaOutP(i+1, 1:3));

else

cross\_vectors(i, :) = cross(PlaneParaOutP(i, 1:3), PlaneParaOutP(1, 1:3)); % 最后一个与第一个相邻

end

cross\_vectors(i, :) = cross\_vectors(i, :) / norm(cross\_vectors(i, :)); % 归一化

end

% 使所有方向一致

reference\_vector = cross\_vectors(1, :); % 选择第一个向量作为参考

for i = 2:8

if dot(reference\_vector, cross\_vectors(i, :)) < 0 % 如果方向相反

cross\_vectors(i, :) = -cross\_vectors(i, :); % 翻转方向

end

end

% 计算平均方向向量，并归一化

n = mean(cross\_vectors, 1);

% 计算顶面方程的 d 值

d\_top = -dot(n, P\_bound1);

% 计算底面方程的 d 值

d\_bottom = -dot(n, P\_bound2);

% 初始化顶点矩阵

PP = zeros(16, 3);

% 计算顶面和底面的8个顶点

for i = 1:8

% 顶面顶点

if i < 8

PlaneParaOut1 = PlaneParaOutP(i, :);

PlaneParaOut2 = PlaneParaOutP(i + 1, :);

else

PlaneParaOut1 = PlaneParaOutP(i, :);

PlaneParaOut2 = PlaneParaOutP(1, :);

end

% 求顶面和两个相邻侧面的交线

A = [PlaneParaOut1(1:3); PlaneParaOut2(1:3); n];

B\_top = [-PlaneParaOut1(4); -PlaneParaOut2(4); -d\_top];

PP(i, :) = A\B\_top;

% 求底面和两个相邻侧面的交线

B\_bottom = [-PlaneParaOut1(4); -PlaneParaOut2(4); -d\_bottom];

PP(i + 8, :) = A\B\_bottom;

end

PP = PP';

TrianglePoints = PP(:,[1,8,9,8,9,16,1,2,10,1,9,10,2,3,10,3,10,11,3,4,11,4,11,12,4,5,12,5,12,13,5,6,13,6,13,14,6,7,14,7,14,15,7,8,15,8,15,16]);

coder.varsize('TrianglePoints',[3 3000],[0 1]);

# 7 四面管路AB面点计算

## 功能概述

该函数的主要功能是对矩形A点集和B点集进行平移和旋转变换，并计算出它们在偏移后的位置。其核心目的是通过一系列几何变换（包括平移、绕轴旋转等）得到相应的新坐标，用于生成经过平移和偏移的矩形点集。

## 算法实现步骤

**1. 输入参数**

* Tao：方向向量，用于对齐的旋转轴。
* UPP：初始参考点，表示将其旋转对齐到z轴的目标。
* Pin：原点的平移基准点。
* b：矩形的宽度。
* h：矩形高度相关的增量系数。
* phi：旋转角度，用于矩形倾斜角度计算。
* shenglunum：矩形块数量。
* Ti：当前块在z方向上的索引。
* toff：偏移量，用于将矩形点在x方向进行平移。

**2. 平移与旋转矩阵计算**

**2.1. 平移操作**

所有点 (UPP 和 PAB) 通过向量 Pin 进行平移，得到新的位置：

**2.2. 旋转矩阵 1：将** UPP 移动到 z 轴

使用旋转矩阵将 UPP 对齐到 **z轴方向**。旋转角度 \theta\_1 通过 **叉积与点积**计算：

其中：

* ：z轴单位向量
* ：需要对齐的向量

得到法向量 ，计算旋转矩阵：

**2.3. 旋转矩阵 2：将 Tao 对齐到 x 轴**

同理，将旋转后的 Tao 向量与 **x轴**对齐：

**2.4. 总旋转矩阵**

最终的旋转矩阵为：

将 PAB 点按照总旋转矩阵旋转：

**3. A、B点的计算**

**3.1. 计算 A 点的增量坐标**

对于每个矩形块，根据公式计算A点的坐标：

* **x 方向增量**：
* **y 方向增量**：
* **z 方向增量**：

根据上述增量，计算出A点旋转后的坐标：

**3.2. 应用偏移量**

将A点的x方向坐标加入偏移量toff：

类似地，计算B点的坐标：

**4. 旋转与平移回原始坐标系**

将A、B点的旋转坐标变换回原始坐标系：

**5. 输出结果**

最终得到平移和旋转后的A、B点坐标矩阵：

## MATLAB实现代码

Pin = Pin';

UPP = UPP';

PAB = PAB';

% 平移 Pin

UPPmove = UPP-Pin;

PABmove = PAB-Pin;

% 构建旋转矩阵1,pt6转到z轴

theta1 = atan2(norm(cross([0, 0, 1], UPPmove)), dot([0, 0, 1], UPPmove));

v1 = cross([0, 0, 1],UPPmove) / norm(cross([0, 0, 1], UPPmove));

rot1 = myvrrotvec2mat([v1, theta1]);

% Tao旋转rot1

TaoRot1 = Tao'\*rot1;

% 构建旋转矩阵2,taorot1转到x轴

theta2 = atan2(norm(cross([1, 0, 0], TaoRot1)), dot([1, 0, 0], TaoRot1));

v2 = cross([1, 0, 0],TaoRot1) / norm(cross([1, 0, 0], TaoRot1));

rot2 = myvrrotvec2mat([v2, theta2]);

% 总的旋转矩阵 ROT

ROT = rot1\*rot2;

PABrot = PABmove\*ROT;

%% 计算A点

% 在PABrot的基础上增量xyz

% x为b\*tan(phi)

% y为b

% z为Ti\*h

xbase = PABrot(1);

Px = ones(1,shenglunum);

Ax = [b./2\*tan(phi).\*Px,-b./2\*tan(phi).\*Px]+xbase;

Ay = [b./2.\*Px,-b./2.\*Px];

Az = Ti.\*h./2;

%% 计算并偏移

Axoff = Ax + toff;

PointsArot = [Axoff;Ay;Az];

PointsBrot = [Axoff;-Ay;Az];

%% 旋转平移回去

PointsA = PointsArot'\*pinv(ROT)+ repmat(Pin,2\*shenglunum,1);

PointsB = PointsBrot'\*pinv(ROT)+ repmat(Pin,2\*shenglunum,1);

PointTable\_A\_off = PointsA';

PointTable\_B\_off = PointsB';

# 8 八面管路AB面点算法

## 功能

旨在根据八个平面顶点数据，计算矩形结构的偏移校正以及与目标点的交点。它处理多面体的面拟合、角度计算、阈值判断、点偏移修正等任务，最终输出偏移后的安装点和斜面偏移量。

## 算法实现及关键步骤

**输入参数说明**

* side\_faces\_transformed1~8：八个变换后的平面顶点矩阵，分别代表八个平面。
* P\_bound1、P\_bound2：矩形的边界点。
* PAB：目标点，用于计算交点。
* phi：倾角角度，用于计算偏移。
* shenglunum：待处理的点数量。
* Ti：待修正的初始偏移数据。
* a：每个点的初始偏移系数数组。
* distanceThreshold：距离阈值，用于拟合计算。

## 主要算法流程

**1. 八面体平面拟合**

使用planefit8函数对八个面进行拟合，获取平面参数及三角形顶点点集。

公式：

给定平面，平面参数为：

**2. 角度计算**

计算相邻平面之间的夹角。

公式：

其中，和分别是两个平面的法向量。

**3. 计算阈值**

通过计算两点间的垂线足点，判断点是否在斜面上。

公式：

1. 点到线的垂足：  
   对给定的点和线段端点、，垂足的计算公式为：

其中：

1. 偏移量计算：

**4. 判断点是否在斜面上**

根据计算出的阈值，判断点是否位于斜面上，并标记在XMFlag中。

**5. 斜面上点的偏移修正**

对于斜面上的点，修正其偏移量。

公式：

若点在斜面上，则：

否则：

斜面偏移量：

**6. 交点计算**

计算偏移后点与平面的交点。

公式：

已知点的方向向量为，求解与平面交点参数：

交点坐标为：

## MATLAB代码实现

%%%%%%%%%%%%%%%%%%%%%% 计算八面参数 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[PlaneParaOut8,TrianglePoints8,~,~] = planefit8(side\_faces\_transformed1,side\_faces\_transformed2,side\_faces\_transformed3,side\_faces\_transformed4,side\_faces\_transformed5,side\_faces\_transformed6,side\_faces\_transformed7,side\_faces\_transformed8,P\_bound1,P\_bound2,distanceThreshold);

%%%%%%%%%%%%%%%%%%%%%% 计算角度 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

PlanePara1 = PlaneParaOut8(1:3,1);

PlanePara2 = PlaneParaOut8(1:3,2);

PlanePara4 = PlaneParaOut8(1:3,4);

PlanePara5 = PlaneParaOut8(1:3,5);

PlanePara6 = PlaneParaOut8(1:3,6);

PlanePara8 = PlaneParaOut8(1:3,8);

theta1 = acos(dot(PlanePara1,PlanePara2)./norm(PlanePara1)./norm(PlanePara2));

theta2 = acos(dot(PlanePara4,PlanePara5)./norm(PlanePara4)./norm(PlanePara5));

theta3 = acos(dot(PlanePara5,PlanePara6)./norm(PlanePara5)./norm(PlanePara6));

theta4 = acos(dot(PlanePara1,PlanePara8)./norm(PlanePara1)./norm(PlanePara8));

%%%%%%%%%%%%%%%%%%%%%% 计算阈值 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 承接16顶点

PP16 = TrianglePoints8(:,[7,8,14,20,26,32,38,44,3,9,18,24,30,36,42,48]);

a1 = PP16(:,1);

b1 = PP16(:,4);

c1 = PP16(:,2);

[xN1,yN1,zN1] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(c1,a1,b1);

d1 = [xN1,yN1,zN1]';

ab1 = norm(a1-b1)./2;

TiYuZhi1 = (ab1-norm(a1-d1))./ab1;

a2 = PP16(:,8);

b2 = PP16(:,5);

c2 = PP16(:,7);

[xN2,yN2,zN2] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(c2,a2,b2);

d2 = [xN2,yN2,zN2]';

ab2 = norm(a2-b2)./2;

TiYuZhi2 = (ab2-norm(a2-d2))./ab2;

a3 = PP16(:,8);

b3 = PP16(:,5);

c3 = PP16(:,6);

[xN3,yN3,zN3] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(c3,a3,b3);

d3 = [xN3,yN3,zN3]';

ab3 = norm(a3-b3)./2;

TiYuZhi3 = (ab3-norm(a3-d3))./ab3;

a4 = PP16(:,1);

b4 = PP16(:,4);

c4 = PP16(:,3);

[xN4,yN4,zN4] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(c4,a4,b4);

d4 = [xN4,yN4,zN4]';

ab4 = norm(a4-b4)./2;

TiYuZhi4 = (ab4-norm(a4-d4))./ab4;

%%%%%%%%%%%%%%%%%%%%%% 判断点是否在斜面上（利用阈值） %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

temp = shenglunum./2;

XMFlag = zeros(1,2\*shenglunum);

for i = 1:2\*shenglunum

Tii = abs(Ti(i));

if i <= temp

if abs(Tii) < abs(TiYuZhi3)

XMFlag(i) = 0;

else

XMFlag(i) = 1;

end

elseif i <= 2\*temp

if abs(Tii) < abs(TiYuZhi2)

XMFlag(i) = 0;

else

XMFlag(i) = 1;

end

elseif i <= 3\*temp

if abs(Tii) < abs(TiYuZhi1)

XMFlag(i) = 0;

else

XMFlag(i) = 1;

end

else

if abs(Tii) < abs(TiYuZhi4)

XMFlag(i) = 0;

else

XMFlag(i) = 1;

end

end

end

Ti2 = zeros(1,2\*shenglunum);

XieMianPianYi = zeros(1,2\*shenglunum);

%%%%%%%%%%%%%%%%%%%%%% 对斜面上的点进行偏移（修正Ti） %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j = 1:2\*shenglunum

if j <= temp

if XMFlag(j) == 1 %斜面上

Ti2(j) = Ti(j) + a(j) \*sin(theta3);

XieMianPianYi(j) = a(j) \*tan(theta3);

else

Ti2(j) = Ti(j);

XieMianPianYi(j) = 0;

end

elseif j <= 2\*temp

if XMFlag(j) == 1 %斜面上

Ti2(j) = Ti(j) - a(j) \*sin(theta2);

XieMianPianYi(j) = a(j) \*tan(theta2);

else

Ti2(j) = Ti(j);

XieMianPianYi(j) = 0;

end

elseif j <= 3\*temp

if XMFlag(j) == 1 %斜面上

Ti2(j) = Ti(j) - a(j) \*sin(theta1);

XieMianPianYi(j) = a(j) \*tan(theta1);

else

Ti2(j) = Ti(j);

XieMianPianYi(j) = 0;

end

else

if XMFlag(j) == 1 %斜面上

Ti2(j) = Ti(j) + a(j) \*sin(theta4);

XieMianPianYi(j) = a(j) \*tan(theta4);

else

Ti2(j) = Ti(j);

XieMianPianYi(j) = 0;

end

end

end

%%%%%%%%%%%%%%%%%%%%%% 调用矩形拟合 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[~,TrianglePoints4,~,~] = planefit4(side\_faces\_transformed1,side\_faces\_transformed3,side\_faces\_transformed5,side\_faces\_transformed7,P\_bound1,P\_bound2,distanceThreshold);

%%%%%%%%%%%%%%%%%%%%%% 调用矩形参数计算 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[Pin,Pout,UPP,b,h,~,Tao,~] = Calculate\_rectangle\_from\_vertex(TrianglePoints4);

%%%%%%%%%%%%%%%%%%%%%% 矩形安装点计算 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

toff = [-a(1:shenglunum)./tan(phi),a(shenglunum+1:2\*shenglunum)./tan(phi)];

[PointTable\_A\_off4,PointTable\_B\_off4] = Calculat\_JuXing\_A\_and\_B\_Points\_after\_Offest(Tao,UPP,Pin,b,h,PAB,phi,shenglunum,Ti2,toff);

%%%%%%%%%%%%%%%%%%%%%% 计算交点 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Pin2Pup = (UPP-Pin)./norm(UPP-Pin); % 单位上方向

[xN6,yN6,zN6] = foot\_of\_perpendicular\_from\_a\_point\_to\_a\_line(PAB,Pin,Pout);

PabLine = [xN6;yN6;zN6];

PointTable\_A\_off8 = zeros(size(PointTable\_A\_off4));

for k = 1:2\*shenglunum

if k <= temp

if XMFlag(k) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(k)./2;

dir\_vec = PointTable\_A\_off4(:,k) - up2;

% 计算方向向量

a = PlaneParaOut8(1,6);

b = PlaneParaOut8(2,6);

c = PlaneParaOut8(3,6);

d = PlaneParaOut8(4,6);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_A\_off4(1,k);

yN5 = PointTable\_A\_off4(2,k);

zN5 = PointTable\_A\_off4(3,k);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_A\_off8(:,k) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_A\_off8(:,k) = PointTable\_A\_off4(:,k);

end

elseif k <= 2\*temp

if XMFlag(k) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(k)./2;

dir\_vec = PointTable\_A\_off4(:,k) - up2;

% 计算方向向量

a = PlaneParaOut8(1,4);

b = PlaneParaOut8(2,4);

c = PlaneParaOut8(3,4);

d = PlaneParaOut8(4,4);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_A\_off4(1,k);

yN5 = PointTable\_A\_off4(2,k);

zN5 = PointTable\_A\_off4(3,k);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_A\_off8(:,k) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_A\_off8(:,k) = PointTable\_A\_off4(:,k);

end

elseif k <= 3\*temp

if XMFlag(k) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(k)./2;

dir\_vec = PointTable\_A\_off4(:,k) - up2;

% 计算方向向量

a = PlaneParaOut8(1,2);

b = PlaneParaOut8(2,2);

c = PlaneParaOut8(3,2);

d = PlaneParaOut8(4,2);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_A\_off4(1,k);

yN5 = PointTable\_A\_off4(2,k);

zN5 = PointTable\_A\_off4(3,k);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_A\_off8(:,k) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_A\_off8(:,k) = PointTable\_A\_off4(:,k);

end

else

if XMFlag(k) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(k)./2;

dir\_vec = PointTable\_A\_off4(:,k) - up2;

% 计算方向向量

a = PlaneParaOut8(1,8);

b = PlaneParaOut8(2,8);

c = PlaneParaOut8(3,8);

d = PlaneParaOut8(4,8);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_A\_off4(1,k);

yN5 = PointTable\_A\_off4(2,k);

zN5 = PointTable\_A\_off4(3,k);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_A\_off8(:,k) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_A\_off8(:,k) = PointTable\_A\_off4(:,k);

end

end

end

PointTable\_B\_off8 = zeros(size(PointTable\_B\_off4));

for l = 1:2\*shenglunum

if l <= temp

if XMFlag(l) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(l)./2;

dir\_vec = PointTable\_B\_off4(:,l) - up2;

% 计算方向向量

a = PlaneParaOut8(1,8);

b = PlaneParaOut8(2,8);

c = PlaneParaOut8(3,8);

d = PlaneParaOut8(4,8);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_B\_off4(1,l);

yN5 = PointTable\_B\_off4(2,l);

zN5 = PointTable\_B\_off4(3,l);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_B\_off8(:,l) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_B\_off8(:,l) = PointTable\_B\_off4(:,l);

end

elseif l <= 2\*temp

if XMFlag(l) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(l)./2;

dir\_vec = PointTable\_B\_off4(:,l) - up2;

% 计算方向向量

a = PlaneParaOut8(1,2);

b = PlaneParaOut8(2,2);

c = PlaneParaOut8(3,2);

d = PlaneParaOut8(4,2);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_B\_off4(1,l);

yN5 = PointTable\_B\_off4(2,l);

zN5 = PointTable\_B\_off4(3,l);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_B\_off8(:,l) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_B\_off8(:,l) = PointTable\_B\_off4(:,l);

end

elseif l <= 3\*temp

if XMFlag(l) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(l)./2;

dir\_vec = PointTable\_B\_off4(:,l) - up2;

% 计算方向向量

a = PlaneParaOut8(1,4);

b = PlaneParaOut8(2,4);

c = PlaneParaOut8(3,4);

d = PlaneParaOut8(4,4);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_B\_off4(1,l);

yN5 = PointTable\_B\_off4(2,l);

zN5 = PointTable\_B\_off4(3,l);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_B\_off8(:,l) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_B\_off8(:,l) = PointTable\_B\_off4(:,l);

end

else

if XMFlag(l) == 1 %斜面上

up2 = PabLine + Pin2Pup.\*h.\*Ti2(l)./2;

dir\_vec = PointTable\_B\_off4(:,l) - up2;

% 计算方向向量

a = PlaneParaOut8(1,6);

b = PlaneParaOut8(2,6);

c = PlaneParaOut8(3,6);

d = PlaneParaOut8(4,6);

% 将直线方程代入平面方程，求解参数 t

xN5 = PointTable\_B\_off4(1,l);

yN5 = PointTable\_B\_off4(2,l);

zN5 = PointTable\_B\_off4(3,l);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect = xN5 + t \* dir\_vec(1);

y\_intersect = yN5 + t \* dir\_vec(2);

z\_intersect = zN5 + t \* dir\_vec(3);

% 输出交点

PointTable\_B\_off8(:,l) = [x\_intersect; y\_intersect; z\_intersect];

else

PointTable\_B\_off8(:,l) = PointTable\_B\_off4(:,l);

end

end

end

# 9 矩形管路复测

## 算法详细说明：公式与逻辑

此算法的核心任务是处理多个空间点的几何关系，涉及距离、角度计算、误差分析和权重分配等。下面是该算法各步骤的详细内容及其对应的公式。

## 算法实现步骤

**1. 距离计算**

计算每条声道的两个端点之间的欧几里得距离，公式为：

其中：

* 和 是第 条声道的起点和终点坐标。

**物理含义**

* **结果**：这是每条声道的长度。
* **用途**：用于后续的误差分析和几何关系计算。

**2. 夹角计算**

计算每条声道方向向量与给定方向向量 之间的夹角，使用点积公式：

其中：

* 是声道的方向向量。
* 表示方向向量 的模长。

**物理含义**

* **结果**：每条声道与给定方向的夹角 。
* **用途**：用于分析声道的方向性变化。

**3. 误差分析与校正距离计算**

通过将声道方向向量与不同平面相交，计算校正距离，并比较原始距离与校正距离之间的差异。

**步骤 1：声道与平面交点计算**

平面方程的一般形式为：

直线的参数方程为：

将直线方程代入平面方程，可以求解参数 ：

计算交点：

其中 是声道的方向向量。

**步骤 2：校正距离计算**

计算两个平面交点之间的距离：

计算误差距离：

**4. 相对高度计算**

计算每个声道端点与某基准平面的相对高度。假设平面方程为：

交点参数 ：

交点坐标：

相对高度：

**5. 权重计算 和**

**步骤 1：生成系数集合**

利用预设的权重系数集合 和参数 ，计算每条声道的权重。

**步骤 2：Lagrange 插值多项式**

计算每条声道的 Lagrange 插值：

**步骤 3：权重公式**

每条声道的权重计算为：

其中， 是归一化后的高度值。

## MATLAB代码实现

%% 计算距离

Distance = zeros(1,shenglunum);

for i = 1:shenglunum

Distance(i) = norm(PointIn(:,2\*i-1)-PointIn(:,2\*i));

end

%% 计算声道角度

theta = zeros(1,shenglunum);

for i = 1:shenglunum

shengdao1 = PointIn(:,2\*i-1);

shengdao2 = PointIn(:,2\*i);

SPXL = shengdao2 - shengdao1;

theta(i) = acos(dot(Tao, SPXL) / (norm(Tao) \* norm(SPXL)));

end

theta = pi- theta;

% degtheta = rad2deg(theta);

%% 计算LT

DistanceYC = zeros(1,shenglunum);

for i = 1:shenglunum

shengdao1 = PointIn(:,2\*i-1);

shengdao2 = PointIn(:,2\*i);

dir\_vec = shengdao2 - shengdao1;

% 计算方向向量

a = PlaneParaOut4(1,1);

b = PlaneParaOut4(2,1);

c = PlaneParaOut4(3,1);

d = PlaneParaOut4(4,1);

% 将直线方程代入平面方程，求解参数 t

xN5 = shengdao1(1);

yN5 = shengdao1(2);

zN5 = shengdao1(3);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect1 = xN5 + t \* dir\_vec(1);

y\_intersect1 = yN5 + t \* dir\_vec(2);

z\_intersect1 = zN5 + t \* dir\_vec(3);

temp3 = [x\_intersect1;y\_intersect1;z\_intersect1];

% 计算方向向量

a = PlaneParaOut4(1,3);

b = PlaneParaOut4(2,3);

c = PlaneParaOut4(3,3);

d = PlaneParaOut4(4,3);

% 将直线方程代入平面方程，求解参数 t

xN5 = shengdao1(1);

yN5 = shengdao1(2);

zN5 = shengdao1(3);

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

% 计算交点

x\_intersect2 = xN5 + t \* dir\_vec(1);

y\_intersect2 = yN5 + t \* dir\_vec(2);

z\_intersect2 = zN5 + t \* dir\_vec(3);

temp4 = [x\_intersect2;y\_intersect2;z\_intersect2];

DistanceYC(i) = norm(temp4-temp3);

end

LTPY = Distance - DistanceYC;

%% 相对高度计算

TiC = zeros(1,2\*shenglunum);

for i = 1:2\*shenglunum

shengdao = PointIn(:,i);

a = PlaneParaOut4(1,2);

b = PlaneParaOut4(2,2);

c = PlaneParaOut4(3,2);

x = Pin(1);

y = Pin(2);

z = Pin(3);

d = -(a\*x+b\*y+c\*z);

xN5 = shengdao(1);

yN5 = shengdao(2);

zN5 = shengdao(3);

dir\_vec = [a,b,c];

% 计算交点

t = -(a\*xN5 + b\*yN5 + c\*zN5 + d) / (a\*dir\_vec(1) + b\*dir\_vec(2) + c\*dir\_vec(3));

x\_intersect3 = xN5 + t \* dir\_vec(1);

y\_intersect3 = yN5 + t \* dir\_vec(2);

z\_intersect3 = zN5 + t \* dir\_vec(3);

temp5 = [x\_intersect3;y\_intersect3;z\_intersect3];

% 计算高度

TiC(i) = 2\*norm(temp5 - shengdao)/h;

end

if mod(shenglunum,2)==1 %奇数

temp6 = floor(shenglunum./2);

TiC(1:2\*temp6) = -TiC(1:2\*temp6);

else %偶数

temp6 = shenglunum./2;

TiC(1:2\*temp6) = -TiC(1:2\*temp6);

end

%% 计算权重

% k1 = 0.5;

% k2 = 0.6;

k3 = 0;

k4 = 0.15;

% gk1 = [1.570796,0.392699,0.19635,0.122718,0.085903];

% gk2 = [1.513365,0.360325,0.174351,0.106311,0.072959];

gk3 = [2,2/3,2/5,2/7,2/9];

gk4 = [1.838286,0.556753,0.315143,0.215852,0.162469];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

gk = gk3;

k = k3;

TiYiCe = TiC(1:2:2\*shenglunum);

if mod(shenglunum,2)==1 %奇数

temp6 = floor(shenglunum./2);

TiYiCe(temp6+1) = 0;

end

t\_k = zeros(shenglunum-1,1);

w = zeros(shenglunum,1);

f = zeros(shenglunum,1);

for i=1:shenglunum

count\_tk = 1;

for m=1:shenglunum

if(m~=i)

t\_k(count\_tk) = TiYiCe(m);

count\_tk = count\_tk + 1;

end

end

for j=1:(shenglunum+1)/2

f(j) = -sum(prod(nchoosek(t\_k,(shenglunum+1-2\*j)),2));

w(i) = w(i) + gk(j)\*f(j);

end

mul = 1;

for m=1:shenglunum

if(m ~= i)

mul = mul\*(TiYiCe(i) - TiYiCe(m));

end

end

w(i)=w(i)/((1-TiYiCe(i)\*TiYiCe(i))^k \* mul);

end

Wquanzhong3 = abs(repelem(w,2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

gk = gk4;

k = k4;

TiYiCe = TiC(1:2:2\*shenglunum);

if mod(shenglunum,2)==1 %奇数

temp6 = floor(shenglunum./2);

TiYiCe(temp6+1) = 0;

end

t\_k = zeros(shenglunum-1,1);

w = zeros(shenglunum,1);

f = zeros(shenglunum,1);

for i=1:shenglunum

count\_tk = 1;

for m=1:shenglunum

if(m~=i)

t\_k(count\_tk) = TiYiCe(m);

count\_tk = count\_tk + 1;

end

end

for j=1:(shenglunum+1)/2

f(j) = -sum(prod(nchoosek(t\_k,(shenglunum+1-2\*j)),2));

w(i) = w(i) + gk(j)\*f(j);

end

mul = 1;

for m=1:shenglunum

if(m ~= i)

mul = mul\*(TiYiCe(i) - TiYiCe(m));

end

end

w(i)=w(i)/((1-TiYiCe(i)\*TiYiCe(i))^k \* mul);

end

Wquanzhong4 = abs(repelem(w,2));