基于柱搜索的无人机系统定位及调整模型

摘　要

随着无线电技术的高速发展，无人机的应用场景越来越丰富，如何合理运用无源定位技术提高无人机定位系统的性能也成为无人机系统设计中的重要工作。本文依据题目所给定的纯方位无源定位方法和无人机编队情况，求得多种条件约束下的无人机系统定位模型。

针对问题 1 的第（1）问，本题需要考虑在给定三个点的位置以及这三个点与另外一个未知点的固定夹角时，如何确定该未知点。首先，本文依据纯方位无源定位下已知的三个编号信息和三个方位信息，结合**定弦定角模型、圆周角定理**，分别作出经过被动接收信号无人机以及三架发射信号无人机中任意两架的三个圆，联立求解这三个圆的**交点**，再将得到的两个交点与圆形编队的轨迹比较，经过合理的取舍便能够获取目标坐标，即求得无人机的定位模型。

针对问题 1 的第（2）问，本题应当以确定编号未知的发射信号无人机的编号为目标，优化第（1）问中的定位模型。首先本文采用**二维等分法**确定某一临界值，再按照该**临界值**将位置的偏差分为两种情况：偏差小于临界值时，用三架无人机进行所有可能位置的枚举，通过**筛选最小值**的方式确定未知编号和实际坐标；偏差大于临界值时，三架难以实现有效定位，于是考虑用四架无人机发射信号，结合上一问中的定位模型，采用**筛选相同值**的方式实现对无人机的准确定位。

针对问题 1 的第（3）问，本文首先建立各无人机的各类位置状态向量，通过枚举发射信号无人机的所有组合，得到不同的预测位置，计算各组合调整后实际位置与理想位置的偏差平方和，将其作为**柱搜索算法的打分函数**，贪心选取最优解。考虑到实际飞行过程中的误差，本文加入呈正态分布的角度测量误差与飞行偏离误差，并以 **Momentum 算法**对调整距离进行优化，最终无人机将在理想位置附近不断徘徊，得到“飞行云”图。

针对问题 2，本文通过 t 分布随机采样得到初始位置点，再利用问题 1 的第（3）问中建立的**无人机编队调整模型**，将该模型在无人机锥形编队中进一步推广，得出问题四的调整方案。

关键词：计算几何　二维等分法　柱搜索算法　Momentum 算法　抽样分布

1. 问题重述

1.1. 问题背景

随着控制工程、电子技术等的高速发展，无人机技术逐渐在我们的生活中发挥着越来越重要的作用。目前，无人机技术已经被广泛应用于国防军事、物流运输、农业灌溉、测绘工程等多个社会领域。

而无源定位系统是一种被定位对象本身不发射信号而依赖接收辐射源的信号来获取目标信息的定位系统。在无源定位下，被定位对象不必发射电磁波信号，只需通过接收辐射源的信号，利用其中的参数信息进行定位，减少了电磁波的释放，使得被定位对象较难被外界感知，避免了被干扰的问题。因此，无源定位系统在有效提高无人机系统的抗干扰能力、提高无人机在社会各领域的工作效率方面呈现出了较大的优势，如何合理的运用无源定位技术来定位无人机也成了影响无人机运行效率的重要因素。

1.2. 需要解决的问题

由题目可知，我们需要解决以下四个问题：

问题一：基于问题 1 中给出的无人机编队模型，接收信号的无人机仅已知其与任意两架发射信号的无人机连线之间的夹角。在 FY00 和另外两架无人机位置无偏差且编号已知的情况下，求解得到被动接收信号无人机的具体位置。

问题二：在问题一的基础上，将发射信号的无人机条件改为 FY00、FY01 和若干架编号未知的无人机发射信号，求解如何实现无人机的有效定位。

问题三：在初始位置存在偏差的条件下，利用无人机 FY00 和圆周上最多 3 架无人机发射信号，如何确定无人机的调整方案。

问题四：将问题 1 中的解决方案进行推广，求解当圆形编队队形改变为锥形编队队形时无人机的调整方案。

2. 问题分析

2.1. 问题一

问题一要求在发射信号的无人机位置无偏差且编号已知的情况下建立被动接收信号无人机的定位模型。首先需要基于无人机圆形编队的特征，建立平面直角坐标系，其次利用发射信号无人机的位置信息和被动接收信号无人机的方向信息，结合圆周角定理、定弦定角模型，通过计算求解得到多个相关圆的解析表达式以及其交点坐标，即被动接收信号无人机的位置信息，进而求得定位模型。

2.2. 问题二

问题二相对问题一而言，减少了编号已知的条件，要求在部分发射信号无人机编号未知的情况下确定被动接收信号无人机的位置。首先，我们需要对“略有偏差”进行定义，认为在该情况下无人机队列仍保持原有顺序，并且按照一定的临界值将被动接收信号无人机的位置偏差分为两种情况。在位置偏差小于临界值的情况下，我们尝试利用三架无人机进行被动接收信号无人机可能位置的枚举，通过比较其与理想位置的距离进行筛选和定位；在位置偏差大于临界值的情况下，考虑利用四架无人机发射信号，基于问题一中的定位模型，采用筛选相同值的方式对无人机的位置进行合理预测。

2.3. 问题三

在问题三中，由于初始位置的偏差，根据问题一中的模型结果，接收信号的无人机无法做出准确的定位，做出相应调整后也无法达到理想位置。因此我们在每一次调整前采用柱搜索枚举不同的发射信号的无人机组合，保留调整后的位置与理想位置更接近前 k 项组合，再重复此轮操作。由此不断迭代与选择，得到最优的选择发射信号无人机的组合策略。而由于飞行过程中存在的误差以及测量角度的误差，我们将误差考虑到无人机实际位置的计算中，使结果更加符合实际生活。

2.4. 问题四

相对于问题三而言，问题四改变了编队队形，要求给出在锥形编队下无人机

的调整策略。首先，我们建立以领队无人机 FY00 为原点，以其航向为 x 轴正方向的平面直角坐标系；此后，便可以利用问题三中所建立的模型进行推广，得到在实际飞行过程中，存在诸多误差的情况下，多次调整时选择发射信号的无人机的组合；在仿真验证时，我们按照 t 分布对理想位置周边的点进行采样，得到一些与理想位置有偏差的点作为初始位置。

3. 模型假设

1. 将每架无人机都视为质点，即其体积可以忽略不计。
2. 假设在无人机调整过程中，发射信号的无人机不进行位置的调整。
3. 假设无人机所接收到的角度信息已经进行过相应的处理。
4. 假设无人机所发射的电磁波不受外界干扰，即电磁波都能够顺利被接收。

4. 符号说明

表 1 符号说明

符号	说明
a，β，γ	接收信号无人机所接收到的方向信号
(x_n, y_n)	发射信号无人机 FY0X 的位置坐标
(x_i, y_i)	接收信号无人机 FY0i 的可行坐标
(x_i', y_i')	接收信号无人机 FY0i 的理想位置坐标
(x_{ij}, y_{ij})	无人机 FY0i 根据角度确定的可行坐标
S_n	无人机 FY0i 可行坐标的集合
$\vec{S}_{(理想，FY0j)}$	无人机 FY0j 的理想位置状态向量
$\vec{S}_{(预测，FY0j，i)}$	无人机 FY0j 的预测位置状态向量
$\vec{S}_{(实际，FY0j，i)}$	无人机 FY0j 的实际位置状态向量
m^t	无人机第 i 次预计调整的距离向量
$\vec{Z}_{(FY0j,i)}$	无人机FY0j第i次调整前相对中心无人机所产生的偏移向量

5. 模型建立与求解

5.1. 问题一

问题一中我们假定发射方向信号的为 FY00、FY01、FY02 三架无人机（其余情况同理），以 FY00 无人机所在位置为原点建立平面直角坐标系，结合圆周角定理、定弦定角模型，分别计算经过被动接收信号无人机以及任意两架发射信号无人机的三个圆的解析表达式，进而计算这三个圆的交点坐标，即被动接收信号无人机的位置。

5.1.1. 定弦定角模型

如果存在一段长度固定的线段 AC，且其所对应的∠B 的大小也一定，则点 B 不是固定的，其运动轨迹是一个圆或一段弧，如图 1 所示。

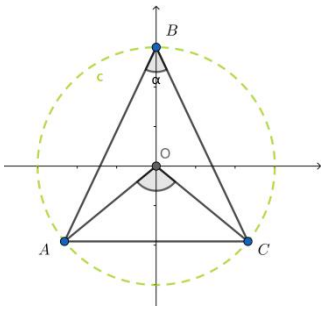

图 1 定弦定角模型

现以圆心 O 为原点建立直角坐标系，假定 AC=2a，∠B=α。根据圆周角定理，可以得到：∠AOC=2α，A$\left(-a,-\frac{a}{\tan\alpha}\right)$，C$\left(a,-\frac{a}{\tan\alpha}\right)$，圆的半径为$\frac{a}{\sin\alpha}$，所以可以得到该圆的解析表达式为$x^2+y^2=\left(\frac{a}{\sin\alpha}\right)^2$。

因此，已知一段长度固定的线段和其对应的固定角度，等价于已知一个圆的弦长和对应的圆周角，可以得到对应圆的解析表达式。

5.1.2. 定位模型的建立

首先，考虑到圆形编队的对称性，我们以 FY00 无人机所在位置为原点 A$(0,0)$ 建

立平面直角坐标系，同时，假定另外两架发射信号的无人机坐标分别为 B(x_1, y_1)和 C(x_2, y_2)，被动接收信号无人机 FY0i 的坐标 D 为(x_i, y_i)，其中 i=3,4,...,8，且被动接收信号无人机接收到的方向信号分别为α，β，γ，如图 2 所示。

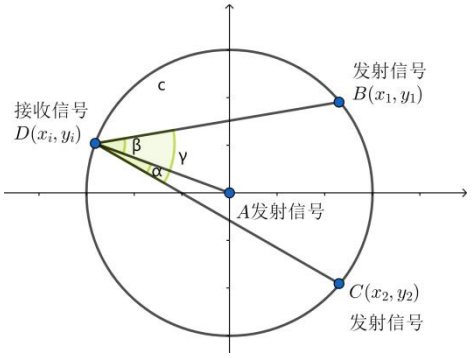

图 2 方向信号示意图

然后利用定弦定角模型，分别作出经过被动接收信号无人机点 D 和三架发射信号的无人机 A、B、C 中任意两点的圆，由于被动接收信号的无人机只能获取方向信息，点 D 的具体坐标不确定，会出现以下两种对称的三圆相交于一点的情况，如图 3、图 4 所示。

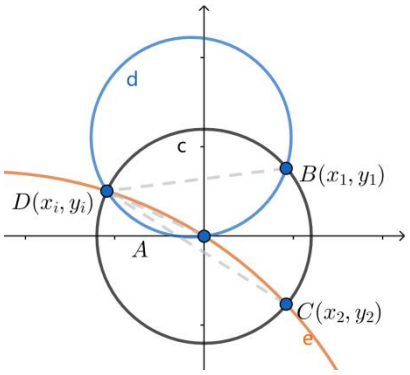

图 3 第一种情况

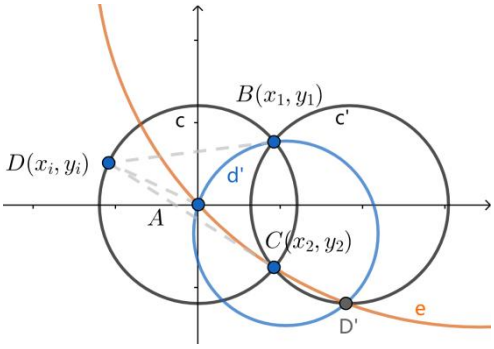

图 4 第二种情况

现假设圆 c、d、e、c’、d’、e’的半径分别为 r_1，r_2，r_3，r_1，r_2，r_3，圆心分为 A_1、A_2、A_3、A_1’、A_2’、A_3’，利用定弦定角模型和圆周角定理求解圆 c 和圆 c’的半径、圆心及解析表达式：

$$BC^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{1}$$

假定M为BC的中点，则$BM = \frac{BC}{2}$，

$$r_1 = \frac{BM}{\sin(\alpha+\beta)} \tag{2}$$

$$k_{BC} = \frac{y_1 - y_2}{x_1 - x_2} \tag{3}$$

$$\theta = \arctan\frac{k_{BC} - 0}{1 - k_{BC}\cdot 0} \tag{4}$$

$$x_{A1} = x_1 + r_1 \cdot \cos(\frac{\pi}{2} - \alpha - \beta + \theta) \tag{5}$$

$$y_{A1} = y_1 + r_1 \cdot \sin(\frac{\pi}{2} - \alpha - \beta + \theta) \tag{6}$$

其中，x_{A1} 是点 A_1 的横坐标，y_{A1} 是点 A_1 的纵坐标，k_{BC} 是直线 BC 的斜率。

同时，可以由对称性得到：

$$x_{A1'} = x_1 + x_2 - x_{A1} \tag{7}$$

$$y_{A1'} = y_1 + y_2 - y_{A1} \tag{8}$$

因此，可以得到圆 c 和圆 c’的解析表达式：

$$(x - x_{A1})^2 + (y - y_{A1})^2 = r_1^2 \tag{9}$$

$$(x - x_{A1'})^2 + (y - y_{A1'})^2 = r_1^2 \tag{10}$$

同理，可以得到圆 d、d’、e、e’的解析表达式：

$$(x - x_{A_2})^2 + (y - y_{A_2})^2 = r_2^2 \tag{11}$$

$$(x - x_{A2'})^2 + (y - y_{A2'})^2 = r_2^2 \tag{12}$$

$$(x - x_{A_3})^2 + (y - y_{A_3})^2 = r_3^2 \tag{13}$$

$$(x - x_{A3'})^2 + (y - y_{A3'})^2 = r_3^2 \tag{14}$$

由于 c 是经过 B、C、D 的圆，d 是经过 A、B、D 的圆，e 是经过 A、C、D 的圆，因此圆 c、圆 d、圆 e 一定会有一个共同的交点 D。又因为任意两个圆相交最多只会存在两个交点(不考虑重合的情况)，所以圆 c 与圆 d 只会交于点 D 与点 B，圆 c 与圆 e 只会交于点 D 与点 C，圆 e 与圆 d 只会交于点 D 与点 A，即圆 c、d、e 不会交于 A、B、C、D 以外的点，圆 c’、d’、e’同理。故圆 c、d、e 有且只可能有一个共同的交点 D；而圆 c’、d’、e’也有且只可能有一个共同的交点 D’，并且只有点 D 在圆形编队的轨迹上。因此通过求解这三个圆的交点坐标可以确定被动接收信号无人机的位置。

于是我们联立以上六个解析表达式，可以求解得到圆 c、d、e 唯一的公共交

点 D 和圆 c’、d’、e’唯一的公共交点 D’。

最后将点 D 与点 D’的坐标与圆形编队的轨迹比较，可以知道点 D’偏离了圆形编队的轨迹。因此，点 D 是我们的目标坐标，即被动接收信号无人机的位置。

5.1.3. 定位模型的求解

（1）无人机定位模型

假定被动接收信号无人机的坐标为 $\mathrm{D}(x,y)$，发射信号的无人机坐标分别为 $\mathrm{A}_1(0,0)$、$\mathrm{B}(x_1,y_1)$ 和 $\mathrm{C}(x_2,y_2)$，无人机接收到的三个方向信号为 $\varepsilon_i(i=1,2,3)$，再依据以上分析过程作圆，圆心坐标分别为 $A_i(i=1,2,3)$，$A_i'(i=1,2,3)$，联立以上式(1)-(14)，可以得到如下定位模型：

$$
\begin{cases}
BC^2=(x_1-x_2)^2+(y_1-y_2)^2 \\
r_i=\dfrac{BC}{2\sin\varepsilon_i} \\
k_{BC}=\dfrac{y_1-y_2}{x_1-x_2} \\
x_i=x_1+r_i\cdot\cos(\dfrac{\pi}{2}-\varepsilon_i+\arctan k_{BC}) \\
y_i=y_1+r_i\cdot\sin(\dfrac{\pi}{2}-\varepsilon_i+\arctan k_{BC}) \\
x_i'=x_1+x_2-x_i \\
y_i'=y_1+y_2-y_i \\
(x-x_i)^2+(y-y_i)^2=r_i^2 \\
(x-x_i')^2+(y-y_i')^2=r_i^2
\end{cases}
\qquad i=1,2,3
$$

（2）算法求解

基于以上定位模型，通过计算几何的算法，利用 Python 运算程序可以求得被动接收信号无人机的位置信息。（详见附录参考代码一）

5.2. 问题二

首先，我们需要对无人机位置的偏差范围进行定义，假定稍有偏差的情况下无人机圆形编队队列仍保持原有顺序，此时三架还是四架无人机可以实现无人机的有效定位取决于无人机的实际偏差范围与临界值的关系。

5.2.1. 三架无人机的定位方法

首先，我们尝试用 FY00、FY01 和一架编号未知的无人机进行定位：

基于问题一所建立的直角坐标系，可知无人机 FY00 的位置坐标为$(0,0)$，现假定圆形编队的半径为 R，无人机 FY01 的位置坐标为(x_1, y_1)，编号未知的信号发射无人机 FY0X 的的位置坐标为(x_n, y_n)，被动接收信号无人机 FY0i 的预测坐标为(x_i, y_i)。其中，$i = 2,3,...,9$；$n = 2,3,...,9$且$n \neq i$。此外，在只选取三架无人机的情况下，可以得到方向信号α，β，γ。同时，为保持其均匀分布且美观的队形，每架无人机都存在其对应的理想位置(x_i', y_i')，例如，FY02 的理想位置是$(R\cos 40°, R\sin 40°)$，FY03 的理想位置是$(R\cos 80°, R\sin 80°)$。

依据已有条件，我们可以通过以下几个步骤定位无人机 FY0i 的位置坐标：

Step1：建立信号发射无人机的选择库。由题目条件可知，发射信号的无人机位置无偏差，于是我们可以先基于各无人机的理想坐标建立一个不包含无人机 FY00、FY01 的信号发射无人机选择库 $\boldsymbol{N}$：

$$\boldsymbol{N} = \{(x_2', y_2'), (x_3', y_3'), ..., (x_n', y_n')\} \quad \text{其中，} n = 2,3,...,9 \text{ 且} n \neq i$$

Step2：利用定位模型枚举求解包含 FY0i 所有可行坐标的集合。当我们想要定位一架接收到的方向信号为α，β，γ 的无人机 FY0i 时，可以依次选取点 FY00、点 FY01 以及选择库 $\boldsymbol{N}$ 中的一个点 FY0X，利用三点坐标和方向信号α，β，γ，结合问题一中的定位模型，得到一系列在该角度条件下，FY00、FY01 以及 FY0X 三点能够确定的被动接收信号无人机可行坐标(x_{ij}, y_{ij})构成的集合 S_n，其中，$j = 1,2,3,...$，且 S_n 可能为空集。在遍历整个选择库 $\boldsymbol{N}$ 后，可以得到 $S' = \{S_2, S_3, ..., S_n\}$。

Step3：合理筛选可行坐标从而确定被动接收信号无人机的实际坐标。考虑到该情况下被动接收信号无人机的位置偏差较小，故可以认为此时的被动接收信号无人机的实际坐标与其理想坐标十分接近。因此我们按照式(15)所示的规则进行筛选，可以得到发射信号无人机的编号以及被动接收信号无人机的实际坐标，即选择与被动接收信号无人机理想坐标的欧式距离 d 最小的可行坐标作为被动接收信号无人机的实际位置坐标(x_{im}, y_{im})：

$$d_m^{\,2} = \min\left\{(x_{i1}-x_i')^2+(y_{i1}-y_i')^2,(x_{i2}-x_i')^2+(y_{i2}-y_i')^2,...,(x_{ij}-x_i')^2+(y_{ij}-y_i')^2\right\}$$
$$=(x_{im}-x_i')^2+(y_{im}-y_i')^2 \tag{15}$$

其中，$j=1,2,3,...$ ，$i=2,3,...,9$ ，$(x_{ij},y_{ij})\in S'$。

5.2.2. 基于二维等分法确定临界值

通过上述关于三架无人机定位方法的分析，可以知道只有被动接收信号无人机的位置偏差在一定范围内时，即偏差要小于某个临界值，三架无人机定位的筛选方法才是可行的。下面，我们利用二维等分法，结合上述三架无人机定位方法计算三架无人机定位可行与否的临界值：

（1）**枚举所有可能信息接发组合。**首先，我们以无人机 FY00、无人机 FY01 和其他任意一架编号已知的无人机作为信号发射无人机，再以上述三架无人机之外的任意一架无人机作为接收信号无人机的，构成一组信息接发组合。枚举所有可能组合。

（2） **二维等分法**。以方向和长度为二维等分的对象，对于一个可能组合，先等分方向，在每个方向上通过二分长度找到临界值，通过最小临界值对应的方向，细化方向范围，进行下一轮迭代，直到在精确的方向找到最小临界值，作为该组合的最小临界值。

（3）**求得最小值**。通过二维等分法，对每一个组合求解无人机定位可行与否的最小临界值。每一个组合的最小临界值的最小值即为所求最小值。

基于以上分析，利用 Python 运算程序求解，得到临界值的结果是 0.1619873046875R，接发组合是 FY00、FY01 FY05 发射 FY04 接收，最小临界坐标 (-0.04235603683550458R, 0.15635169658021093R)。误差轨迹如右图所示（详见附录参考代码二）：

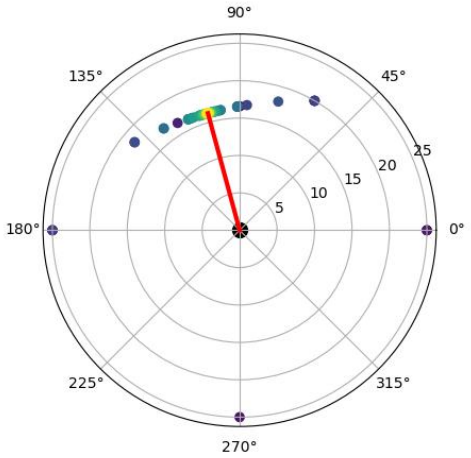

图 5 三架无人机定位时的误差轨迹

5.2.3. 四架无人机的定位方法

当被动接收信号无人机的位置偏差超过上述临界值时，被动接收信号无人机的位置无法通过三架无人机确定。于是，我们考虑用 FY00、FY01 和两架编号未知的无人机，通过求解两组可行解的交集进行定位：

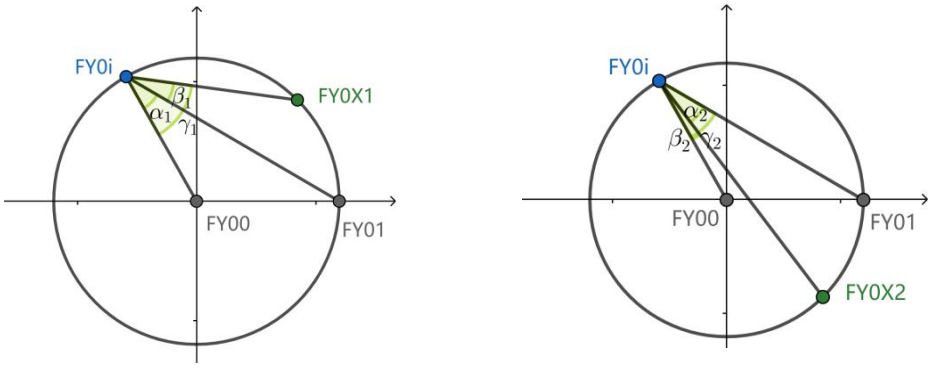

图 6 两组方向信息示意图

仍基于问题一中建立的平面直角坐标系，无人机 FY00 的位置坐标为$(0,0)$，假定圆形编队的半径为 R，无人机 FY01 的位置坐标为(x_1, y_1)，编号未知的信号发射无人机 FY0X1、FY0X2 的位置坐标分别为(x_n, y_n)和(x_m, y_m)，被动接收信号无人机 FY0i 的预测坐标为(x_i, y_i)，其中，$i = 2,3,...,9$；$n, m = 2,3,...,9$且$n \neq m \neq i$。该情况下，被动接收信号无人机可以接收到两组方向信号，一组为 FY0X1 形成的α_1、β_1、γ_1，另一组为 FY0X2 形成的α_2、β_2、γ_2，如图 5 所示。

基于以上信息，我们可以通过以下两个步骤定位无人机 FY0i 的位置坐标：

Step1：将四架飞机分为两组，分别求解被动接收信号无人机所有可行坐标的集合。实际上，利用四架飞机进行定位等价于用两组三架飞机进行定位，因此我们将这四架飞机分为两组，其中一组包括无人机 FY00、FY01、FY0X1，另一组包括无人机 FY00、FY01、FY0X2，两组分别按照三架飞机时的步骤进行求解，可以得到两个包含 FY0i 所有可行坐标的集合，分别为S_1'和S_2'。

Step2：利用被动接收信号无人机自身的编号信息在有限个可行坐标中筛选得到实际坐标。因为被动接收信号无人机的实际坐标同时存在于集合S_1'和S_2'中，所以集合S_1'和S_2'的交集一定包含 FY0i 的实际坐标。又由于被动接收信号无人机对于自己的编号是已知的，我们可以通过其自身的编号信息对集合S_1'和S_2'中的交集坐标进行筛选，从而确定无人机的实际坐标。相较于三架飞机，利用四架

飞机定位时不需再考虑误差，无论被动接收信号无人机的位置偏差多大，都能准确实现被动接收信号无人机的定位。

5.2.4. 基于无人机偏差范围的方法选择

依据以上分析可知，当无人机的偏差范围小于临界值 0.1619873046875R 时，通过以上方法求解得到的最小值点一定是无人机的实际坐标点，因此无人机 FY00、无人机 FY01 和一架编号未知的信号发射无人机即可实现无人机的有效定位；

但当无人机的偏差范围大于临界值时，选取最小值的方式无法确定实际坐标，故需要通过无人机 FY00、无人机 FY01 和两架编号未知的信号发射无人机来实现无人机的有效定位。

5.2.5. 模型二的求解

基于以上分析，利用 Python 运算程序可以求解被动接收信号无人机在其位置偏差小于临界值、位置偏差大于临界值两种情况下准确的位置坐标。（详见参考代码二）

5.3. 问题三

5.3.1. 建模前的准备

（1）无人机初始位置的偏离

由题意可知，要求 10 架无人机组成圆形编队，其中编号 FY00 在圆心位置，而编号 FY01-FY09 均匀分布在半径为 100m 的圆周上。在初始时刻，除编号为 FY00 和编号 FY01 的无人机处在理想位置上，其余无人机均略有偏差，其偏差程度如图 7 所示。

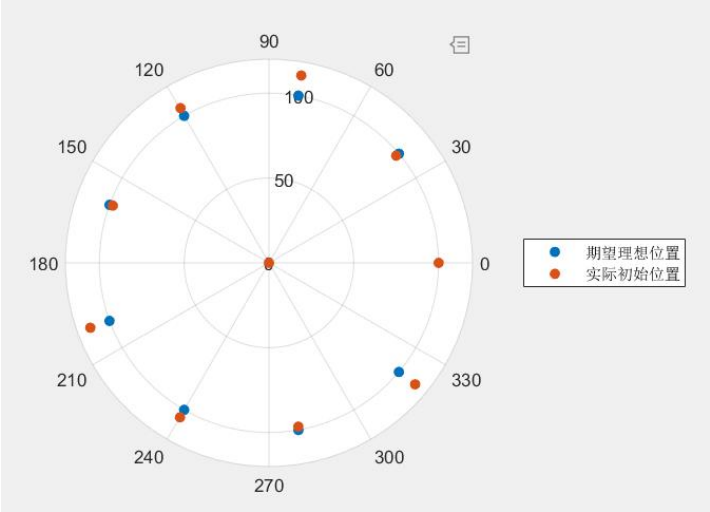

图 7 无人机初始位置的偏离情况

（2）柱搜索（Beam Search）

柱搜索[1]是一种启发式图搜索算法，使用广度优先策略建立搜索树在树的每一层，在图的解空间比较大的情况下，在深度扩展时按照所预设的规则，对所有的节点进行排序，保留下质量较高的前 k 个节点，剪掉其他质量较差的节点，以减少搜索占用的空间和时间。在每一步深度扩展时，均按照此规则保留 k 个较优候选节点，最后从 k 个候选中选择最优的策略。

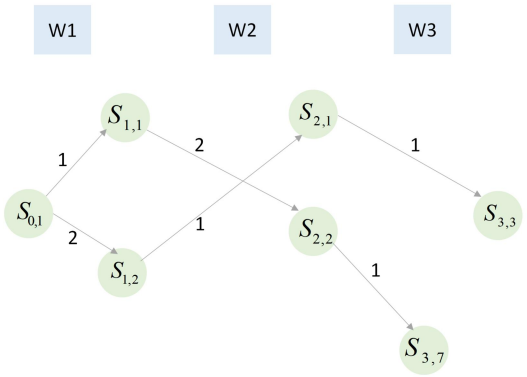

图 8 beamsize 为 2 时的柱搜索示例

5.3.2. 理想情况下模型三的建立

（1）固定发射信号无人机的选取

在初始位置时，编号为 FY00 和编号 FY01 的无人机处在理想位置上，不需要调整，并且作为参考系误差最小，因此在选择发射信号的无人机时，我们将编号为 FY00 的中心无人机和编号为 FY01 的无人机作为固定选择的信号发射机，另外

选取一架或者两架无人机作为一个组合发射信号，并运用问题一中所得到的定位模型对接收信号的无人机进行定位。

（2）状态变量的建立

在问题三中，对于编号为FY0j的无人机，我们将无人机根据定位模型所得到的位置称为预测位置，将无人机实际所处的位置称为实际位置，将无人机最终所需达到的位置称为理想位置。

在极坐标系下理想位置状态向量为

$$\vec{S}_{(\text{理想，FY0j})}=\begin{cases}(0,0)\text{，}\ \mathrm{j}=0\\(100,40j)\text{，}\ j=1,2,3,4,5,6,7,8,9\end{cases} \tag{16}$$

将极坐标转换为直角坐标，则有

$$\vec{S}_{(\text{理想，FY0j})}=\begin{cases}(0,0)\text{，}\ \mathrm{j}=0\\(100\cos(40\mathrm{j}),100\sin(40j))\text{，}\ j=1,2,3,4,5,6,7,8,9\end{cases} \tag{17}$$

该无人机第 i 次根据定位模型得到的预测位置状态向量为

$$\vec{S}_{(\text{预测，}FY0\mathrm{j}\text{，i})}=(\mathrm{x1}_{(FY0j,i)}\text{，}\ \mathrm{y1}_{(FY0j,i)})\text{，}$$

第 i 次预计调整的距离向量应为理想位置状态向量减去预测位置状态向量，即

$$\vec{\mathrm{m}}_{(\text{理想，FY0j，i})}=(\mathrm{m}_{\mathrm{x(FY0j，i)}},\mathrm{m}_{\mathrm{y(FY0j，i)}})$$

当 i=0 时，实际位置状态向量为初始位置的状态向量；当$\mathrm{i}\geq 1$时，经过第 i 次调整后的实际位置状态向量为第 i-1 次调整以后的实际位置状态向量加上预计调整的距离向量，即

$$\vec{S}_{(\text{实际，}FY0\mathrm{j}\text{，i})}=\begin{cases}(x0_{FY0j},y0_{FY0j}),i=0\\\vec{S}_{(\text{实际，}FY0\mathrm{j}\text{，i-1})}+\vec{\mathrm{m}}_{(\text{理想，FY0j，i})},i\geq 1\end{cases} \tag{18}$$

（3）基于 Momentum 优化算法对距离向量的优化

Momentum 优化算法是对梯度下降法的一种优化，具体做法是加入历史梯度向量作为一个参考，通过指数移动加权平均，并加入衰减率，计算梯度的更新方向，以消除当前轮迭代梯度向量存在的方向抖动。

在本模型中，第 i 轮迭代的梯度向量即第 i 次预计调整的距离向量$\mathrm{m}^{\mathrm{t}}=[\mathrm{m}_{\mathrm{x}}^{\mathrm{t}},\mathrm{m}_{\mathrm{y}}^{\mathrm{t}}]^{\mathrm{T}}$，设置衰减率为β，$\mathrm{m}_{\mathrm{w}}^{\mathrm{t}}$作为新的梯度更新方向，则有

$$\begin{bmatrix} \beta^{t-1}m_x^1+\beta^{t-2}m_x^2+\ldots+\beta^0 m_x^t \\ \beta^{t-1}m_y^1+\beta^{t-2}m_y^2+\ldots+\beta^0 m_y^t \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{t-1}\beta^i m_x^i \\ \sum_{i=0}^{t-1}\beta^i m_y^i \end{bmatrix} = m_{\mathrm{w}}^{\mathrm{t}} \tag{19}$$

即在下一次调整前将预测位置状态向量进行更新后，再进行下一次迭代，而经实验验证，发现当衰减率β=1 时，迭代速度是最快的。

但是，在实际情况中，无人机在做出调整时，做出调整前的历史速度会对调整后移动的距离向量有所影响，因此在后面的优化模型中，我们利用衰减率β作为历史速度所带来的误差对队形编队的影响。

（4）基于柱搜索的最优调整模型

综上可得，由于初始位置的偏差，部分发射信号的无人机并未处在理想位置上，导致其他接收信号的无人机所得到的预测位置也与实际位置有所偏差。因此，当接收信号的无人机根据预测位置来做出相应的调整后，并不能达到理想位置，因此我们在每一次调整前采用柱搜索枚举不同的发射信号的无人机组合，保留调整后的位置与理想位置更接近前 k 项组合，再重复此轮操作。由此不断迭代与选择，得到最优的选择发射信号无人机的组合策略。

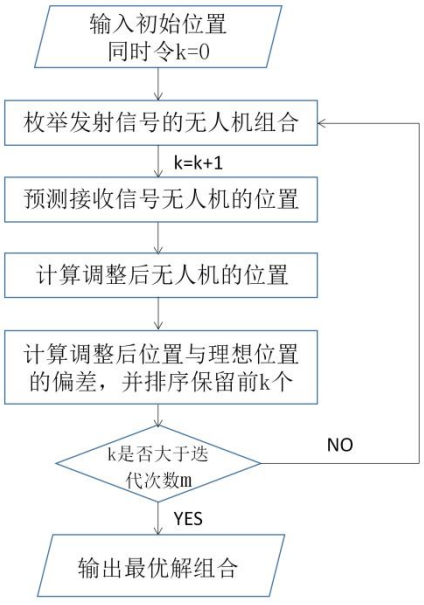

图 9 最优调整模型的实现流程

5.3.3. 模型三的求解

利用 Python 运算程序求解可得，每次调整所选择的发射信号的无人机组合

如表 1 所示（详见附录参考代码三）：

表 1 在理想情况下发射信号无人机的组合及误差

第 i 次调整	在该位置上发射信号的无人机组合	该位置的误差
0	(0, 1, 7, 8)	5.2083
1	(0, 1, 2)	1.1196
2	(0, 1, 5)	0.21869
3	(0, 1, 9)	0.01932
4	(0, 1, 8)	6.9994E-05
5	(0, 1, 6)	8.8057E-07
6	(0, 1, 2)	1.9042E-10
...	...	...

由表可得，误差 w0 与调整次数 i 之间的关系如下：

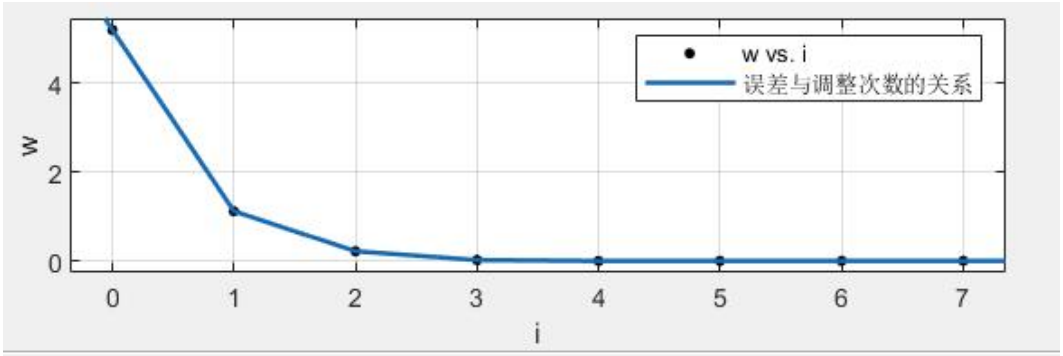

图 10 误差与调整次数 i 的关系

由此可得，经过多次调整后，无人机实际位置与理想位置的误差逐渐趋近于 0 即实际位置逐渐收敛于理想位置。利用 MATLAB 画出每次调整后，无人机所在位置，如图 11 所示。

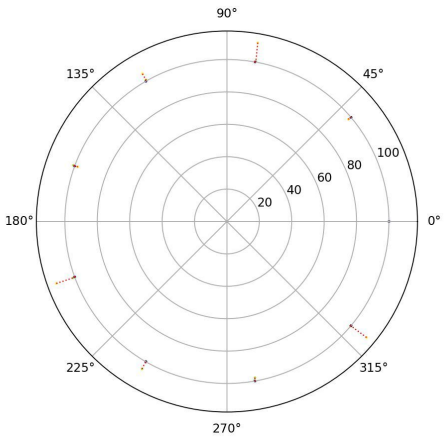

图 11 无误差时的调整轨迹

在实际调整中，我们可根据需要的误差，选择合适的迭代调整次数，减少电磁波的发射。

5.3.4. 模型三的优化

由上述模型的结果可得，无人机实际位置会越来越趋近于理想位置，误差可以逐渐趋向于 0。但是在实际飞行中，由于操作误差或环境的影响，误差不可能趋近于 0。

在上述的考虑中，第一，我们认为编号为 FY00，FY01 的无人机始终处在理想位置上，且其他无人机在未调整时坐标无变化，而在实际飞行过程中，由于无人机传感器的角度、惯性、空气阻力以及其他障碍物的出现等因素，在未做出调整时无人机的坐标也会发生改变，导致编号为 FY00，FY01 的无人机始终处在理想位置这一条件无法成立，以及在第 i 次做出调整前的实际位置也会发生改变；第二，在实际情况中，方位角传感器的测量也会有误差；第三，无人机在向理想方位调整的过程中，由于无人机的惯性，做出调整前的历史速度会对调整后移动的距离向量有所影响。

因此，我们根据资料[5,6]将诸多误差因素大致分为三大类，一类使由于传感器接受方向信息时产生的随机误差，该随机误差服从正态分布 N（0，0.00001），我们选择在运用问题一的定位模型时在方位角中加入误差模拟实际测量；另一类是在飞机保持编队过程中，由于操作者与环境所影响的无人机位置偏差，同样用以服从 N（0，1）的正态分布随机误差表示；还有一类是由于无人机的惯性带来的历史速度方向的距离误差，我们利用衰减率β模拟历史速度所带来的误差对队形编队的影响。那么，上述模型所需做出的调整如下：

（1）编号为 FY00 的无人机给予编号 FY01-09 无人机群体偏差

由于飞行时的误差，会导致编号为 FY00 与 FY01 无人机也会产生偏差。但由于编号 FY00 无人机的特殊位置，我们认为编号 FY01-09 无人机始终配合中心 FY00 无人机保持编队队形，因此我们建立一个以该无人机作为坐标原点，以其航向作为坐标轴正向的极坐标系。为了消除编号 FY00 在飞行过程中所带来的误差，我们给予编号 FY01-09 无人机一个群体偏差 N（0，1），以认为 FY00 无人机一直处在理想位置。例如当 FY00 无人机往偏上时，我们认为 FY00 无人机没

有偏差，而编号 FY01-09 无人机有一个向下的偏差，因此可等价于 FY00 无人机的向上偏差。

（2）第 i 次调整前实际位置状态向量

假设在保持编队的飞行过程中，圆周上编号为 FY0j 的无人机在第 i 次调整前相对中心无人机所产生的偏移向量为 $\bar{Z}_{(FY0j,i)}=(z_{x(FY0j,i)},z_{y(FY0j,i)})$，其服从正态分布 N（0，1）。

由于实际飞行时误差的存在，第 i 次调整前的实际位置状态向量为第 i-1 次调整以后的实际位置状态向量加上无人机偏移向量，即

$$\vec{S}_{(实际，FY0j，i)}=\vec{S}_{(实际，FY0j，i-1)}+\vec{Z}_{(FY0j，i)} \tag{20}$$

（3）方位角传感器的测量误差

假设由于传感器接受方向信息时产生的随机误差，误差服从正态分布。则接收角 $\alpha_{接收}$ 等于真实值 $\alpha_{真实}$ 加上误差 $\alpha_{误差}$~N（0, 0.00001）

$$\alpha_{接收}=\alpha_{真实}+\alpha_{误差} \tag{21}$$

（4）**第 i 次调整前历史速度误差**

假设无人机的惯性带来的历史速度方向的距离误差，利用衰减率β模拟历史速度所带来的误差对队形编队的影响。

该误差在理想情况下已经阐述，此处选择β=0.9 作为模拟误差的参数。

（5）**发射信号无人机的选取**

当考虑实际情况，FY01 位置并不理想，则不再必定选取 FY01。选取的策略泛化为选取中心无人机 FY00 和任意两架或三架无人机。

对模型进行优化后，我们得到了更加符合实际情况的结果，每次调整所选择的发射信号的无人机组合如表 2 所示（详见附录参考代码三）：

表 2 在有随机误差的情况下发射信号无人机组合及误差

第 i 次调整	在该位置上发射信号的无人机组合	该位置的误差
0	(0，1，2，4)	5.2083
1	(0，1，6，9)	2.2104
2	(0，1，3，7)	0.8208

3	(0, 2, 4, 9)	0.6075
4	(0, 1, 3)	0.5938
5	(0, 4, 7, 8)	0.4899
6	(0, 2, 7, 9)	0.4604
7	(0, 1, 5, 7)	0.4239
...	...	...

由表可得，误差 w1 与调整次数 i 之间的关系如下：

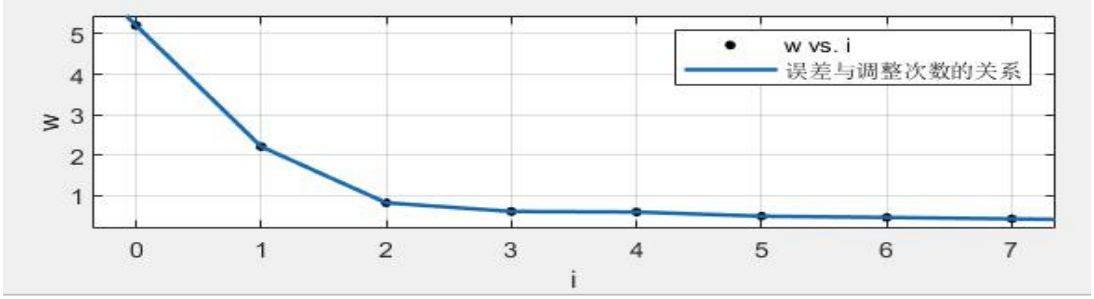

图 12 误差 w0 与调整次数 i 的关系

由此可得，在实际情况中，由于误差的存在，无人机并不能无限的接近理想位置，而是在一定的误差范围内，无人机不断在理想位置附近徘徊，而该误差的大小则由无人机传感器的角度、速度惯性、空气阻力以及其他障碍物的出现等诸多因素决定。因此在实际情况中无人机无限逼近或者在飞行过程中一直保持队形是不可能的，但根据我们所建立的模型，在每次调整时，可以根据所偏差的距离调整队形，并且一直保持一个相对整齐的队形。

根据算法所求得的结果，利用 Python 的 matplotlib 库，我们可以得到每次调整后无人机所在位置如图 13 所示。

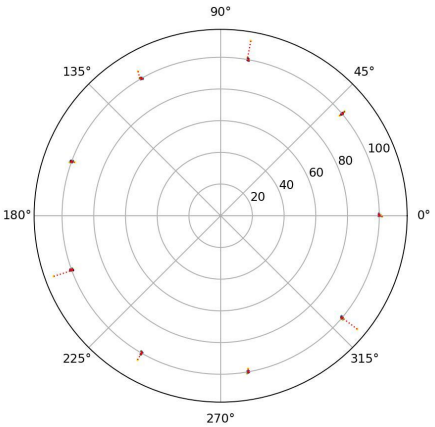

图 13 有误差时的调整轨迹

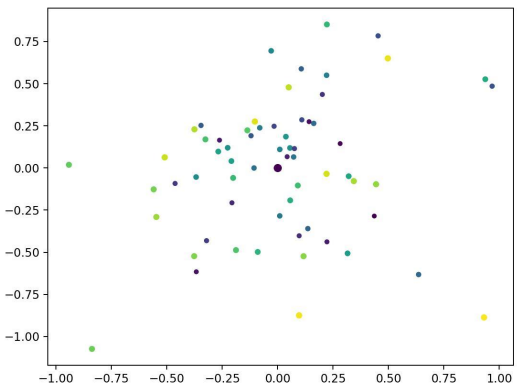

图 14 “飞行云”图

如上所说，无人机不断在理想位置处徘徊。因此，我们根据所求得的位置数据，利用利用 Python 的 matplotlib 库可以得到一个“飞行云”，如图 14 所示。

5.4. 问题四

实际飞行中，无人机集群也可以是其他编队队形。因此问题四要求调整算法适用于其他编队队形，证明算法的泛化性。由于问题三中我们的优化后模型具有十分良好的普适性，因此无需太大改动就可以适用于任何编队队形。下面以问题四给出的编队队形为例，通过求解该例子证明正确性。

在实际情况下，无人机具体位置是已知的[4]。但是问题二并没有给出初始位置，因此我们需要进行数据仿真。根据问题三的数据，计算 student 分布，通过随机采样，得到初始位置的估计。

5.4.1. 坐标系的建立

在问题四的例子中，无人机集群保持锥形编队队形，我们认为编号为 FY02-FY15 的无人机始终配合领头无人机 FY01 保持队形，因此我们以无人机 FY01 作为坐标原点，以该无人机航向作为 x 轴正方向建立平面直角坐标系。

建立直角坐标系后，我们发现当无人机均处于理想位置时，如 FY08,FY09,FY10 这样的无人机作为发射信号的无人机给 FY07 无人机时，由于四个无人机共线，且所形成的直线会垂直于 y 轴，会导致无法得到三个方向信息，也无法运用问题一所求得的定位模型进行定位，因此我们在运用柱搜索进行枚举时，采取舍弃这类发射接收信号无人机组合，而选取其他符合要求的组合进行排序，求得最优解。

5.4.2. 初始位置的仿真

对于编号为 FY0j 的无人机，在极坐标系中，无人机所处位置为 $O_{(FY0j)}=(\rho_j,\ \theta_j)$；

在直角坐标系中，有 $O'_{FY0j}=(x_j,\ y_j)=(\rho_j\cos(\theta_j),\ \rho_j\sin(\theta_j))$，

对于编号为 FY0j 的无人机，其横坐标与理想位置的差值为

$x_j^{'} = \rho_j\cos(\theta_j) - 100\times\cos(40\times j)$，同理可得到其纵坐标与理想位置的差值为 $y_j^{'} = \rho_j\sin(\theta_j) - 100\times\sin(40\times j)$；横纵坐标（统称为坐标）与理想位置的差的样本均值分别为 $\overline{H} = \dfrac{\sum_{j=0}^{9}(x_j^{'} + y_j^{'})}{20}$。

我们将横纵坐标放在一个样本中，求样本标准差，表示的是对坐标的样本标准差估计 $S = \sqrt{\dfrac{\sum_{j=0}^{20}(x_j^{'} - \overline{H})^2}{19}} = 5.1006113$，因此采样服从分布 t（19），则 $S\times t(19)$ 分布为对理想位置添加的误差分布。

5.4.3. 模型的求解

当无人机集群保持锥形编队队形时，仍考虑纯方位无源定位的情形，由于问题三中的优化后模型具有十分良好的普适性，因此我们在初始时刻按照上述初始误差估计方法，按照 $S\times t(19)$ 分布，随机采样对理想位置添加的误差分布，得到初始位置的估计，将该模型推广至问题四，运用该模型解决锥形编队队形的队形保持问题。

根据模型，可知每次调整所选择的发射信号的无人机组合如表 3 所示（详见附录参考代码四）：

表 3 在有随机误差的情况下发射信号无人机组合及误差

第 i 次调整	在该位置上发射信号的无人机组合	该位置的误差
0	(0, 4, 6, 11)	6.0150
1	(0, 2, 7, 12)	3.0757
2	(0, 3, 11, 13)	2.3080
3	(0, 5, 7, 13)	1.3209
4	(0, 4, 11)	0.9789
5	(0, 8, 13)	0.8120

6	(0, 1, 3, 11)	0.9170
7	(0, 2, 3, 4)	1.0580
8	(0, 4, 9)	0.9241
...	...	...

由表 3 可得，误差 w2 与调整次数 i 之间的关系如下：

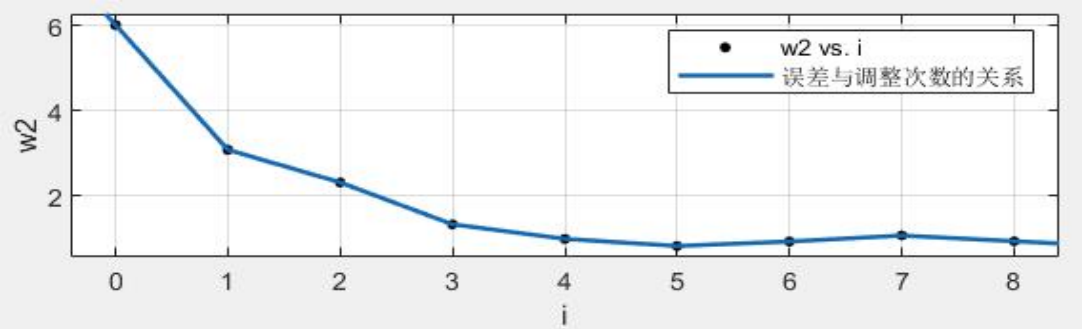

图 15 误差 w2 与调整次数 i 的关系

与问题三同理，根据算法所求得的结果，利用 MATLAB,我们可以得到每次调整后无人机所在位置如图 16 所示：

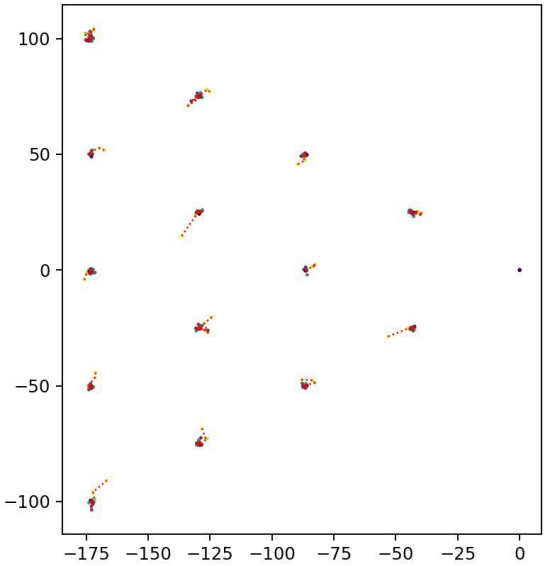

图 16 无人机调整轨迹

6. 模型评价与推广

6.1. 模型的优点

（1）问题一中建立的无人机定位模型原理简单、清晰易懂，该定位模型以及相应的求解算法能够得出无模型误差的定位结果，体现了模型的实用性和算法的有效性。

（2）问题二中的模型利用临界值来划分偏差范围的大小，在一定偏差范围内三架无人机即可实现有效定位，而利用四架无人机定位时可以不考虑偏差，准确得到被动接收信号无人机的实际坐标。

（3）问题三中首先考虑了无误差的优化模型，之后考虑了实际飞行过程中，无人机惯性、空气阻力以及其他障碍物等因素对无人机位置调整的影响。此外，本文采用柱搜索求解，扩大了搜索空间，兼顾了模型的效率和性能。

（4）问题四中运用的模型是问题三中模型的推广，该模型不仅适用于圆形编队，还适用于锥形编队等多种不同的编队类型，具备一定的普适性。

6.2. 模型的缺点

（1）问题一中的模型主要的缺陷在于，该模型在求解的过程中会出现两个可能的解，只能结合编队的轨迹进行取舍，在某些情况下这两点较难区分。

（2）问题二中的模型，由于问题的对称性，交集中必定存在两个沿 x 轴对称的点，无论选取多少架无人机都不可避免。只有假设飞机纵坐标正负性不变，才能准确定位。

（3）问题三中的模型未从初始的方位角和无人机运动规律等角度考虑其中可能对无人机调整产生影响的信息，将来可以根据实际情况优化。

7. 参考文献

[1] 范齐楠,孔存良,杨麟儿,杨尔弘.基于 BERT 与柱搜索的中文释义生成[J].中文信息学报,2021,35(11):80-90.

[2] 程川. 基于柱搜索和神经网络的组块分析研究[D].南京大学,2016.

[3] 朱重儒. 低轨卫星无源定位关键技术研究[D].电子科技大学,2022.

[4] 任立超. 通信辐射源无源定位算法的精度分析[D].西安理工大学,2021.

[5] 传感器知识资讯. 高精度倾角传感器：AIS2000 超高精度倾角传感器 最高精度 0.002° 低温漂 高精度[J]. https://www.ceomba.cn/4848.html, 2021.

[6] 与非网. 无人机如何知道自己的位置？悬停精度究竟如何？[J]. https://www.eefocus.com/sensor/442715, 2019.

8. 附录

8.1. 使用软件名称

Python、MATLAB、Excel、

8.2. 代码

（1）参考代码一

```
# 功能：给定三个发射坐标点，三个接收的两两角（天然满足 a1+a2=a3），定位接收者的点
from math import *
from lib import *
from pos1 import *
from _1_1_1 import main as _1_1_1
# 弦的线段两点 A,B，圆周角 a，设圆心是 Q
# 输入：A,B,a
# 输出：[(Q1,r),(Q2,r)]
from _1_1_2 import main as _1_1_2
# 两圆的圆心半径 o1(O1[0],O1[1],r1),o2(O2[0],O2[1],r2)，求交点[D1, D2]
# 输入:o1,o2
# 输出：[D1, D2]
from _1_1_3 import main as _1_1_3
# 3 发射：ABC        1 接收：D     余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC



# 给定三个发射坐标点 A(x1,y1)B(x2,y2)C(x3,y3)，三个接收的两两角 aAB aAC aBC，求满足的点 D
# 输入：A,B,C,aAB,aAC,aBC
```

```
# 输出：[D1,D2...]（可能多个满足）
def main(A, B, C, aAB, aAC, aBC):
    # if simp(A,B) or simp(A,C) or simp(B,C):
    #     print(A,B,C)
    # 得到两两夹角的可能圆，有上下两个
    circle_list = [_1_1_1(A, B, aAB), _1_1_1(A, C, aAC), _1_1_1(B, C, aBC)]
    # 可行解的列表
    ans_list = []
    # 二进制的 000~111
    for i in range(8):
        # 选取圆
        now_circle_list = [circle_list[0][i & 1], circle_list[1][(i >> 1) & 1],
circle_list[2][(i >> 2) & 1]]
        # 两两圆的公共点
        l1 = _1_1_2(now_circle_list[0], now_circle_list[1])
        l2 = _1_1_2(now_circle_list[0], now_circle_list[2])
        l3 = _1_1_2(now_circle_list[1], now_circle_list[2])
        # 三个圆之间如果有公共点，就 ok
        for P1 in l1:
            for P2 in l2:
                for P3 in l3:
                    # 公共点
                    if simp(P1, P2) and simp(P1, P3):
                        # 附加限制：这个点不是 ABC 之一
                        if pinv(P1, [A, B, C]):
                            continue
                        # 附加限制：这个点不是之前出现过的
                        if pinv(P1, ans_list):
                            continue
                        ans_list.append(P1)
    return ans_list
```

```
if __name__ == '__main__':
    A = (0.0, 0.0)
    B = (100.0, 0.0)
    C = (17.364606251646684, 98.47495514978128)
    # aAB, aAC, aBC = 0.8738267244276083, 0.17483672782800355, 0.6989899965996057
    D = (17.365058742962116, -98.47873509355637)
    aAB, aAC, aBC = _1_1_3(A, B, C, D)
    print('求解：', main(A, B, C, aAB, aAC, aBC))
    # print('真实：', D)
# 功能:已知弦和圆周角，求圆的方程（算出公式了）
from math import *
from lib import *
from pos1 import *

# 弦的线段两点 A,B，圆周角 a，设圆心是 Q
# 输入：A,B,a
# 输出：[(Q1,r),(Q2,r)]
def main(A, B, a):
    # if simp(A,B):
    #     print(A,B)
    # 把公式实现
    AB = dis(A, B)
    AD = AB / 2
    r = AD / sin(a)
    _AB = vinit(A,B)
    th = K(_AB,(1,0))
    b = pi/2 - a
    Q1 = (A[0] + r * cos(th + b), A[1] + r * sin(th + b))
    Q2 = (A[0] + B[0] - Q1[0], A[1] + B[1] - Q1[1])
    return [(Q1, r), (Q2, r)]
```

```
if __name__ == '__main__':
    A = (6, 0)
    B = (2, 0)
    a = 0.17985349979247786
    print(main(A, B, a))
# 功能：已知两圆，求交点
from math import *
from lib import *
from pos1 import *

# 两圆的圆心半径 o1(O1[0],O1[1],r1),o2(O2[0],O2[1],r2)，求交点[D1, D2]
# 输入:o1,o2
# 输出：[D1, D2]
def main(o1, o2):
    # 把公式实现
    O1 = o1[0]
    r1 = o1[1]
    O2 = o2[0]
    r2 = o2[1]
    d = dis(O1, O2)
    if d > (r1 + r2) or d < (r1 - r2):
        return []
    a = costh(r2, r1, d)
    _O1O2 = vinit(O1, O2)
    th = K(_O1O2, (1, 0))
    D1 = (O1[0] + r1 * cos(th + a), O1[1] + r1 * sin(th + a))
    if abs(a) < 1e-10:
        return [D1]
    D2 = (O1[0] + r1 * cos(th - a), O1[1] + r1 * sin(th - a))
    return [D1, D2]
```

```
if __name__ == '__main__':
    O1 = (0, 0)
    r1 = 2
    O2 = (2, 2)
    r2 = 2
    o1 = (O1, r1)
    o2 = (O2, r2)
    print(main(o1, o2))
# 功能：已知 3 发射 1 接收，余弦定理求 3 角度。用于造数据和推算
from math import *
from lib import *
from pos1 import *

# 3 发射：ABC        1 接收：D     余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC
def main(A, B, C, D):
    AB = dis(A, B)
    AC = dis(A, C)
    AD = dis(A, D)
    BC = dis(B, C)
    BD = dis(B, D)
    CD = dis(C, D)
    aAB = costh(AB, AD, BD)
    aAC = costh(AC, AD, CD)
    aBC = costh(BC, BD, CD)
    return aAB, aAC, aBC


if __name__ == '__main__':
    A = (0, 0)
    B = (2, 0)
    C = (0, 2)
```

```
    D = (-1, -1)
    print(main(A, B, C, D))
```

（2）参考代码二

```
# 功能：给出 2 个已知的发射者 01 和 2 个未知的发射者 xy 的 5 个角度(用不到 axy)，推测两个编号，定位接收者位置
from lib import *
from _1_2_1 import main as _1_2_1
# 输入：接收的角度 a01,a0x,a1x
# 输出：[[(),...（x 确定时的解）]*8（2~9 下标对应 0~7）]
from _1_1_3 import main as _1_1_3
from pos1 import *

# 3 发射：ABC        1 接收：D    余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC

# 输入：a01,a0x,a1x,a0y,a1y
# 输出：[(D,bx,by),...]（D 是定位，b 是编号，...多解）
def main(a01, a0x, a1x, a0y, a1y):
    ans_x = _1_2_1(a01, a0x, a1x)
    ans_y = _1_2_1(a01, a0y, a1y)
    ans_list = []
    for i in range(len(ans_x)):
        for j in range(len(ans_y)):
            if i == j:
                continue   # 发射的两个不相同
            # 相容
            for Pi in ans_x[i]:
                for Pj in ans_y[j]:
                    if simp(Pi, Pj):
                        ans_list.append((Pi, i + 2, j + 2))
                        break
```

```
    return ans_list


if __name__ == '__main__':
    # 四个发射者
    A = pos(0)
    B = pos(1)
    C = pos(4)
    D = pos(7)
    # 一个接收者
    E = 100, 100
    # 期望的位置
    bE = 2
    aAB, aAC, aBC = _1_1_3(A, B, C, E)
    aAB, aAD, aBD = _1_1_3(A, B, D, E)
    ans = main(aAB, aAC, aBC, aAD, aBD)
    print(ans)
    dmi = inf
    # 找到实际与理想距离最小的，作为实际位置
    for i in ans:
        if dis(i[0],  pos(bE)) < dmi:
            pmi = i
            dmi = dis(i[0], pos(bE))
    print('预测未知发射者编号：', pmi[1],pmi[2])
    print('预测接收者位置：', pmi[0])
    print('期望接收者位置：', pos(bE))
    print('偏差距离：', dmi)
# 功能：给出 2 个已知的发射者 01 和 1 个未知的发射者，和接收的角度，得到
当未知发射者为 2~9 时，接收者的定位
# 适用于 3 架无人机，选取最近的作解
from lib import *
from _1_1 import main as _1_1
```

```
# 给定三个发射坐标点 A(x1,y1)B(x2,y2)C(x3,y3)，三个接收的两两角 aAB aAC aBC，求满足的点 D
# 输入：A,B,C,aAB,aAC,aBC
# 输出：[D1,D2...]（可能多个满足）
from _1_1_3 import main as _1_1_3
from pos1 import *

# 3 发射：ABC        1 接收：D     余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC


# 输入：接收的角度 a01,a0x,a1x
# 输出：[[(),...（x 确定时的解）]*8（2~9 下标对应 0~7）]
def main(a01, a0x, a1x):
    ans_list = []
    P0 = pos(0)
    P1 = pos(1)
    for x in range(2, 10):
        Px = pos(x)
        ans_list.append(_1_1(P0, P1, Px, a01, a0x, a1x))
    return ans_list


# 从结果中找最近的点作为答案返回
def findans(ans, bE):
    E = pos(bE)
    dmi = inf
    # 找到实际与理想距离最小的，作为实际位置
    for i in range(len(ans)):
        if i + 2 == bE:
            continue  # 自己知道自己不是发射
```

```
            for j in ans[i]:
                if dis(j, E) < dmi:
                    bmi = i + 2
                    pmi = j
                    dmi = dis(j, E)
        return bmi, pmi, dmi


# 测试用
def test(fr=3, bE=5, D=pos(5)):
        # 三个发射者
        A = pos(0)
        B = pos(1)
        C = pos(fr)
        # D 接收者实际位置
        # D 的编号 bE
        # bE = bE
        # D = D
        aAB, aAC, aBC = _1_1_3(A, B, C, D)
        ans = main(aAB, aAC, aBC)
        # print(ans)
        bmi, pmi, dmi = findans(ans, bE)
        return fr, bmi, bE, pmi, dmi


if __name__ == '__main__':
        D = tpt(pos(5), (25,0))
        ret = test(D=D)

        print('实际未知发射者编号：', ret[0])
        print('预测未知发射者编号：', ret[1])
        print('期望接收者编号：', ret[2])
        print('期望接收者位置：', pos(ret[2]))
```

```
    print('预测接收者位置：', ret[3])
    print('偏差距离：', ret[4])
# 功能：给定三个发射者 01x 和一个接收者 D，求位置判断正确的临界
from lib import *
from _1_2_1 import main as _1_2_1
# 输入：接收的角度 a01,a0x,a1x
# 输出：[[(),...（x 确定时的解）]*8（2~9 下标对应 0~7）]
from _1_2_1 import findans
# 从结果中找最近的点作为答案返回
from _1_2_1 import test
# 测试用
from pos1 import *



# 给定实际发射 0,1,fr 和接收 to，在 to 的角度 th 方向的 mid 长度看是否 3 个定位可行
def ok3(fr, to, th, mid):
    # D 接收者实际位置
    D = tpt(pos(to),cotran((mid,th)))
    ret = test(fr, to, D)
    # try:
    #     ret=test(fr,to,D)
    # except:
    #     print(fr, to, D)
    #     ret = test(fr, to, D)
    return simp(D,ret[3])



# 给定实际发射 0,1,fr 和接收 to，在 to 的角度 th 方向找到临界值
def jgth(fr, to, th):
```

```
    l=0
    r=25
    while (r-l)>eps:
        mid = (r+l)/2
        if ok3(fr,to,th,mid):
            l=mid
        else:
            r=mid
    return l


# 给定实际发射 0,1,fr 和接收 to，在角度[be,ed]分 nt(>=2)个方向，得到各方向的临界值，返回最小值
# 输入:fr,to,nt,be,ed
# 输出:mi_list[(r,th)...],ep【精度】
def rangejudge(fr, to, be=0, ed=2 * pi, nt=4):
    if be==0 and ed == 2*pi:
        ed-=    (2 * pi)/nt
    ep = (ed - be) / (nt - 1)
    ang = [be + ep*i for i in range(nt)]
    mi_list = []
    for th in ang:
        ret=jgth(fr, to, th)
        mi_list.append((ret, th))
    return mi_list,ep


# 给定实际发射 0,1,fr 和接收 to，返回最小值，各方向临界值
# 输入：fr,to
# 输出：Pmi,mi_list[(r,th)...]
def judge(fr,to):
    lth=0
    rth=2*pi
```

```
    Pmi=None
    mi=inf
    mi_list=[]
    while rth-lth>eps:
        ret=rangejudge(fr,to,lth,rth)
        ep=ret[1]
        ret_list=ret[0]
        for i in ret_list:
            mi_list.append(i)
            if i[0]<mi:
                Pmi=i
                mi=i[0]
        rth=Pmi[1]+ep
        lth=Pmi[1]-ep
    return Pmi,mi_list



# 判断这个队形下的最小值
# 先枚举每个队形， 求出临界的结果，再求其中最小值
# 输出：最小临界值 dmi,最小临界极坐标 Pmi，发射 fr,接收 to，临界点列表 tr
def main():
    dmi = inf
    Pmi = None
    fr = None
    to = None
    tr = None
    for i in range(2, 10):   # 发射
        for j in range(2, 10):   # 接收
            if i == j:
                continue
            # i=3
            # j=5
```

```
                ret = judge(i, j)
                if dmi > ret[0][0]:
                    dmi = ret[0][0]
                    Pmi = ret[0]
                    tr = ret[1]
                    fr = i
                    to = j
    return dmi,Pmi, fr, to, tr


if __name__ == '__main__':
    ans = main()
    print('最小临界值 dmi', ans[0])
    print('最小临界极坐标 Pmi', ans[1])
    print('发射 fr', ans[2])
    print('接收 to', ans[3])
    print('临界点列表 tr', ans[4])

# 功能：画点图
import matplotlib.pyplot as plt
from lib import *
import numpy as np


# 画图
def main():
    # plt.subplot(111,projection='polar')
    Pmi=(16.19873046875, 1.835348977474421)
    Pmi=cotran(Pmi)
    tr = [(24.993896484375, 0.0), (16.5771484375, 1.5707963267948966),
(24.993896484375, 3.141592653589793),
          (24.993896484375, 4.71238898038469), (24.993896484375, 0.0),
(19.989013671875, 1.0471975511965976),
```

(16.56494140625, 2.0943951023931953), (24.993896484375, 3.141592653589793),
(19.989013671875, 1.0471975511965976), (16.241455078125, 1.7453292519943295),
(18.304443359375, 2.443460952792061), (24.993896484375, 3.141592653589793),
(19.989013671875, 1.0471975511965976), (16.766357421875, 1.5126186850617522),
(16.30859375, 1.9780398189269066), (18.304443359375, 2.443460952792061), (17.962646484375, 1.279908118129175),
(16.5283203125, 1.590188874039278), (16.22314453125, 1.9004696299493808),
(16.973876953125, 2.210750385859484), (16.5283203125, 1.5901888740392778), (16.2109375, 1.79704271131268),
(16.351318359375, 2.003896548586082), (16.973876953125, 2.210750385859484),
(16.5283203125, 1.5901888740392778), (16.259765625, 1.728091432221546), (16.204833984375, 1.8659939904038139),
(16.351318359375, 2.003896548586082), (16.259765625, 1.7280914322215457),
(16.204833984375, 1.8200264710097245), (16.2353515625, 1.9119615097979032),
(16.351318359375, 2.003896548586082), (16.22314453125, 1.774058951615635),
(16.19873046875, 1.835348977474421), (16.22314453125, 1.8966390033332068),
(16.2841796875, 1.9579290291919926), (16.22314453125, 1.774058951615635),
(16.204833984375, 1.8149189688548257), (16.204833984375, 1.8557789860940161),
(16.22314453125, 1.8966390033332068), (16.2109375, 1.7944889602352303), (16.204833984375, 1.821728971728024),
(16.204833984375, 1.8489689832208178), (16.2109375, 1.8762089947136116),

```
(16.204833984375, 1.8081089659816272), (16.19873046875,
1.8262689736434896),
(16.19873046875, 1.8444289813053523), (16.204833984375,
1.8625889889672147),
(16.204833984375, 1.8171889698125585), (16.19873046875,
1.8292956415871335),
(16.19873046875, 1.8414023133617083), (16.204833984375,
1.8535089851362834),
(16.204833984375, 1.823242305699846), (16.19873046875,
1.8313134202162293),
(16.19873046875, 1.8393845347326125), (16.204833984375,
1.847455649248996),
(16.19873046875, 1.8272778629580375), (16.19873046875,
1.8326586059689598),
(16.19873046875, 1.838039348979882), (16.19873046875,
1.8434200919908044),
(16.19873046875, 1.8299682344634987), (16.19873046875,
1.8335553964707803),
(16.19873046875, 1.8371425584780616), (16.19873046875,
1.8407297204853432)]
    tr = [cotran(i) for i in tr]
    x = [i[0] for i in tr]
    y = [i[1] for i in tr]
    colors = np.linspace(0, 100, num=len(tr))
    plt.scatter(x, y, c=colors, cmap='viridis')
    plt.scatter([0], [0], s=[100], color='black')
    plt.plot([0, Pmi[0]], [0, Pmi[1]], linewidth=3, color='red')
    plt.show()


if __name__ == '__main__':
    main()

'''
```

最小临界值 dmi 16.19873046875

最小临界极坐标 Pmi (16.19873046875, 1.835348977474421)

发射 fr 5

接收 to 4

临界点列表 tr [(24.993896484375, 0.0), (16.5771484375, 1.5707963267948966), (24.993896484375, 3.141592653589793), (24.993896484375, 4.71238898038469), (24.993896484375, 0.0), (19.989013671875, 1.0471975511965976), (16.56494140625, 2.0943951023931953), (24.993896484375, 3.141592653589793), (19.989013671875, 1.0471975511965976), (16.241455078125, 1.7453292519943295), (18.304443359375, 2.443460952792061), (24.993896484375, 3.141592653589793), (19.989013671875, 1.0471975511965976), (16.766357421875, 1.5126186850617522), (16.30859375, 1.9780398189269066), (18.304443359375, 2.443460952792061), (17.962646484375, 1.279908118129175), (16.5283203125, 1.590188874039278), (16.22314453125, 1.9004696299493808), (16.973876953125, 2.210750385859484), (16.5283203125, 1.5901888740392778), (16.2109375, 1.79704271131268), (16.351318359375, 2.003896548586082), (16.973876953125, 2.210750385859484), (16.5283203125, 1.5901888740392778), (16.259765625, 1.728091432221546), (16.204833984375, 1.8659939904038139), (16.351318359375, 2.003896548586082), (16.259765625, 1.7280914322215457), (16.204833984375, 1.8200264710097245), (16.2353515625, 1.9119615097979032), (16.351318359375, 2.003896548586082), (16.22314453125, 1.774058951615635), (16.19873046875, 1.835348977474421), (16.22314453125, 1.8966390033332068), (16.2841796875, 1.9579290291919926), (16.22314453125, 1.774058951615635), (16.204833984375, 1.8149189688548257), (16.204833984375, 1.8557789860940161), (16.22314453125, 1.8966390033332068), (16.2109375, 1.7944889602352303), (16.204833984375, 1.821728971728024), (16.204833984375, 1.8489689832208178), (16.2109375, 1.8762089947136116), (16.204833984375, 1.8081089659816272), (16.19873046875, 1.8262689736434896), (16.19873046875, 1.8444289813053523), (16.204833984375, 1.8625889889672147), (16.204833984375, 1.8171889698125585), (16.19873046875, 1.8292956415871335), (16.19873046875, 1.8414023133617083), (16.204833984375, 1.8535089851362834), (16.204833984375, 1.823242305699846), (16.19873046875, 1.8313134202162293), (16.19873046875, 1.8393845347326125), (16.204833984375, 1.847455649248996), (16.19873046875, 1.8272778629580375), (16.19873046875,

1.8326586059689598), (16.19873046875, 1.838039348979882), (16.19873046875, 1.8434200919908044), (16.19873046875, 1.8299682344634987), (16.19873046875, 1.8335553964707803), (16.19873046875, 1.8371425584780616), (16.19873046875, 1.8407297204853432)]

'''

（3）参考代码三

```
# 功能：柱搜索
# （初始状态 S0）初始时刻无人机的位置略有偏差，
# （择优返回）多次调整，
# （动作 A1）每次选择编号为 FY00 的无人机和圆周上最多 3 架无人机发射信号，
# （备选状态）其余无人机根据接收到的方向信息，调整到理想位置（每次调整的时间忽略不计），
# 使得 9 架无人机最终均匀分布在某个圆周上。
from lib import *
from _1_1 import main as _1_1
# 给定三个发射坐标点 A(x1,y1)B(x2,y2)C(x3,y3)，三个接收的两两角 aAB aAC aBC，求满足的点 D
# 输入：A,B,C,aAB,aAC,aBC
# 输出：[D1,D2...]（可能多个满足）
from _1_1_3 import main as _1_1_3
from pos1 import *

# 3 发射：ABC        1 接收：D     余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC


# 状态数
ns = 10
```

```
# 给定接收者 i 和发射的飞机三元组 a，求预测定位到理想位置的向量 iv 返回
# 3 个发射者与该飞机点得到仿真角度_1_1_3
# 仿真角度代入理想的发射者位置得到预测定位（2 个，取离理想位置较近的那一个）_1_1
# 求预测定位到理想位置的向量 iv 返回
# 输入：pos_list, i, a
# 输出：iv
def getiv(pos_list, i, a, N2):
    _3a = _1_1_3(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
    # 给角度 a01,a02,a12 一个正态分布(0,N2)的误差，表示传感器接收夹角的随机误差
    a01, a02, a12 = anor(_3a, N2)
    Piok = _1_1(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    dmi = inf
    if not Piok:
        print(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
        print(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    for Pi in Piok:
        if dis(Pi, pos(i)) < dmi:
            dmi = dis(Pi, pos(i))
            imi = Pi
    vi = tst(pos(i), imi)
    return vi


# 给定实际 pos_list 和发射的飞机三元组 a，给出其他飞机离预测理想位置的距离 tov
# 对每一个飞机，如果在 a 中，就是(0,0)表示不知道，否则用 getiv 求预测定位到理想位置的向量
# 输入：pos_list,a
# 输出：tov
```

```
def tranv(pos_list, a, N2):
    ans_list = []
    for i in range(ns):
        if i in a or inlinel([pos(i), pos(a[0]), pos(a[1]), pos(a[2])]):
            i_v = (0, 0)
        else:
            i_v = getiv(pos_list, i, a, N2)
        ans_list.append(i_v)
    return tuple(ans_list)


# 给定当前状态 q 与动作 a，给出理想位置 ex 与加权参数 beta，求下一状态 s
# 模拟得到角度——last_pos 和 a 得接收角度
# 猜测到理想位置的向量——按理想的发射位置，3 机此，4 机选 01x 两个平均
# 决定速度和长度——加权求得
# 输入：q, a, ex, beta
# 输出：s
def tran(q, a, ex, alpha, beta, bm, N2):
    last_pos = q[1]
    last_v = q[2]
    if len(a) == 3:
        tov = tranv(last_pos, a, N2)
    else:
        tov = tpt(tmn(tranv(last_pos, (a[0], a[1], a[3]), N2), 0.5), tmn(tranv(last_pos,
(a[0], a[1], a[2]), N2), 0.5))
        tov = list(tov)
        tov[a[2]] = tov[a[3]] = (0, 0)
        tov = tuple(tov)

    t1 = tmn(last_v, 1 - beta)
    t2 = tmn(tov, beta)
    v = tmn(tpt(t1, t2), 1 / (1 - (1 - beta) ** bm))
    va = tmn(v, alpha)
```

```
        now_pos = tpt(last_pos, va)
        g = grade(ex, now_pos)
        return g, now_pos, v

# 柱搜索的动作集合
# 输出：动作集合
def get_actions():
        ans = []
        for i in range(2, ns):
                ans.append((0, 1, i))
                for j in range(i + 1, ns):
                        ans.append((0, 1, i, j))
        return ans


# 柱搜索迭代优化
# 柱搜索的状态(分数，实际点元组((,)...),实际点速度向量元组((,)...))
# 输入：ex[理想点元组], raw[实际点元组]，epoch 迭代的轮数，width 保留的较优状态个数，beta 加权参数
# 输出：[状态...]（从 0 到 epoch）
def bean(ex, raw, epoch=5, width=30, alpha=1, beta=1, N1=0, N2=0):
        # M:(子状态，(父状态,动作))的字典。
        M = dict()
        # Q:基础状态列表
        Q = [(grade(ex, raw), raw, tuple((0, 0) for i in range(len(raw))))]
        # A:动作列表
        A = get_actions()
        for i in range(1, epoch + 1):
                # S:备选(子状态，(父状态,动作))列表
                S = []
                for q in Q:
                        # 给 q[1]一个正态分布(0,N1)的误差，表示上一次到这一次接收信
```

号间的随机误差（除 0 号）

```
                # q 用来寻找父节点
                q1_ = tpn(tnor(q[1], N1), nor(N1))
                q1_ = list(q1_)
                q1_[0] = (0, 0)
                q1_ = tuple(q1_)
                q_ = (q[0], q1_, q[2])
                for a in A:
                    S.append((tran(q_, a, ex, alpha, beta, i, N2), (q, a)))
            S.sort(key=xkey)
            S = S[:width]
            for s in S:
                M[s[0]] = s[1]
            Q = [s[0] for s in S]
        ans_list = [(Q[0], tuple())]
        while ans_list[-1][0] in M:
            ans_list.append(M[ans_list[-1][0]])
        ans_list = ans_list[::-1]
        return ans_list


if __name__ == '__main__':
        ex = tuple(pos(i) for i in range(ns))
        raw = (
            (0, 0),
            (100, 0),
            (98, 40.10),
            (112, 80.21),
            (105, 119.75),
            (98, 159.86),
            (112, 199.96),
            (105, 240.07),
            (98, 280.17),
```

```
            (112, 320.28)
        )
        raw = tuple(cotran((x[0], angtran(x[1]))) for x in raw)
        epoch, width, alpha, beta, N1, N2 = 8, 20, 1, 1, 0, 0
        ans = bean(ex, raw, epoch=epoch, width=width, alpha=alpha, beta=beta, N1=N1,
N2=N2)
        print(ans)
        to_excel(ans,'无误差.xlsx')



# 参数：epoch*width，alpha, beta
'''实验：
epoch     width     alpha     beta      分数
10        10        1         0.7       0.007098
10        10        1         0.85      0.000260
10        10        1         0.95      1.1845635165778284e-06
10        10        1         0.99      6.312782312418901e-09
10        10        0.95      1         8.961979455147747e-10
10        10        0.95      0.99      3.700949506836242e-09
10        10        0.99      0.99      3.6557168402291616e-09
10        10        0.99      1         2.537993871872589e-11
10        10        1         1         4.147130052742561e-14
10        3         1         1         4.8700694537728296e-14
3         3         1         1         0.002795536274978537
3         3         0.99      0.99      0.013538078676633444
3         10        1         1         0.002124570310733179
3         20        1         1         0.0006690236346775145
3         100       1         1         0.00029917302636744346

'''
# 功能：柱搜索（考虑误差版）
# （初始状态 S0）初始时刻无人机的位置略有偏差，
# （择优返回）多次调整，
```

```
# （动作 A1）每次选择编号为 FY00 的无人机和圆周上最多 3 架无人机发射信号，
# （备选状态）其余无人机根据接收到的方向信息，调整到理想位置（每次调整的时间忽略不计），
# 使得 9 架无人机最终均匀分布在某个圆周上。
from lib import *
from _1_1 import main as _1_1
# 给定三个发射坐标点 A(x1,y1)B(x2,y2)C(x3,y3)，三个接收的两两角 aAB aAC aBC，求满足的点 D
# 输入：A,B,C,aAB,aAC,aBC
# 输出：[D1,D2...]（可能多个满足）
from _1_1_3 import main as _1_1_3
from pos1 import *


# 3 发射：ABC        1 接收：D     余弦定理求 3 角度  aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC

# 状态数
ns = 10

# 给定接收者 i 和发射的飞机三元组 a，求预测定位到理想位置的向量 iv 返回
# 3 个发射者与该飞机点得到仿真角度_1_1_3
# 仿真角度代入理想的发射者位置得到预测定位（2 个，取离理想位置较近的那一个）_1_1
# 求预测定位到理想位置的向量 iv 返回
# 输入：pos_list, i, a
# 输出：iv
def getiv(pos_list, i, a, N2):
    _3a = _1_1_3(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
```

```
    # 给角度 a01,a02,a12 一个正态分布(0,N2)的误差，表示传感器接收夹角的随机误差
    a01, a02, a12 = anor(_3a, N2)
    # A,B,C=pos(a[0]),pos(a[1]),pos(a[2])
    # if simp(A,B) or simp(A,C) or simp(B,C):
    #     print(A,B,C)
    Piok = _1_1(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    dmi = inf
    # if not Piok:
    #     print(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
    #     print(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    for Pi in Piok:
        if dis(Pi, pos(i)) < dmi:
            dmi = dis(Pi, pos(i))
            imi = Pi
    vi = tst(pos(i), imi)
    return vi


# 给定实际 pos_list 和发射的飞机三元组 a，给出其他飞机离预测理想位置的距离 tov
# 对每一个飞机，如果在 a 中，就是(0,0)表示不知道，否则用 getiv 求预测定位到理想位置的向量
# 输入：pos_list,a
# 输出：tov
def tranv(pos_list, a, N2):
    ans_list = []
    for i in range(ns):
        if i in a or inlinel([pos(i), pos(a[0]), pos(a[1]), pos(a[2])]):
            i_v = (0, 0)
        else:
            i_v = getiv(pos_list, i, a, N2)
        ans_list.append(i_v)
```

```
    return tuple(ans_list)


# 给定当前状态 q 与动作 a，给出理想位置 ex 与加权参数 beta，求下一状态 s
# 模拟得到角度——last_pos 和 a 得接收角度
# 猜测到理想位置的向量——按理想的发射位置，3 机此，4 机选 01x 两个平均
# 决定速度和长度——加权求得
# 输入：q, a, ex, beta
# 输出：s
def tran(q, a, ex, alpha, beta, bm, N2):
    last_pos = q[1]
    last_v = q[2]
    if len(a) == 3:
        tov = tranv(last_pos, a, N2)
    else:
        tov = tmn(tpt(tpt(tranv(last_pos, (a[0], a[1], a[2]), N2), tranv(last_pos, (a[0],
a[1], a[3]), N2)),
                                tranv(last_pos, (a[0], a[2], a[3]), N2)), 1 / 3)
        tov = list(tov)
        tov[a[1]] = tov[a[2]] = tov[a[3]] = (0, 0)
        tov = tuple(tov)

    t1 = tmn(last_v, 1 - beta)
    t2 = tmn(tov, beta)
    v = tmn(tpt(t1, t2), 1 / (1 - (1 - beta) ** bm))
    va = tmn(v, alpha)
    now_pos = tpt(last_pos, va)
    g = grade(ex, now_pos)
    return g, now_pos, v


# 柱搜索的动作集合
# 输出：动作集合
```

```
def get_actions():
    ans = []
    for i in range(1, ns):
        for j in range(i + 1, ns):
            ans.append((0, i, j))
            for k in range(j + 1, ns):
                ans.append((0, i, j, k))
    return ans


# 柱搜索迭代优化
# 柱搜索的状态(分数，实际点元组((,)...),实际点速度向量元组((,)...))
# 输入：ex[理想点元组], raw[实际点元组]，epoch 迭代的轮数，width 保留的较优状态个数，beta 加权参数
# 输出：[状态...]（从 0 到 epoch）
def bean(ex, raw, epoch=5, width=30, alpha=1, beta=1, N1=0, N2=0):
    # M:(子状态，(父状态,动作))的字典。
    M = dict()
    # Q:基础状态列表
    Q = [(grade(ex, raw), raw, tuple((0, 0) for i in range(len(raw))))]
    # A:动作列表
    A = get_actions()
    for i in range(1, epoch + 1):
        # S:备选(子状态，(父状态,动作))列表
        S = []
        for q in Q:
            # 给 q[1]一个正态分布(0,N1)的误差，表示上一次到这一次接收信号间的随机误差（除 0 号）
            # q 用来寻找父节点
            q1_ = tpn(tnor(q[1], N1), nor(N1))
            q1_ = list(q1_)
            q1_[0] = (0, 0)
```

```
                q1_ = tuple(q1_)
                q_ = (q[0], q1_, q[2])
                for a in A:
                    S.append((tran(q_, a, ex, alpha, beta, i, N2), (q, a)))
            S.sort(key=xkey)
            S = S[:width]
            for s in S:
                M[s[0]] = s[1]
            Q = [s[0] for s in S]
        ans_list = [(Q[0], tuple())]
        while ans_list[-1][0] in M:
            ans_list.append(M[ans_list[-1][0]])
        ans_list = ans_list[::-1]
        return ans_list


if __name__ == '__main__':
    ex = tuple(pos(i) for i in range(ns))
    raw = (
        (0, 0),
        (100, 0),
        (98, 40.10),
        (112, 80.21),
        (105, 119.75),
        (98, 159.86),
        (112, 199.96),
        (105, 240.07),
        (98, 280.17),
        (112, 320.28)
    )
    raw = tuple(cotran((x[0], angtran(x[1]))) for x in raw)
    epoch, width, alpha, beta, N1, N2 = 64, 10, 1, 0.9, 1, 0.00001
    ans = bean(ex, raw, epoch=epoch, width=width, alpha=alpha, beta=beta, N1=N1,
N2=N2)
```

```
    print(ans)
    to_excel(ans, '有误差（历史动量比例误差{}，位置误差{}，角度误差{}）.xlsx'.format(1 - beta, N1, N2))

# 参数：epoch*width，alpha, beta
# 功能：画点图
import matplotlib.pyplot as plt
from lib import *
import numpy as np
import pandas as pd
from pos1 import *

# 画图
def main():
    # 高 宽 写字顺序 1~高*宽
    # plt.subplot(247,projection='polar')
    ns = 10
    df = pd.read_excel('有误差（历史动量比例误差 0.09999999999999998，位置误差 1，角度误差 1e-05）.xlsx')
    # df = pd.read_excel('无误差.xlsx')
    # for i in range(8):
    #     posn = 111
    #     sp = plt.subplot(posn, projection='polar')
    #     if i == 0:
    #         sp.set_title('initial position', y=-0.15, fontsize=14)
    #     else:
    #         sp.set_title('after %d adjustment' % i, y=-0.15, fontsize=14)
    #     rho = [df[x][i] for x in range(2, 2 + ns)]
    #     th = [df[x][i] for x in range(2 + ns, 2 + 2 * ns)]
    #     colors = np.linspace(50, 100, num=ns)
    #     plt.scatter(th, rho, c=colors, cmap='viridis')
    #     plt.show()
```

```
plt.subplot(111, projection='polar')
for i in range(ns):
    # plt.subplot(111, projection='polar')
    rho = df[i+2]
    th = df[i+2+ns]
    th = [(x if x<180 else x-360) for x in th]
    th = [angtran(x) for x in th]
    colors = np.linspace(100, 0, num=len(df))
    s = np.linspace(1, 1, num=len(df))
    plt.scatter(th, rho, c=colors, s=s, cmap='viridis')
    plt.plot(th, rho, "r:", linewidth=1)
    # plt.show()
plt.show()
for i in range(1):
    plt.subplot(111)
    i = 3
    rho = df[i + 2]
    th = df[i + 2 + ns]
    th = [angtran(x) for x in th]
    Ps = [(a[0],a[1])for a in zip(rho,th)]
    Ps = [cotran(x) for x in Ps]
    Ps = [tst(x,pos(i)) for x in Ps]
    Ps = Ps[2:]
    x = [a[0] for a in Ps]
    y = [a[1] for a in Ps]
    colors = np.linspace(100, 0, num=len(Ps))
    s = np.linspace(20, 10, num=len(Ps))
    plt.scatter(x, y, c=colors, s=s, cmap='viridis')
    plt.scatter([0], [0], c=[0])
    # plt.plot(x, y, "r:", linewidth=2)
    plt.show()
# plt.show()
#
# print(df)
```

```
    # colors = np.linspace(0, 100, num=len(tr))
    # plt.scatter(y, x, c=colors, cmap='viridis')
    # plt.scatter([0], [0], s=[100], color='black')
    # plt.plot([0, Pmi[1]], [0, Pmi[0]], linewidth=3, color='red')


if __name__ == '__main__':
    main()
```

（4）参考代码四

```
# 功能：柱搜索通解（考虑误差版）
# （初始状态 S0）初始时刻无人机的位置略有偏差，
# （择优返回）多次调整，
# （动作 A1）每次选择编号为 FY00 的无人机和圆周上最多 3 架无人机发射信号，
# （备选状态）其余无人机根据接收到的方向信息，调整到理想位置（每次调整的时间忽略不计），
# 使得 9 架无人机最终均匀分布在某个圆周上。
from lib import *
from _1_1 import main as _1_1
# 给定三个发射坐标点 A(x1,y1)B(x2,y2)C(x3,y3)，三个接收的两两角 aAB aAC aBC，求满足的点 D
# 输入：A,B,C,aAB,aAC,aBC
# 输出：[D1,D2...]（可能多个满足）
from _1_1_3 import main as _1_1_3
from pos2 import *

# 3 发射：ABC        1 接收：D    余弦定理求 3 角度 aAB aAC aBC
# 输入：A,B,C,D
# 输出：aAB, aAC, aBC
```

```
# 状态数
ns = 15


# 给定接收者 i 和发射的飞机三元组 a，求预测定位到理想位置的向量 iv 返回
# 3 个发射者与该飞机点得到仿真角度_1_1_3
# 仿真角度代入理想的发射者位置得到预测定位（2 个，取离理想位置较近的那一个）_1_1
# 求预测定位到理想位置的向量 iv 返回
# 输入：pos_list, i, a
# 输出：iv
def getiv(pos_list, i, a, N2):
    _3a = _1_1_3(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
    # 给角度 a01,a02,a12 一个正态分布(0,N2)的误差，表示传感器接收夹角的随机误差
    a01, a02, a12 = anor(_3a, N2)
    # A,B,C=pos(a[0]),pos(a[1]),pos(a[2])
    # if simp(A,B) or simp(A,C) or simp(B,C):
    #     print(A,B,C)
    Piok = _1_1(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    if not Piok:
        print(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
        print(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
        _3a = _1_1_3(pos_list[a[0]], pos_list[a[1]], pos_list[a[2]], pos_list[i])
        # 给角度 a01,a02,a12 一个正态分布(0,N2)的误差，表示传感器接收夹角的随机误差
        a01, a02, a12 = anor(_3a, N2)
        # A,B,C=pos(a[0]),pos(a[1]),pos(a[2])
        # if simp(A,B) or simp(A,C) or simp(B,C):
        #     print(A,B,C)
        Piok = _1_1(pos(a[0]), pos(a[1]), pos(a[2]), a01, a02, a12)
    dmi = inf
```

```
    for Pi in Piok:
        if dis(Pi, pos(i)) < dmi:
            dmi = dis(Pi, pos(i))
            imi = Pi
    vi = tst(pos(i), imi)
    return vi


# 给定实际 pos_list 和发射的飞机三元组 a，给出其他飞机离预测理想位置的距离 tov
# 对每一个飞机，如果在 a 中，就是(0,0)表示不知道，否则用 getiv 求预测定位到理想位置的向量
# 输入：pos_list,a
# 输出：tov
def tranv(pos_list, a, N2):
    ans_list = []
    for i in range(ns):
        if i in a or inlinel([pos(i), pos(a[0]), pos(a[1]), pos(a[2])]):
            i_v = (0, 0)
        else:
            i_v = getiv(pos_list, i, a, N2)
        ans_list.append(i_v)
    return tuple(ans_list)


# 给定当前状态 q 与动作 a，给出理想位置 ex 与加权参数 beta，求下一状态 s
# 模拟得到角度——last_pos 和 a 得接收角度
# 猜测到理想位置的向量——按理想的发射位置，3 机此，4 机选 01x 两个平均
# 决定速度和长度——加权求得
# 输入：q, a, ex, beta
# 输出：s
def tran(q, a, ex, alpha, beta, bm, N2):
```

```
        last_pos = q[1]
        last_v = q[2]
        if len(a) == 3:
            tov = tranv(last_pos, a, N2)
        else:
            tov = tmn(tpt(tpt(tranv(last_pos, (a[0], a[1], a[2]), N2), tranv(last_pos, (a[0],
a[1], a[3]), N2)),
                                  tranv(last_pos, (a[0], a[2], a[3]), N2)), 1 / 3)
            tov = list(tov)
            tov[a[1]] = tov[a[2]] = tov[a[3]] = (0, 0)
            tov = tuple(tov)

        t1 = tmn(last_v, 1 - beta)
        t2 = tmn(tov, beta)
        v = tmn(tpt(t1, t2), 1 / (1 - (1 - beta) ** bm))
        va = tmn(v, alpha)
        now_pos = tpt(last_pos, va)
        g = grade(ex, now_pos)
        return g, now_pos, v


# 柱搜索的动作集合
# 输出：动作集合
def get_actions():
        ans = []
        for i in range(1, ns):
            for j in range(i + 1, ns):
                ans.append((0, i, j))
                for k in range(j + 1, ns):
                    ans.append((0, i, j, k))
        return ans
```

```
# 柱搜索迭代优化
# 柱搜索的状态(分数，实际点元组((,)...),实际点速度向量元组((,)...))
# 输入：ex[理想点元组], raw[实际点元组]，epoch 迭代的轮数，width 保留的较优状态个数，beta 加权参数
# 输出：[状态...]（从 0 到 epoch）
def bean(ex, raw, epoch=5, width=30, alpha=1, beta=1, N1=0, N2=0):
    # M:(子状态，(父状态,动作))的字典。
    M = dict()
    # Q:基础状态列表
    Q = [(grade(ex, raw), raw, tuple((0, 0) for i in range(len(raw))))]
    # A:动作列表
    A = get_actions()
    for i in range(1, epoch + 1):
        # S:备选(子状态，(父状态,动作))列表
        S = []
        for q in Q:
            # 给 q[1]一个正态分布(0,N1)的误差，表示上一次到这一次接收信号间的随机误差（除 0 号）
            # q 用来寻找父节点
            q1_ = tpn(tnor(q[1], N1), nor(N1))
            q1_ = list(q1_)
            q1_[0] = (0, 0)
            q1_ = tuple(q1_)
            q_ = (q[0], q1_, q[2])
            for a in A:
                S.append((tran(q_, a, ex, alpha, beta, i, N2), (q, a)))
        S.sort(key=xkey)
        S = S[:width]
        for s in S:
            M[s[0]] = s[1]
        Q = [s[0] for s in S]
    ans_list = [(Q[0], tuple())]
```

```
        while ans_list[-1][0] in M:
            ans_list.append(M[ans_list[-1][0]])
        ans_list = ans_list[::-1]
        return ans_list


if __name__ == '__main__':
        ex = tuple(pos(i) for i in range(ns))
        # t 分布 t(19)，样本方差 s=5.100611312091458
        s = 5.100611312091458
        n = 19
        raw = tt(ex, s, n)
        raw = list(raw)
        raw[0] = (0, 0)
        raw = tuple(raw)
        epoch, width, alpha, beta, N1, N2 = 16, 5, 1, 1, 1, 0.00001
        ans = bean(ex, raw, epoch=epoch, width=width, alpha=alpha, beta=beta, N1=N1,
N2=N2)
        print(ans)
        to_excel(ans, '_2_.xlsx')

# 参数：epoch*width，alpha, beta
# 功能：画点图
import matplotlib.pyplot as plt
from lib import *
import numpy as np
import pandas as pd
from pos2 import *

# 画图
def main():
        # 高 宽 写字顺序 1~高*宽
        # plt.subplot(247,projection='polar')
```

```
    ns = 15
    # df = pd.read_excel('有误差（历史动量比例误差 0.09999999999999998，位置误差 1，角度误差 1e-05）.xlsx')
    df = pd.read_excel('_2_.xlsx')
    # for i in range(8):
    #     posn = 111
    #     sp = plt.subplot(posn, projection='polar')
    #     if i == 0:
    #         sp.set_title('initial position', y=-0.15, fontsize=14)
    #     else:
    #         sp.set_title('after %d adjustment' % i, y=-0.15, fontsize=14)
    #     rho = [df[x][i] for x in range(2, 2 + ns)]
    #     th = [df[x][i] for x in range(2 + ns, 2 + 2 * ns)]
    #     colors = np.linspace(50, 100, num=ns)
    #     plt.scatter(th, rho, c=colors, cmap='viridis')
    #     plt.show()
    plt.subplot(111)
    for i in range(ns):
        x = df[i+2]
        y = df[i+2+ns]
        colors = np.linspace(100, 0, num=len(df))
        s = np.linspace(1, 1, num=len(df))
        plt.scatter(x, y, c=colors, s=s, cmap='viridis')
        plt.plot(x, y, "r:", linewidth=1)
        # plt.show()
    plt.show()
    # for i in range(1):
    #     plt.subplot(111)
    #     i = 3
    #     rho = df[i + 2]
    #     th = df[i + 2 + ns]
    #     Ps = [(a[0],a[1])for a in zip(rho,th)]
    #     Ps = [cotran(x) for x in Ps]
```

```
        #         Ps = [tst(x,pos(i)) for x in Ps]
        #         # Ps = Ps[2:]
        #         x = [a[0] for a in Ps]
        #         y = [a[1] for a in Ps]
        #         colors = np.linspace(100, 0, num=len(Ps))
        #         s = np.linspace(20, 10, num=len(Ps))
        #         plt.scatter(x, y, c=colors, s=s, cmap='viridis')
        #         plt.plot(x, y, "r:", linewidth=2)
        #         plt.show()
        # plt.show()
        #
        # print(df)
        # colors = np.linspace(0, 100, num=len(tr))
        # plt.scatter(y, x, c=colors, cmap='viridis')
        # plt.scatter([0], [0], s=[100], color='black')
        # plt.plot([0, Pmi[1]], [0, Pmi[0]], linewidth=3, color='red')


if __name__ == '__main__':
    main()
# 功能：通过问题一数据，估算 x,y 坐标的 t 分布参数（相当于方差）
from lib import *
from pos1 import *

# 输入：raw 列表
# 输出：ans
def main(raw):
    avg=sum(raw)/len(raw)
    ans2 = sum([(x-avg)**2 for x in raw])/(len(raw)-1)
    ans=ans2**(1/2)
    return ans
```

```
if __name__ == '__main__':
    raw = (
        (0, 0),
        (100, 0),
        (98, 40.10),
        (112, 80.21),
        (105, 119.75),
        (98, 159.86),
        (112, 199.96),
        (105, 240.07),
        (98, 280.17),
        (112, 320.28)
    )
    ex = tuple(pos(i) for i in range(len(raw)))
    raw = tuple(cotran((x[0],angtran(x[1]))) for x in raw)
    raw = tst(raw, ex)
    raw = [x[0] for x in raw]+[x[1] for x in raw]
    ans = main(raw)
    print(ans)
```