基于“三角定位法”的无人机几何定位与位置调整方案

摘要

针对无人机纯方位无源定位问题，设计定位与位置调整方案.在发射信号无人机位置信息不同的条件下，对于偏差无人机，分别给出了相应的精确定位方案；通过“三角定位”、“极坐标迭代修正”等算法，对无人机圆形编队的位置偏差调整模型进行求解.再推广到方阵、锥形阵的无人机编队形态.

问题一第 1 问：为了刻画被动接受信号定位模型，在方向角度信息的不同情景下（见图 1~4），运用了**正弦定理**构建几何关系，分别建立了接收信号无人机的极坐标被动定位模型（见公式 1~14），通过**消元法求解**（见图 5），得到了偏差无人机的解析极坐标，并模拟求解（结果见表 1）.

问题一第 2 问：针对存在发射信号无人机编号未知的情况，**假设**只需要 1 架准确但编号未知的辅助无人机，且有偏差无人机极角精确.建立辅助**无人机编号查找模型**（见公式 15~19），从而转化为问题一第 1 问处理（见公式 20~21）.通过实际偏差数据仿真计算，得到精确的极坐标（结果见表 2），验证了假设**“只需要一架飞机”**的正确性.

问题一第 3 问：要求整个编队中存在偏差的无人机仅根据方向信息调整位置，基于正弦定理几何关系和多元参数隐函数，建立了基于**三角定位法**的**二次极径/极角调整模型**（见公式 22~28），使用**迭代法**与最大误差精度判别进行求解（见图 9），得到满足精度要求的调整后编队极坐标（见表 3），并计量了迭代次数.

问题二：要求将问题一第 3 问的位置调整方法推广到其他编队队形，选择了**锥形、方阵形**两种编队形态.刻画出两种编队无偏差时的极坐标状态。引用类似的极径/极角调整—迭代修正—精度判断的循环算法，求解调整后的编队极坐标（见表 5、7）.

总体而言，利用以正弦定理为基础的几何模型，实现了基于纯方位无源定位的遂行编队的定位与调整，并进行了合理的推广.

关键词：正弦定理　三角定位法　方向信息　迭代近似

一．问题重述

在当下科技日新月异的今天，无人机的应用更加多样，如进行集群编队飞行以实现多无人机之间的任务协同工作，对无人机编队的控制显得尤为重要，且已有不少成熟方案[1][2].为了避免电磁波干扰，将每架无人机编号，由位置准确的几架无人机发送信号，需要调整方位的无人机接收并提取位置信息以实现定位.收到信号的无人机会测算出其与任何不同两架发出信号无人机的连线的夹角.

问题 1：一支由 10 架无人机组成、在同一高度飞行的圆形编队（编号 FY00~FY09），编号为 FY00 的无人机位于圆心，其余均匀分布在圆周上飞行.

（1）除圆心外，再选取已知编号的两架无人机发射信号，根据接收信号无人机测算出的夹角，建立基础数学模型进行定位；

（2）除已知位置和编号的无人机 FY00 和 FY01 外，建立数学模型，判断还需要至少几架位置准确但编号未知的无人机发射信号，才能实现无人机的定位；

（3）已知 10 架飞机的极坐标，在（1）（2）模型的基础上，将无人机逐个调整，使其最终在同一半径的圆周上.

问题 2：考虑其他编队形态下，基于纯方位无源定位的无人机位置调整方案.

二．问题分析

2.1 问题一的分析

问题一要求基于纯方位无源定位法，完成对圆形编队中的无人机的定位与调整.

第一小问，通过以中心无人机建立极坐标系，要求使用圆心无人机和两架位于圆周上编号已知的无人机，利用几何关系将接收信号的无人机的极坐标用方位信息角进行表示，构成被动接收信号模型；

第二小问，将情况更一般化，确定编号未知但位置准确的无人机个数，使得其与圆心无人机 FY00 和圆周无人机 FY01 共同借助方向信息，完成对某位置偏差无人机的精确极坐标求取，即有效定位.

第三小问，考虑整个无人机编队，设计借助于不超过三架圆周无人机的方案，对存在偏差的无人机的极轴与极角进行修正.且多次调整后，在符合精度要求的情况下完成整个无人机编队的位置调整，实现位于同一圆周、均匀分布的调整目标.

2.2 问题二的分析

问题二要求仍然基于纯方位无源定位的方法，推广考虑锥形无人机编队或其

他编队形态下，设计无人机位置调整方案.

该推广依然属于三无人机定位问题，仍考虑建立以某无人机为原点的极坐标系，设计出根据原点无人机与一个位置固定的无人机和任一辅助无人机所构成的三角结构，调整其他无人机的极角和极径的方案，使所有无人机的位置都尽可能按预期被修正.

三．模型假设

1. 无人机都处于同一飞行平面；
2. 每一次发射电磁波信号都能得到相应的方位信息；
3. 不考虑无人机在调整位置过程中对方向信息测定造成的影响；
4. 在使用几何模型对无人机进行描绘时，不考虑无人机自身形状对问题的影响；
5. 要调整位置的无人机从原位置调整到目标位置，只知道目标位置所能接收到的方位信息，即可在电磁波信号方位信息的辅助下完成调整.

四．符号说明

符号	说明	单位
α_i	待测无人机与编号为0和i的无人机连线形成的夹角	rad
α_{ij}	待测无人机与编号为i和j的无人机连线形成的夹角	rad
d	待测无人机的极径	m
U_i	位置准确且编号为i的无人机的极坐标点	
X_k	位置有偏差且编号为k的无人机的极坐标点	
β_k	编号为k的待测无人机的极角	rad
θ_i	编号为i的已知的无人机的的极角	rad

五．模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 第一小问模型的建立与求解

第一小问要求通过已知编号且位置准确的三架无人机，包括位于圆心的无人机U_0与编号为i和j的无人机U_i和U_j发射信号，被动接收信号的待测无人机X_k提取出方向信息，进而确定其精确的极坐标进行定位.

步骤一，根据待测无人机X_k所测角度大小，对其与无人机U_0、U_i、U_j位置关系的一般情况进行分类讨论；

步骤二，分别在不同情况下使用正弦定理将X_k与U_0、U_i、U_j不同的位置关系

进行几何表示，列出方程；

步骤三，描绘算法实现过程，并以一组数据进行模拟求解.

（1） 讨论待测无人机X_k与无人机U_0、U_i、U_j位置关系

待测无人机X_k收到无人机U_0，U_i，U_j发射的电磁波信号，检测到的方向信息夹角共有三个，即X_k与U_0，U_i连线所成夹角为α_i，X_k与U_0、U_j连线所成夹角α_j，X_k与U_i、U_j连线所成夹角α_{ij}. 根据待测无人机X_k与无人机U_0、U_i、U_j形成夹角的大小，位置关系有以下三种情况：

①情况一：X_k与U_0、U_j连线所成的夹角 α_j 为三个夹角中最大的角；

②情况二：X_k与U_i、U_j连线所成的夹角 α_{ij} 为三个夹角中最大的角，其中X_k的位置还有两种情形；

③情况三：X_k与U_0、U_i连线所成的夹角 α_i 为三个夹角中最大的角.

在四种不同的位置关系下，依赖于几何关系的待测无人机X_k极坐标表示方法不尽相同. 借助位置准确且编号已知的无人机U_0、U_i、U_j的极坐标情况，选定不同三角形使用正弦定理，即可完成对X_k的几何表示.

（2） 不同位置关系下无人机之间的几何表示

建立极坐标系，无人机U_0位于极点$(0,0)$，无人机 U_i、U_j均位于圆形编队的圆周上. U_i坐标为(R_0,θ_i)，U_j坐标为(R_0,θ_j)，由于选定的无人机U_i、U_j位置准确且编号已知，可以确定两个无人机对应的极角：

$$\theta_i=\frac{2(i-1)\pi}{9},\ \theta_j=\frac{2(j-1)\pi}{9}\ (i,j\geq 1) \tag{1}$$

其中，θ_i表示无人机U_i的极角，θ_j表示无人机U_j的极角，i、j指代在无人机编队中的编号.

①设待测无人机X_k坐标为(d,β_k).在情况一的位置关系下，各个无人机的几何关系如图 1 所示.

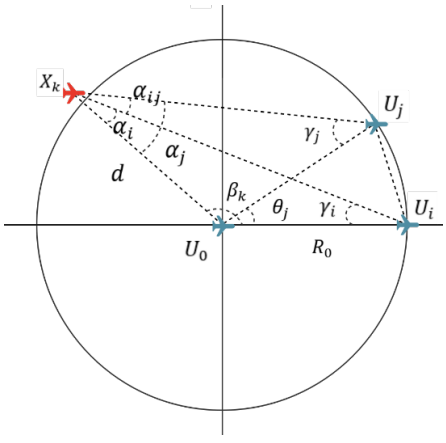

图 1：情况一无人机位置关系图

此时X_k测得的三个角中α_j最大.为了表示出β_k与d，选择在△$X_kU_0U_i$与△$X_kU_0U_2$两个三角形中，以d为公共边使用正弦定理，列出关于β_k与d的方程.

在△$X_kU_0U_i$中：

$$\frac{R_0}{\sin\alpha_i}=\frac{d}{\sin(\pi-\alpha_i-\beta_k)} \tag{2}$$

其中α_i表示待测无人机X_k分别和无人机U_0、U_i的两条连线形成的夹角.由于U_0、U_i位置准确，所以U_i的极径即为所设圆周的半径R_0.d为被测飞机X_k的极径，其对应的角为：U_i与X_k、U_0连线所成的夹角γ_i，可表示为：

$$\gamma_i = \pi-\alpha_i-\beta_k \tag{3}$$

其中，θ_i是无人机U_i的极角.

在△$X_kU_0U_j$中：

$$\frac{R_0}{\sin\alpha_j}=\frac{d}{\sin\left(\pi-\alpha_j-(\beta_k-\theta_j)\right)} \tag{4}$$

与△$X_kU_0U_i$同理，α_j表示待测无人机X_k分别和无人机U_0、U_j的两条连线形成的夹角，U_j的极径也为所设圆周的半径R_0.d为待测飞机X_k的极径，其对应的角为：U_j与X_k、U_0连线所成的夹角γ_j，可表示为：

$$\gamma_j = \pi-\alpha_j-(\beta_k-\theta_j) \tag{5}$$

其中，θ_j是无人机U_j的极角.

②类似地，情况二下第一种情形的位置关系如图 2 所示.

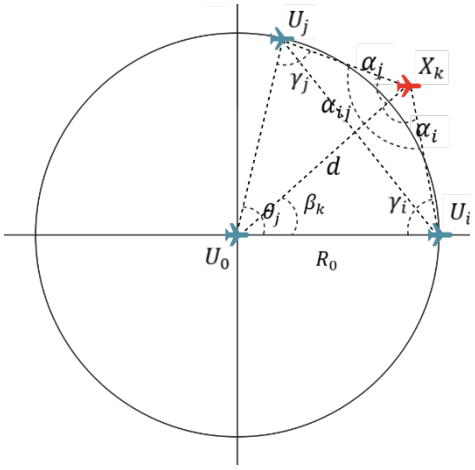

图 2：情况二情形一无人机位置关系图

此时X_k测得的三个角中α_{ij}最大，仍然以d为公共边，列出关于β_k与d的方程.

在△$X_kU_0U_i$中：

$$\frac{R_0}{sin\alpha_i} = \frac{d}{\sin(\pi - \alpha_i - \beta_k)} \tag{6}$$

与情况一同理，R_0为U_i的极径，其对应的角为α_i.待测飞机X_k的极径为d，此时对应的角为U_i分别与X_k、U_0连线所成的夹角γ_i，可表示为：

$$\gamma_i = \pi - \alpha_i - \beta_k \tag{7}$$

在△$X_k U_0 U_j$中：

$$\frac{R_0}{\sin\alpha_j} = \frac{d}{\sin\left(\pi - \alpha_j - (\theta_j - \beta_k)\right)} \tag{8}$$

U_j的极径等于R_0，其对应的角为α_i.待测飞机X_k的极径为d，此时对应的角为U_j分别与X_k、U_0连线所成的夹角γ_j，可表示为：

$$\gamma_j = \pi - \alpha_j - (\theta_j - \beta_k) \tag{9}$$

情况二下第二种情形的位置关系如图 3 所示.

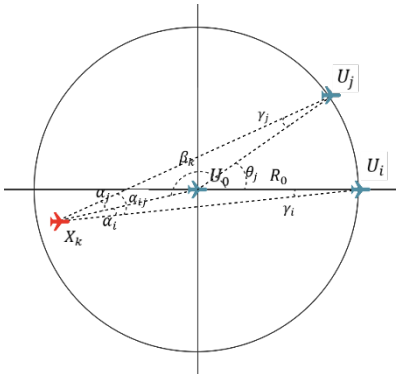

图 3：情况二情形二无人机位置关系图

与情形一相似，仅有公式（8）不同，应如公式（10）：

$$\frac{R_0}{\sin\alpha_i} = \frac{d}{\sin\left(\pi - \alpha_j - (\beta_k - \theta_j)\right)} \tag{10}$$

③类似地，情况三的位置关系如图 4 所示.

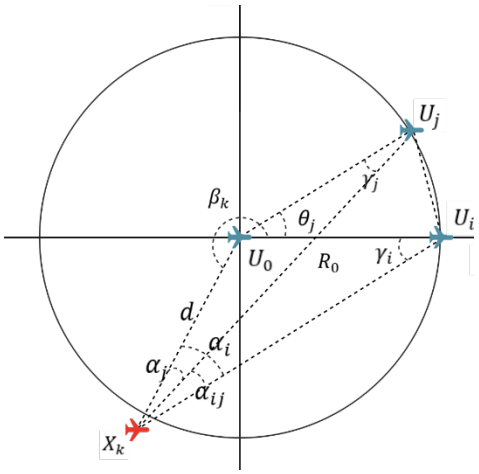

图 4：情况三无人机位置关系图

此时X_k测得的三个角中α_i最大，以d为公共边并使用正弦定理，列出关于β_k与d的方程.

在$\triangle X_k U_0 U_i$中：

$$\frac{R_0}{\sin\alpha_i}=\frac{d}{\sin(\pi-\alpha_i-(2\pi-\beta_k))} \tag{11}$$

U_i的极径为R_0，其对应的极角为α_i得到方程左端.待测飞机X_k的极径为d，在此三角形对应的角为γ_i，可表示为：

$$\gamma_i=\pi-\alpha_i-(2\pi-\beta_k) \tag{12}$$

在$\triangle X_k U_0 U_j$中：

$$\frac{R_0}{\sin\alpha_j}=\frac{d}{\sin\left(2\pi-\alpha_j-(\beta_k-\theta_j)\right)} \tag{13}$$

U_j的极径R_0，其对应的角为α_i得到方程左端.待测飞机X_k的极径为d，此时对应的角为U_j与X_k、U_0连线所成的夹角γ_j，可表示为：

$$\gamma_j=2\pi-\alpha_j-(\beta_k-\theta_j) \tag{14}$$

以上三种情况中，γ_i、γ_j都依据于多边形的内角和的性质，依据总内角和减去其他角来间接表示.将两个三角形的正弦定理表示式联立，即可表示出关于β_k与d的解析解，以此确定了被测无人机X_k的精确定位.

（3） 模型求解与仿真

实际选取数据做仿真模拟，步骤如下：

① 给出编号 1-9 下所有的无人机应该分布的位置的极坐标；

② 根据读取出的α_i，α_j，α_{ij}的大小关系判别不同情况下应使用的模型并进

行求解；

③ 使用三角定位法. 即根据几何关系，列举出两组正弦定理关系方程. 两个方程即为关于β和d的二元方程组；

④ 通过将两个方程作比，消去未知量d. 根据所选定的情况确立β的取值范围，并且求解其具体值；

⑤ 将β代入到原来的方程中求出d.

具体流程如图 5.

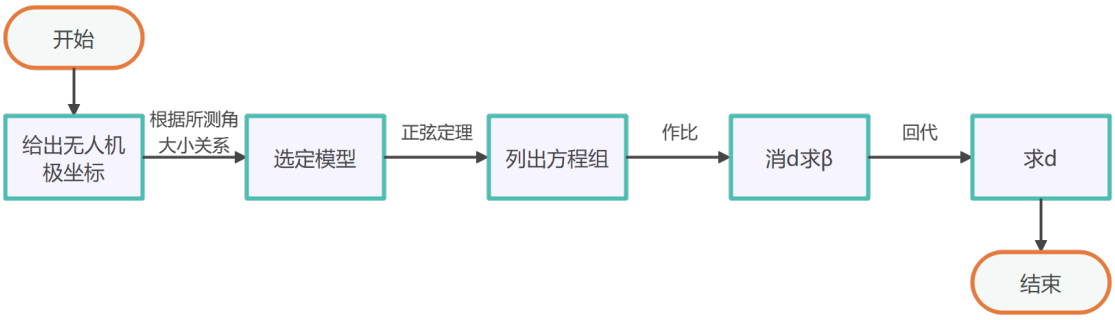

图 5：利用三架编号已知且精确的无人机对偏差无人机定位算法流程图

选择位置精确的编号为 FY00、FY01、FY02 无人机作为发射信号无人机，随机选取编队中的一架有误差的无人机，经检验编号为 FY04 且存在位置误差.接受信号无人机在收到方位信号后，读出代表方向信息的三个角度：$\alpha_1 = \frac{\pi}{6}, \alpha_2 = \frac{5\pi}{18}, \alpha_{12} = \frac{\pi}{9}$.

利用 python 数学软件进行求解(见程序 1)，计算结果如表 1 所示.

表 1：利用三架位置准确且编号已知无人机定位结果

符号	计算结果	单位
d	100.12114708	m
β_4	0.66644398π	rad

由表 1 可知，基于测定的方向信息，计算得到的极坐标几乎是贴合圆形编队的曲线，这证明了被动定位模型是准确的.

5.1.2 第二问模型的建立与求解

第二小问，由于部分位置准确的无人机编号未知，根据待测无人机接收到的信号，除已知编号且位置准确的无人机U_0、U_1外，假设至少需要 1 架未知编号但位置准确的辅助无人机U_i发射信号就能实现定位，通过模拟结果验证假设的正确性.

步骤一，待测无人机X_k为研究对象，先确定其大致位置，后近似确定辅助无人机编号；

步骤二，将求解出的辅助无人机编号i确定后，重新定位待测无人机X_k.

（1）辅助无人机编号确定

已知U_1，U_i均位置准确，因此其极坐标可表示为$U_1(R_0,0)$，$U_i(R_0,\theta_i)$由于该待测无人机X_k是被研究对象，所以其在极坐标系中的极径d和极角β_k大致可以确定为：

$$\begin{cases} d \approx R_0 \\ \beta_k \approx \dfrac{2\pi}{9}(k-1) \end{cases} \tag{15}$$

在本小问中仅考虑第一小问中情况一的位置关系，如图 6 所示.其余情况原理相同，可类比 5.1.1，在此不做展开说明.

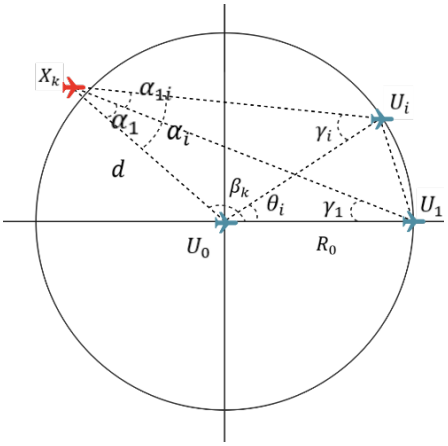

图 6：编号位置无人机辅助定位偏差无人机几何关系示意图

假定β_k精确，由U_0，U_1，U_i向待测无人机X_k发射信号，测得角度可以通过在两个三角形中，使用正弦定理将不同要素进行几何关系的表示.

在$\triangle X_kU_0U_1$中：

$$\frac{R_0}{\sin\alpha_1} = \frac{d}{\sin(\pi-\alpha_1-\beta_k)} \tag{16}$$

其中，α_1为X_k测得其与U_0，U_1连线的夹角，也是U_1极径R_0在此三角形中所对应的角.X_k极径d所对应的角为U_1与U_0，X_k连线的夹角γ_1，根据三角形内角和得出$\gamma_1 = \pi-\alpha_1-\beta_k$.

在$\triangle X_kU_0U_i$中：

$$\frac{R_0}{\sin\alpha_i} = \frac{d}{\sin(\pi-\alpha_i-(\beta_k-\theta_i))} \tag{17}$$

其中，α_i为X_k测得其与U_0，U_i连线的夹角，也是U_i极径R_0在此三角形中对应的角.X_k极径d所对应的角为U_i与U_0，X_k连线的夹角γ_i，根据三角形内角和得出$\gamma_i = \pi-\alpha_i-(\beta_k-\theta_i)$.

方程（16）（17）作比消去d和R_0可以得到关于θ_i的公式（18）：

$$\frac{\sin\alpha_i}{\sin\alpha_1} = \frac{\sin(\pi-\alpha_i-(\beta_k-\theta_i))}{\sin(\pi-\alpha_1-\beta_k)} \tag{18}$$

求解出近似的θ_i，再用θ_i向上下寻找最近的$\frac{2\pi}{9}$的整数倍记为n，记辅助无人机编号为i，通过公式（19）：

$$i = n+1 \tag{19}$$

求出编号i再将寻找到的整数倍的角记为θ_i'.

（2）求解待测无人机X_k极坐标

将θ_i'回代入方程（18），得到方程（20），此时是关于β_k的方程：

$$\frac{\sin\alpha_i}{\sin\alpha_1} = \frac{\sin(\pi-\alpha_i-(\beta_k-\theta_i'))}{\sin(\pi-\alpha_1-\beta_k)} \tag{20}$$

解出β_k'，此时为X_k准确的极角，将β_k'回代入方程（15），得到方程（21）：

$$\frac{R_0}{\sin\alpha_1} = \frac{d}{\sin(\pi-\alpha_1-\beta_k')} \tag{21}$$

求出X_k准确的极径d.

（4） 算法实现与仿真

对算法进行仿真模拟，步骤如下：

① 给出编号 1-9 下所有的无人机应该分布的位置的极坐标；

② 首先假设β是精确的，在$\triangle X_kU_0U_1$中根据正弦定理求解出d. 再在$\triangle X_kU_0U_2$中求解出γ_2来. 由此可知θ_2的近似值再根据θ_2是$\frac{2\pi}{9}$的整数倍求出其精确值；

③ 根据θ_2的大小确定其无人机的编号；

④ 根据α_1、α_2、α_{12}的大小关系判别该情况下应该使用哪个模型进行求解；

⑤ 使用三角定位法. 即根据几何关系，列举出两组正弦定理关系等式. 两个等式即为关于β和d的二元方程组；

⑥ 通过将两个等式做比，消去未知量d. 根据所选定的情况确立β的取值范围，并且求解其具体值；

⑦ 将β带入到原来的方程中求出d.

流程如图 7 所示.

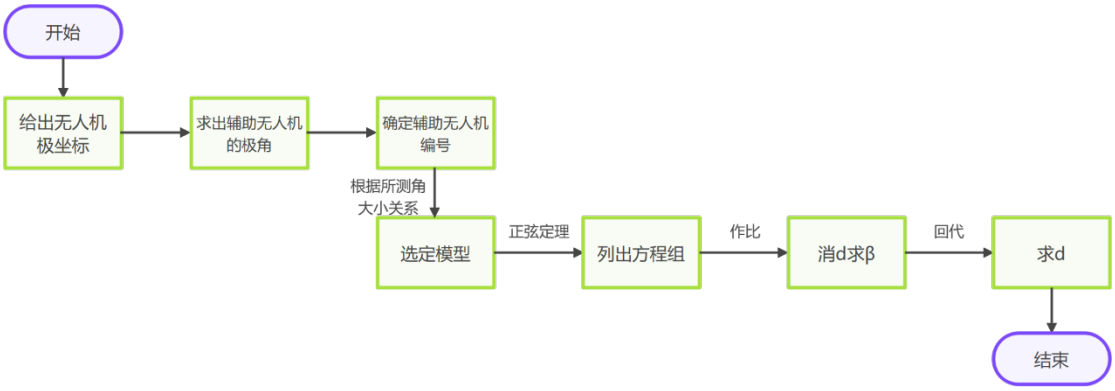

图 7：未知编号无人机辅助下有偏差无人机坐标定位算法流程图

选定 FY05 作为测试对象. 已知 FY00 、FY01 向其发射信号. 另外选择 FY02 作为未知编号的发射信号无人机. FY05 在接收到三架无人机发射出来的方位信息后，得到 3 个方向角度：$\alpha_1 = 0.0565095\pi, \alpha_2 = 0.3041387\pi, \alpha_{12} = 0.24762599\pi$.

通过 python 求解出 FY05 的极坐标结果（见程序 2），如表 2 所示.

表 2：利用编号未知无人机对存在位置偏差的 FY05 进行模拟定位的结果

符号	计算结果	单位
d	100.16271172	m
β_5	0.8868845π	rad

如表所示，根据给出的方向信息进行模拟测试，得到了位置偏差无人机的精确坐标.验证了只需要 1 架编号未知但准确的无人机即可完成定位.

5.1.3 第三问的模型建立与求解

第三小问要求我们在已知道无人机编队存在位置误差的情况下，设计方案对所有无人机的位置进行调整，使得它们最终均匀分布在圆周上.

步骤一，确定调整要达到的预期目标，分析初始情况下的位置数据的特点与隐含信息.

步骤二，使用“三角定位法”，构造出方向信息与待调整无人机的极角和极径等参数的关系.

步骤三，根据参量确定出向准确极角、极径的调整方法.

步骤四，在整个编队中重复使用调整方法，通过多次重复，使最终调整完成后的编队内无人机的极坐标在精度范围内达到目标.

（1） 数据预处理与分析

首先对题目中所给的“表 1 无人机初始位置”一表内的数据进行分析处理，表格内有两列数据，第一列给定了编队中所有无人机编号的缩写，第二列给定了初始实际情况下的所有无人机的实际位置.

若某编号为$i(0<i<10, i\in N)$的无人机极径是准确的，则其极径R_i、极角θ_i满足关系：

$$\begin{cases} R_i = 100 \\ \theta_i = 40°(i-1) \end{cases} \tag{22}$$

公式(20)也是要求的无人机编队调整目标，对比表格不难发现，除编号为FY00 与 FY01 的两架无人机外，绝大多数无人机与准确位置存在误差，因此可以类比前两问中的处理方法，将两架无偏且编号准确的无人机作为用于组成定位的无人机组，再找寻另一架辅助无人机即可.

为了方便计算，将所有无人机的极角由角度制转化为弧度制使用.

（2）构建极坐标与方向信息的关系

为了构建坐标与采用“三角定位法”，这里指的是选定极点无人机、FY01 无人机（认为位置准确）、一架编号随机且可以存在偏差的辅助无人机，用于向“锥形编队“中待调整的无人机U_k发射电磁信号，使得U_k能够接收到方位信息并对自身位置做出调整.

若随机选定了无人机U_i作为辅助无人机.对于待调整无人机U_k而言，由于保持无线电静默，所以无法知道辅助无人机U_i的精确位置，只能假定U_i按照飞行编队的要有准确到达了相应的极坐标处，此时理论情况下的辅助无人机极坐标为：

$$U_i^0 = (R_i^0, \theta_i^0) \tag{23}$$

其中，R_i^0、θ_i^0分别表示在编队准确位置中的辅助无人机的极径与极角.但是实际情况下，极径与极角都是存在偏差的，真实情况的极坐标为：

$$U_i = (R_i, \theta_i) \tag{24}$$

其中，R_i、θ_i分别表示真实初始情况下辅助无人机的极径与极角.

对于待调整无人机U_k而言，由于位置的调整只能依据于传来的电磁信号中提取到的方位信息.所以在发射信号飞机都是固定的情况下，每接受到一组方位信息，有且仅有一个极坐标位置与之对应.

首先，为了调整其余无人机位置，除U_0，U_1这两架位置准确的无人机外，需要再随机选择一架编号为i辅助无人机记为U_i，需要调整的无人机记为 U_k ，它们的位置分布如图 8 所示，仅是给出一种待调整无人机与定位飞机位置关系情况的展示，求解时则考虑一般情况.

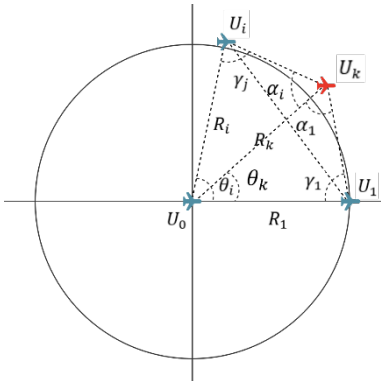

图 8：“三角定位法”几何关系示意图

如图，在△ $U_iU_0U_k$中：

$$\frac{R_i}{\sin\alpha_i} = \frac{R_k}{\sin(\pi-\alpha_i-(\theta_i-\theta_k))} \tag{25}$$

R_i，R_k分别为辅助无人机U_i和待调整无人机U_k的极径，θ_i^0，θ_k分别为U_i^0，U_k的极角，α_i为U_k测出的与U_i^0，U_0连线的夹角.

在△ $U_0U_1U_k$中：

$$\frac{R_1}{\sin\alpha_1} = \frac{R_k}{\sin(\pi-\alpha_1-\theta_k)} \tag{26}$$

R_1为确定的无人机U_1的极径，α_1为U_k测出的与U_1，U_0连线的夹角.

将方程（23）与方程（24）作比得到综合方程（25）：

$$\frac{R_i^0\sin\alpha_1}{R_1\sin\alpha_i} = \frac{\sin(\pi-\theta_k-\alpha_1)}{\sin\left(\pi-\left(\theta_i^0-\theta_k\right)-\alpha_i\right)} \tag{27}$$

设六元隐函数F：

$$F(\theta_k,\theta_i^0,\alpha_1,\alpha_i,R_i^0,R_1) = \frac{\sin(\pi-\theta_k-\alpha_1)}{\sin\left(\pi-\left(\theta_i^0-\theta_k\right)-\alpha_i\right)} - \frac{R_i^0\sin\alpha_1}{R_1\sin\alpha_i} \tag{28}$$

因为“三角定位法”中使用的三架无人机都是固定不变的，所以隐函数F中的θ_i^0、R_i、R_1都是固定取值的参数.所以隐函数只取决于θ_k、α_1、α_i三个参量，令$F=0$，得到的就是三个参量的关系式.

（3）调整极角

在调整极角时，固定极径不动，此时极径是一个常数.

步骤 1：首先确定要调整到的目标极角值θ_{target}，令$\theta_k=\theta_{target}$，代入公式

（26） 计算出此时的α_1值.

步骤 2：将得到的α_1值、θ_{target}代入到隐函数公式（28）中，根据三个参数的关系式得到α_i的值.

此时得到了调整极角所需要的两个方向信息α_1、α_i的值，即待调整无人机U_k不断调整自身位置过程中，当检测到方向信息角取值为以上结果时，即说明调整到了目标极角状态.

（4）调整极径

在调整极径时，固定极角不动，此时极角是一个常数.

首先要确定调整到的目标极距值R_{target}，令$R_k = R_{target}$，代入公式（26），计算出此时的α_1，再代入公式（25），计算出α_i的取值.

此时得到的方向信息α_1、α_i即可指导无人机U_k完成对极径的调整，调整过程与调整极角相似.

（5）迭代修正

利用以上的极距与极角的调整方法，对无人机编队中的位置偏差进行修正，主要分为以下几个步骤：

① 读取编队中所有飞机的极坐标数据

② 使用“三角定位”方法，每架无人机的修正进行两次，使用两架辅助无人机U_i、$U_t(i \neq t)$,分别得到修正后的极坐标数据组U_{k1}, U_{k2}，并取均值得到修正后最终极坐标数据：

$$U_k = \frac{U_{k1} + U_{k2}}{2} \tag{29}$$

③ 除了固定不变的极点无人机与设定为精确的 FY01 无人机，依次对编队中进行其他无人机进行极坐标修正，得到编队中所有无人机修正之后的极坐标，一次迭代完成.

④ 计算一次迭代之后的极坐标误差，如果最大误差满足精度要求，就认为达到了调整要求，调整结束；否则认为调整不达标，将数据返回到步骤②，再次迭代.迭代次数越少，说明向外发射信号次数越少，无线电静默效果越好.

迭代计算的流程图如图 9 所示.

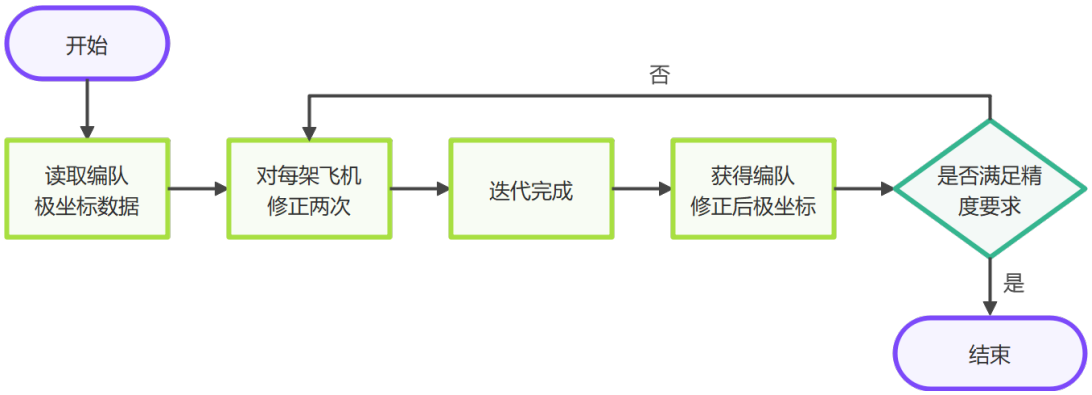

图 9：三角定位法调整无人机编队位置算法流程图

通过以上流程，即可实现对于初始有误差的整组无人机位置的调整.

（6）算法实现与模型求解

利用 python 程序(见程序 3)，对题目中“表 1 无人机初始位置”里的偏差数据进行修正，借助以上“三角定位法”和“极角/极距调整”模型与“迭代修正”算法.并设计误差精度要求为 1%，同时计量迭代次数，得到的调整结果如表 3 所示.

表 3：迭代调整后无人机极坐标

编号	极径（m）	极角（rad）
0	0	0
1	100.0	0
2	99.89933997218799	0.22311487090567π
3	99.8533010452353	0.44499885267783π
4	99.92117176014001	0.66681243780564π
5	99.91493784697604	0.88893663258870π
6	100.5136739644955	1.11139946524393π
7	99.89828094452713	1.33314568368918π
8	100.30430517160761	1.55671093845621π
9	100.04235957456474	1.77815184248192π

由表可知，修正后的无人机极径和极距已被合理控制在目标值的左右 1%误差内，在 3-8 次内完成迭代，尽可能地保持了电磁沉默.说明实现算法达到了位置调整的预期目标，方案的功能设计是合理的.

5.2 问题二模型的建立与求解

将圆形编队的无人机位置调整方法向其他情况进行推广，以探究在不同编队形态下无方位纯源定位的方法是否仍然适用，这里选择了“锥形”和“方阵形”无人机编队.

5.2.1“锥形形态”与“方阵形状态”下的无人机位置调整方案

探讨锥形/方阵形形态下的无人机编队的位置调整方案，主要包括：无偏差位置下各个无人机位置的描绘、调整方案的建立、数据仿真三部分.由于在以下两

种形态下除极坐标系与无人机的编队的极坐标不同，其他的定位原理与调整方法均一致，所以两种编队形态的调整方案将一同陈述.

（1） “锥形形态”无偏位置描述

存在一种由 15 架无人机构成的锥形编队，15 架无人机总体构成了一个边长为 200m 大三角形，该大三角形实际由多个三元小三角形构成，且每个三元三角形的顶点都代表一架无人机，编队形态如图 10 所示.

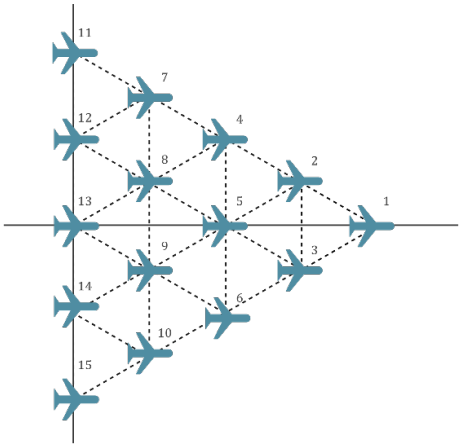

图 10：锥形编队位置无偏示意图

以编号为 FY13 的无人机为极点，有向线段<FY13,FY01>为极轴方向，建立极坐标系.在该坐标系下，各无人机位置准确时，任意一架无人机到最相邻的无人机的距离为 50 m，即所有的三角形都是等边的.且所有无人机都位于同一个高度平面内.

（2） “方阵形态”无偏位置描述

方阵形态下的无人机编队由 9 架无人机组成，其中一架位于方阵的几何中心，其他 8 架均匀分布在方阵的四个顶点与四条边的中点上。设定任意两架相邻无人机间距为 50m，在所有无人机的位置准确无偏的条件下，编队形态如图 11 所示.

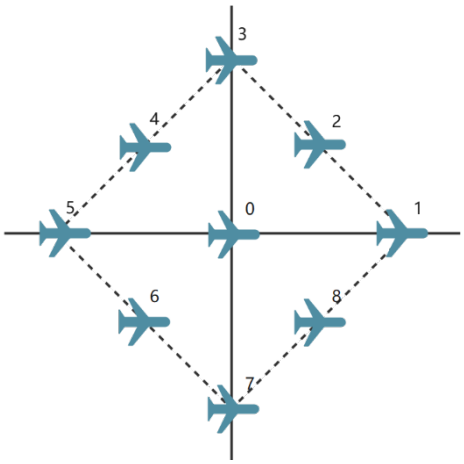

图 11：方阵编队的无偏示意图

以几何中心为极点，有向线段$<0,1>$为极轴，建立极坐标系.

在该坐标系下，无人机坐标无偏时，无人机对应的极角都为$\frac{\pi}{4}$的自然数倍，极径是方阵边长的$\frac{1}{2}$倍或$\frac{1}{\sqrt{2}}$倍.

（3） 调整方案设计

与圆形编队的调整方案相类似，在建立方阵形与锥形编队形态的调整方案时，依然使用“三角定位法”.

在锥形编队中，选定极点无人机U_{13}与顶点无人机U_1作为编号已知且无误差的发射信号无人机，再从锥形编队中随机抽取一架可有偏差的无人机U_i作为辅助无人机，对待调整无人机U_k进行调整，如图 12 所示.

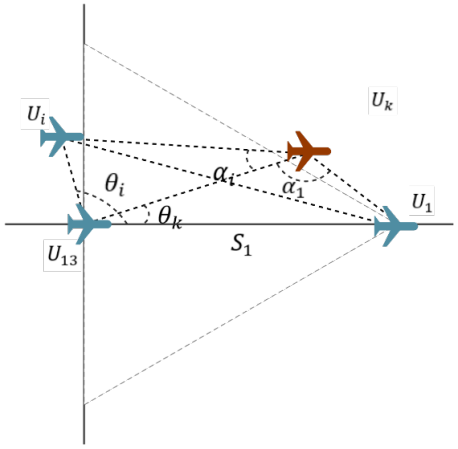

图 12：“锥形”编队下几何位置调整示意图

采用与 5.1.3 中相同的几何关系构建法、极角与极径调整方法、经过数次迭代之后，在精度控制的范围内，使得锥形编队的所有无人机都近似满足其无偏位置下的极坐标位置状态.

在方阵形编队中，选定极点无人机U_0,位于极轴上的顶点无人机U_1作为“三角定位法”中的两架用于发射信号的无人机.此时认为U_1的位置是无偏的，并认为其极径等于方阵边长的$\frac{1}{\sqrt{2}}$倍.再从方阵编队中随机选一架辅助无人机U_i，共同构成“三角定位”中的三架无人机，向待调整无人机发射电磁波信号以调整位置，四

者几何关系如图 13 所示.

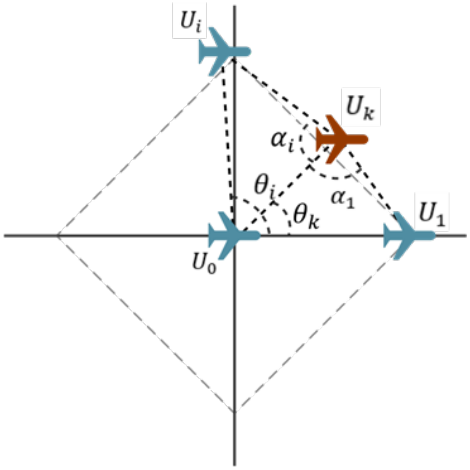

图 13：“方阵形”编队下几何位置调整示意图

具体的编队位置调整方案依然是构建几何关系-调整极角与极径-迭代计算-满足收敛.最终在一定的精度下，得到的方阵编队的无人机极坐标状态接近满足于设定的无偏位置.

（4） 数据仿真

为了测试使用在原圆形编队的位置调整方案是否可以被进一步推广，以及其推广计算后时候是否依旧保证稳定性，下面分别对两种编队形态下各选取一组有偏差的无人机初始位置（极坐标），分别使用 5.1.3 中的位置调整模型与算法进行仿真求解.

方阵形编队：

设定精度为 0.8%，首先设定一组与标准无偏位置相比，存在一定误差的数据，如图表 4 所示，其中编号 0、1 无人机认定是无误差的，在“三角定位法”中一直作为信号发射无人机.

表 4：方阵形编队原始有偏差位置数据

编号	极径（m）	极角（rad）
0	0	0
1	70.7	0
2	48	0.27222222π
3	67.3	0.49666667π
4	54	0.74222222π
5	76.4	0.98333333π
6	47	1.26277778π
7	73.3	1.49111111π
8	44	1.77777778π

使用位置调整方案，经过仿真计算（**见程序 4**），位置调整后的编队极坐标如表 5 所示.

表 5：调整后方阵形编队无人机极坐标

编号	极径 (m)	极角 (rad)
0	0	0
1	70.700000	0
2	49.733032	0.252966π
3	70.246155	0.499555π
4	50.533936	0.748962π
5	71.460858	0.997775π
6	49.599548	1.251706π
7	71.047058	1.498813π
8	49.199097	1.753708π

如表所示，得到的调整后结果基本符合设定精度的要求。一次仿真迭代 7 次最终收敛，结果与设定的无偏位置状态几乎相近，说明圆形编队的位置调整方案可以推广到方阵形编队中.

锥形编队：

同样设定精度为 0.8%，设置一组存在偏差的锥形编队初始位置，如表 6 所示.

表 6：锥形编队原始有偏差位置数据

编号	极径 (m)	极角 (rad)
1	173.2	0
2	134	0.06666667π
3	128	-0.05611111π
4	94	0.15555556π
5	93	0.01111111π
6	96	1.82777778π
7	85	0.32222222π
8	48	0.15555556π
9	53	1.82777778π
10	89	1.67222222π
11	102	0.47777778π
12	55	0.51111111π
13	3	0.01111111π
14	46	1.51111111π
15	92	1.48888889π

再次使用圆形编队中的位置调整方案，仿真计算后（见程序 5），得到的调整后编队极坐标如表 7 所示，其中编号 13、1 无人机认定是无误差的，在“三角定

位法”中一直作为信号发射无人机.

表 7：调整后锥形编队无人机极坐标

编号	极径 (m)	极角 (rad)
1	173.2	0
2	132.459	0.061136π
3	131.858325	-0.060079π
4	99.399323	0.165554π
5	87.240723	0.001112π
6	99.599548	1.832777π
7	86.439819	0.332221π
8	49.799774	0.165554π
9	50.30033989	1.832777π
10	86.840271	1.667223π
11	100.200226	0.497775π
12	50.500565	0.501112π
13	0.300339	0.001112π
14	49.599548	1.501112π
15	99.199097	1.498888π

如表所示，调整结果基本达到了所设定的精度范围。一次仿真迭代 8 次最终收敛，结果接近预先设定的无偏位置状态，说明圆形编队的位置调整方案可以推广到锥形编队中.

六. 灵敏度分析

为了测试算法的稳定性，现另外取了一组略有偏差的无人机极坐标数据（极角已转化为含π的形式）.其具体数值如表 8 所示.

表 8：测试数据初始极坐标数据

编号	极径 (m)	极角 (rad)
0	0	0
1	100	0
2	89	0.20π
3	105	0.45π
4	110	0.67π
5	96	0.87π
6	94	1.12π
7	108	1.31π
8	96	1.57π
9	99	1.77π

使用同样的算法，设置为精度 0.5%，按照目标要求计算迭代收敛后无人机编队中各个点的极坐标情况（极角已转化为含π的形式），如表 9：

表 9：测试数据调整后极坐标数据

编号	极径 (m)	极角 (rad)
0	0	0
1	100	0
2	99.7748	0.224197π
3	100.387	0.442984π
4	100.214	0.666275π
5	100.343	0.888697π
6	100.46	1.111369π
7	100.19	1.333683π
8	99.9816	1.555484π
9	100.294	1.780333π

由表 9 可知，更换一组误差数据和精度要求后，原算法依旧能够得到符合要求的结果，说明模型能够在不同的参数与数据下正常实现功能，且能按照不同精度的要求输出相应的结果，具有一定的灵敏性.

七．模型的评价与改进

7.1 模型的优点与缺点

优点：

（1） 模型使用三角函数对目标无人机的极坐标进行表示与求取，得到的是解析解，精确度高.

（2） 在无编号定位过程中，为了查找该发射信号无人机的编号，假设极角精确而非极径精确，误差相对较小.

（3） 在调整无人机位置过程中，对于一架无人机进行两次“三角定位”法调整极坐标并取均值作为调整后结果，提高了模型的鲁棒性与稳定性，并适当地提高了迭代收敛速度.同时使用隐函数的根表示出各个参量之间的关系，更具有一般性和普适性.

缺点：

（1） 第三问中的迭代收敛次数为计算得到的数值解，缺少严格数学证明.

（2） 模型的建立基于一定的理想条件，如：待位置调整的无人机只要知道正确位置下对应的方向信息，就能够不断调整位置直到被动接收到的方向信息如正确位置所示.而实际上接收到方向信息也并非完全准确，与模型存在一定的区别.

7.2 模型的改进与推广

位置调整的迭代过程中，可以在“三角定位”之前对辅助无人机的位置进行适当的判断，避免使用一些位置发生严重偏差的无人机作为辅助无人机.以此防止出现待调整无人机的调整偏差，以提高准确度和迭代速度.

参考文献

[1] 王欢,刘树光,张博洋.基于分层控制结构的有人/无人机编队队形控制.电光与控制[J].2022-09-18.

[2] 郭鹏军,张睿,高关根,许斌.基于相对速度和位置辅助的无人机编队协同导航.上海交通大学学报[J]2022-09-18.

附录

附录一、支撑材料文件清单

附件编号	文件名称	附件说明
程序 1	1.1.py	问题 1.1 被动接受信号模型仿真程序
程序 2	1.2.py	问题 1.2 编号未知但准确辅助无人机定位仿真计算程序
程序 3	1.3.py	问题 1.3 位置调整方案迭代计算程序
程序 4	2.方形阵.py	问题 2：方阵形编队位置调整方案迭代计算程序
程序 5	2.锥形阵.py	问题 2：锥形编队位置调整方案迭代计算程序

附录二、

程序 1:问题 1.1 被动接受信号模型仿真程序

程序说明：基于 python 3.9.7 开发环境编译 编译解释器：Spyder

```
import pandas as pd
import numpy as np
import math

#定义圆周率
pi=3.1415926

#构建极坐标数据
r=np.array([100]*9)
r=np.append(0,r)
theta=np.linspace(0,360-40,9)
theta=np.append(0,theta)

coordinate=pd.DataFrame({
    "r":r,
    "theta":theta
    })


#计算零点
```

```
def findzero(function_points):   #function_points 一定要是 np.DataFrame 类型的
    zeros=np.array([])
    for i in range(len(function_points)-1):
        product=function_points.iloc[i,1]*function_points.iloc[i+1,1]
        if product<=0:
            zeros=np.append(zeros,function_points.iloc[i,0])
        else:
            continue
    return zeros


#分类讨论
#case1 情况下 alpha1 最大
def case1(alpha1,alpha2,alpha12):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[2,1]*pi/180

    if (varphi<=pi):
        minum=varphi+pi
        maxum=2*pi
    else:
        minum=0
        maxum=varphi-pi
    #解方程
    beta=np.linspace(minum,maxum,500)
    fun=np.sin(-beta+varphi+alpha2)*np.sin(alpha1)-np.sin(alpha1-beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1-sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta

```

```
#case2 情况下 alpha2 最大
def case2(alpha1,alpha2,alpha12):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[2,1]*pi/180
    if (varphi<=pi):
        minum=varphi
        maxum=pi
    else:
        minum=pi
        maxum=varphi+pi
    #解方程
    beta=np.linspace(minum,maxum,500)
    fun=np.sin(alpha2+beta-varphi)*np.sin(alpha1)-np.sin(alpha1+beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1+sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta

#case3 情况下 alpha12 最大
def case3(alpha1,alpha2,alpha12):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[2,1]*pi/180
    if (varphi<=pi):
        minum1=0
        maxum1=varphi
        minum2=pi
        maxum2=varphi+pi
    else:
        minum1=varphi-pi
        maxum1=pi
        minum2=varphi
        maxum2=2*pi

    #解方程
```

```
    beta=np.append(np.linspace(minum1,maxum1,500),np.linspace(minum2,maxum2,500))
    #beta=np.linspace(pi,varphi+pi,500)
    fun=np.sin(-beta+varphi+alpha2)*np.sin(alpha1)-np.sin(alpha1+beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1-sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta


#对数据进行分类:
def solution(alpha1,alpha2,alpha12):
    if (alpha1>alpha2)&(alpha1>alpha12):
        d,sol_beta=case1(alpha1, alpha2, alpha12)
    elif (alpha2>alpha1)&(alpha2>alpha12):
        d,sol_beta=case2(alpha1, alpha2, alpha12)
    else:
        d,sol_beta=case3(alpha1, alpha2, alpha12)
    return d,sol_beta*pi/180


if __name__ == "__main__" :
    #测试代码正确性
    #三个孔分别填：α1，α2，α12
    d,beta=solution(30,50,20)
    print('无人机的极径为:')
    print(d)
    print('无人机的方位角为：')
    print(beta)
```

附录三、

程序 2:问题 1.2 编号未知但准确辅助无人机定位仿真计算程序

程序说明：基于 python 3.9.7 开发环境编译 编译解释器：Spyder

```
#加载库
import pandas as pd
```

```
import numpy as np
import math

#定义圆周率
pi=3.1415926

#构建极坐标数据
r=np.array([100]*9)
r=np.append(0,r)
theta=np.linspace(0,360-40,9)
theta=np.append(0,theta)

coordinate=pd.DataFrame({
    "r":r,
    "theta":theta
    })

#计算零点
def findzero(function_points):   #function_points 一定要是 np.DataFrame 类型的
    zeros=np.array([])
    for i in range(len(function_points)-1):
        product=function_points.iloc[i,1]*function_points.iloc[i+1,1]
        if product<=0:
            zeros=np.append(zeros,function_points.iloc[i,0])
        else:
            continue
    return zeros

#近似模块
def searching(i,alpha1,alpha2,theta,coordinate):
    #转为弧度制
    alpha1,alpha2=alpha1*pi/180,alpha2*pi/180
    R0=100
    beta_i=2/9*pi*(i-1)
    theta1=np.linspace(0,pi,500)
    fun=np.sin(alpha2+beta_i-theta1)*np.sin(alpha1)-np.sin(alpha1+theta1)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":theta1,
        "y":fun
        })
```

```
    theta2=findzero(points)
    theta2=theta2*180/pi
    Difference=abs(theta[1:len(theta)+1]-theta2)
    j=np.argmin(Difference)+1
    return j

#分类讨论
#case1 情况下 alpha1 最大
def case1(alpha1,alpha2,alpha12,j):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[j,1]*pi/180

    #解方程
    beta=np.linspace(varphi+pi,2*pi,500)
    fun=np.sin(-beta+varphi+alpha2)*np.sin(alpha1)-np.sin(alpha1-beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1-sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta


#case2 情况下 alpha2 最大
def case2(alpha1,alpha2,alpha12,j):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[j,1]*pi/180

    #解方程
    beta=np.linspace(varphi,varphi+pi,500)
    fun=np.sin(alpha2+beta-varphi)*np.sin(alpha1)-np.sin(alpha1+beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
```

```
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1+sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta

#case3 情况下 alpha12 最大
def case3(alpha1,alpha2,alpha12,j):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[j,1]*pi/180

    #解方程
    beta=np.linspace(0,varphi,500)
    fun=np.sin(-beta+varphi+alpha2)*np.sin(alpha1)-np.sin(alpha1+beta)*np.sin(alpha2)
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    d=coordinate.iloc[1,0]*np.sin(alpha1-sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta

#对数据进行分类:
def solution(i,alpha1,alpha2,alpha12):
    j=searching(i,alpha1,alpha2,theta,coordinate)
    if (alpha1>alpha2)&(alpha1>alpha12):
        d,sol_beta=case1(alpha1, alpha2, alpha12,j)
    elif (alpha2>alpha1)&(alpha2>alpha12):
        d,sol_beta=case2(alpha1, alpha2, alpha12,j)
    else:
        d,sol_beta=case3(alpha1, alpha2, alpha12,j)

    if (d<0):
        d=-d
        sol_beta=sol_beta-180
    return d,sol_beta


if __name__ == "__main__" :
```

```
#测试代码正确性
#四个孔分别填：要调整的飞机编号，α1，α2，α12
d,beta=solution(5,10.172,54.745,44.573)
print('无人机的极径为:')
print(d)
print('无人机的方位角为：')
print(beta)
```

附录四、

程序 3:问题 1.3 位置调整方案迭代计算程序

程序说明：基于 python 3.9.7 开发环境编译 编译解释器：Spyder

```
#加载库
import pandas as pd
import numpy as np
import math
import random
#定义圆周率
pi=3.1415926

#构建极坐标数据

#正确的情况
r=np.array([100]*9)
r=np.append(0,r)
theta=np.linspace(0,360-40,9)
theta=np.append(0,theta)

coordinate=pd.DataFrame({
    "r":r,
    "theta":theta
    })

#现实的情况
r_real=np.array([0,100,89,105,110,96,94,108,96,99])
theta_real=np.array([0,0,37.6,81,122,157,202,237,283,320.28
])

coordinate_real=pd.DataFrame({
    "r":r_real,
    "theta":theta_real
    })
```

```
30.
31.
32.
33.#计算零点
34.def findzero(function_points):   #function_points 一定要是
   np.DataFrame 类型的
35.    m=np.argmin(abs(function_points.iloc[:,1]))
36.    zeros=function_points.iloc[m,0]
37.    return zeros
38.
39.
40.#分类讨论
41.#case1 情况下 alpha1 最大
42.def case1(alpha1,alpha2,alpha12,i,j,selection1,selection2,m
   inum,maxum):
43.    #转为弧度制
44.    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha
   12*pi/180
45.    varphi=coordinate.iloc[j,1]*pi/180
46.
47.    #解方程
48.    beta=np.linspace(minum,maxum,500)
49.
50.    if (selection1==1):
51.        gramma1=alpha1-beta
52.    else:
53.        gramma1=alpha1+beta
54.
55.    if (selection2==1):
56.        gramma2=varphi-beta
57.    else :
58.        gramma2=beta-varphi
59.
60.    fun=np.sin(gramma2+alpha2)*np.sin(alpha1)-
   np.sin(gramma1)*np.sin(alpha2)
61.    points=pd.DataFrame({
62.        "x":beta,
63.        "y":fun
64.        })
65.    sol_beta=findzero(points)
66.    if (selection1==1):
67.        d=coordinate.iloc[1,0]*np.sin(alpha1-
   sol_beta)/np.sin(alpha1)
68.    elif (selection1==2):
```

```
        d=coordinate.iloc[1,0]*np.sin(alpha1+sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta


#对数据进行分类：
#i 代表要测量的无人机的编号，j 代表着发射信号的无人机的编号
def solution(alpha1,alpha2,alpha12,i,j):
    if (coordinate_real.iloc[i,1]>=180):
        selection1=1
    elif (coordinate_real.iloc[i,1]<=180):
        selection1=2

    if (coordinate_real.iloc[j,1]>=180):
        if (alpha1>=alpha2)&(alpha1>=alpha12):
            minum=0
            maxum=coordinate_real.iloc[j,1]*pi/180 -pi
            selection2=2
        elif (alpha2>=alpha1)&(alpha2>=alpha12):
            minum=pi
            maxum=coordinate_real.iloc[j,1]*pi/180
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
            minum=coordinate_real.iloc[j,1]*pi/180-pi
            maxum=pi
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
            minum=coordinate_real.iloc[j,1]*pi/180
            maxum=2*pi
            selection2=2

      else :
          if (alpha1>=alpha2)&(alpha1>=alpha12):
              minum=coordinate_real.iloc[j,1]*pi/180+pi
              maxum=2*pi
              selection2=1
          elif (alpha2>=alpha1)&(alpha2>=alpha12):
              minum=coordinate_real.iloc[j,1]*pi/180
              maxum=pi
              selection2=2
          elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
              minum=0
              maxum=coordinate_real.iloc[j,1]*pi/180
```

```
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
            minum=pi
            maxum=coordinate_real.iloc[j,1]*pi/180+pi
            selection2=2

    d,sol_beta=case1(alpha1, alpha2, alpha12, i, j, selection1, selection2, minum, maxum)
    return d,sol_beta

#计算角度 bac 的大小
def angle(coordinate,a,b,c):
    cauculate1=(coordinate.iloc[a,0]*np.cos(coordinate.iloc[a,1]*pi/180)\
                -coordinate.iloc[b,0]*np.cos(coordinate.iloc[b,1]*pi/180))
    cauculate2=(coordinate.iloc[a,0]*np.sin(coordinate.iloc[a,1]*pi/180)\
                -coordinate.iloc[b,0]*np.sin(coordinate.iloc[b,1]*pi/180))
    cauculate3=(coordinate.iloc[a,0]*np.cos(coordinate.iloc[a,1]*pi/180)\
                -coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
    cauculate4=(coordinate.iloc[a,0]*np.sin(coordinate.iloc[a,1]*pi/180)\
                -coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
    cauculate5=(coordinate.iloc[b,0]*np.cos(coordinate.iloc[b,1]*pi/180)\
                -coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
    cauculate6=(coordinate.iloc[b,0]*np.sin(coordinate.iloc[b,1]*pi/180)\
                -coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
    angle_1=cauculate1**2+cauculate2**2+cauculate3**2+cauculate4**2-cauculate5**2-cauculate6**2
    angle_2=2*np.sqrt(cauculate1**2+cauculate2**2)*np.sqrt(cauculate3**2+cauculate4**2)
    angle=np.arccos(angle_1/angle_2)*180/pi
    return angle

def createrandomnumbers(j):
```

```
141.     number=[]
142.     while (len(number)<3):
143.         x=random.randint(1,9)
144.         if (x not in number)&(x!=1)&(x!=j):
145.             number.append(x)
146.     return number
147.
148. def figureout(i):
149.     number=createrandomnumbers(i)
150.     alpha1=angle(coordinate_real,i,0,1)
151.     alpha2_1=angle(coordinate_real,i,0,number[0])
152.     alpha12_1=angle(coordinate_real,i,1,number[0])
153.     d_1,varphi_1=solution(alpha1,alpha2_1,alpha12_1,i)
154.     alpha2_2=angle(coordinate,i,0,number[1])
155.     alpha12_2=angle(coordinate,i,1,number[1])
156.     d_2,varphi_2=solution(alpha1,alpha2_2,alpha12_2,i)
157.     D=(d_1+d_2)/2
158.     Varphi=(varphi_1+varphi_2)/2
159.     return D,Varphi
160.
161.
162. def problem(i,j):
163.     alpha1=angle(coordinate_real,i,1,0)
164.     alpha2=angle(coordinate_real,i,j,0)
165.     alpha12=angle(coordinate_real,i,j,1)
166.     m,n=solution(alpha1,alpha2,alpha12,i,j)
167.     return m,n
168.
169. if __name__ == "__main__" :
170.     coordinate_new=coordinate_real
171.
172.     iterationnum=0
173.     while (abs(np.max(np.array(coordinate_real-
   coordinate)))>=0.8):
174.         for i in range(2,9+1):
175.             number=createrandomnumbers(i)
176.             d1,beta1=problem(i,number[0])
177.             d2,beta2=problem(i,number[1])
178.             d3,beta3=problem(i,number[2])
179.             D=(d1+d2)/2
180.             Beta=(beta1+beta2)/2
181.             coordinate_new.iloc[i,0]=coordinate.iloc[i,0]
   -D+coordinate_new.iloc[i,0]
```

```
                coordinate_new.iloc[i,1]=coordinate.iloc[i,1]
-Beta+coordinate_new.iloc[i,1]

            coordinate_real=coordinate_new
            iterationnum=iterationnum+1

        print('迭代次数为')
        print(iterationnum)
        print('结果数据')
        print(coordinate_real)
```

附录五、

程序 4：问题 2：方阵形编队位置调整方案迭代计算程序

程序说明：基于 python 3.9.7 开发环境编译 编译解释器：Spyder

```
#加载库
import pandas as pd
import numpy as np
import math
import random
#定义圆周率
pi=3.1415926

#构建极坐标数据

#正确的情况
r=np.array([0,70.7,50,70.7,50,70.7,50,70.7,50])
theta=np.array([0,0,45,90,135,180,225,270,315])
coordinate=pd.DataFrame({
    "r":r,
    "theta":theta
    })

#现实的情况
r_real=np.array([0,70.7,48,67.3,54,76.4,47,73.3,44])
theta_real=np.array([0,0,49,89.4,133.6,177,227.3,268.4,320]
)

coordinate_real=pd.DataFrame({
    "r":r_real,
    "theta":theta_real
    })
```

```
27.
28.
29.#计算零点
30.def findzero(function_points):   #function_points 一定要是
   np.DataFrame 类型的
31.    m=np.argmin(abs(function_points.iloc[:,1]))
32.    zeros=function_points.iloc[m,0]
33.    return zeros
34.
35.
36.#分类讨论
37.def case1(alpha1,alpha2,alpha12,i,j,selection1,selection2,m
   inum,maxum):
38.    #转为弧度制
39.    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha
   12*pi/180
40.    varphi=coordinate.iloc[j,1]*pi/180
41.
42.    #解方程
43.    beta=np.linspace(minum,maxum,500)
44.
45.    if (selection1==1):
46.        gramma1=alpha1-beta
47.    else:
48.        gramma1=alpha1+beta
49.
50.    if (selection2==1):
51.        gramma2=varphi-beta
52.    else :
53.        gramma2=beta-varphi
54.
55.    fun=np.sin(gramma2+alpha2)*np.sin(alpha1)*coordinate.il
   oc[j,0]-np.sin(gramma1)*np.sin(alpha2)*coordinate.iloc[1,0]
56.    points=pd.DataFrame({
57.        "x":beta,
58.        "y":fun
59.        })
60.    sol_beta=findzero(points)
61.    if (selection1==1):
62.        d=coordinate.iloc[1,0]*np.sin(alpha1-
   sol_beta)/np.sin(alpha1)
63.    elif (selection1==2):
64.        d=coordinate.iloc[1,0]*np.sin(alpha1+sol_beta)/np.s
   in(alpha1)
```

```
    sol_beta=sol_beta*180/pi
    return d,sol_beta


#对数据进行分类：
#i 代表要测量的无人机的编号，j 代表着发射信号的无人机的编号
def solution(alpha1,alpha2,alpha12,i,j):
    if (coordinate_real.iloc[i,1]>=180):
        selection1=1
    elif (coordinate_real.iloc[i,1]<=180):
        selection1=2

    if (coordinate_real.iloc[j,1]>=180):
        if (alpha1>=alpha2)&(alpha1>=alpha12):
            minum=0
            maxum=coordinate_real.iloc[j,1]*pi/180 -pi
            selection2=2
        elif (alpha2>=alpha1)&(alpha2>=alpha12):
            minum=pi
            maxum=coordinate_real.iloc[j,1]*pi/180
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
            minum=coordinate_real.iloc[j,1]*pi/180-pi
            maxum=pi
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
            minum=coordinate_real.iloc[j,1]*pi/180
            maxum=2*pi
            selection2=2

    else :
        if (alpha1>=alpha2)&(alpha1>=alpha12):
            minum=coordinate_real.iloc[j,1]*pi/180+pi
            maxum=2*pi
            selection2=1
          elif (alpha2>=alpha1)&(alpha2>=alpha12):
              minum=coordinate_real.iloc[j,1]*pi/180
              maxum=pi
              selection2=2
          elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
              minum=0
              maxum=coordinate_real.iloc[j,1]*pi/180
              selection2=1
          elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
```

```
109.                 minum=pi
110.                 maxum=coordinate_real.iloc[j,1]*pi/180+pi
111.                 selection2=2
112.
113.         d,sol_beta=case1(alpha1, alpha2, alpha12, i, j, selec
    tion1, selection2, minum, maxum)
114.         return d,sol_beta
115.
116.
117.     coordinate_real1=pd.DataFrame({
118.         "r":r_real,
119.         "theta":theta_real
120.         })
121.     iterationnum1=0
122.     coordinate_new1=coordinate_real1
123.     while (abs(np.max(np.array(coordinate_real1-
    coordinate)))>=0.8):
124.         coordinate_new1.iloc[:,0]=(coordinate.iloc[:,0]-
    coordinate_real1.iloc[:,0])/4+coordinate_real1.iloc[:,0]
125.         coordinate_new1.iloc[:,1]=(coordinate.iloc[:,1]-
    coordinate_real1.iloc[:,1])/4+coordinate_real1.iloc[:,1]
126.         coordinate_real1=coordinate_new1
127.         iterationnum1=iterationnum1+1
128.
129.
130.
131.     #计算角度 bac 的大小
132.     def angle(coordinate,a,b,c):
133.         cauculate1=(coordinate.iloc[a,0]*np.cos(coordinate.il
    oc[a,1]*pi/180)\
134.                     -
    coordinate.iloc[b,0]*np.cos(coordinate.iloc[b,1]*pi/180))
135.         cauculate2=(coordinate.iloc[a,0]*np.sin(coordinate.il
    oc[a,1]*pi/180)\
136.                     -
    coordinate.iloc[b,0]*np.sin(coordinate.iloc[b,1]*pi/180))
137.         cauculate3=(coordinate.iloc[a,0]*np.cos(coordinate.il
    oc[a,1]*pi/180)\
138.                     -
    coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
139.         cauculate4=(coordinate.iloc[a,0]*np.sin(coordinate.il
    oc[a,1]*pi/180)\
140.                     -
    coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
```

```
141.     cauculate5=(coordinate.iloc[b,0]*np.cos(coordinate.il
   oc[b,1]*pi/180)\
142.                 -
   coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
143.     cauculate6=(coordinate.iloc[b,0]*np.sin(coordinate.il
   oc[b,1]*pi/180)\
144.                 -
   coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
145.     angle_1=cauculate1**2+cauculate2**2+cauculate3**2+cau
   culate4**2-cauculate5**2-cauculate6**2
146.     angle_2=2*np.sqrt(cauculate1**2+cauculate2**2)*np.sqr
   t(cauculate3**2+cauculate4**2)
147.     angle=np.arccos(angle_1/angle_2)*180/pi
148.     return angle
149.
150. #随机生成数
151. def createrandomnumbers(j):
152.     number=[]
153.     while (len(number)<2):
154.         x=random.randint(1,8)
155.         if (x not in number)&(x!=1)&(x!=j)&(x!=5)&(x!=j+4
   )&(x!=j-4):
156.             number.append(x)
157.     return number
158.
159.
160. #执行结果
161. def problem(i,j):
162.     alpha1=angle(coordinate_real,i,1,0)
163.     alpha2=angle(coordinate_real,i,j,0)
164.     alpha12=angle(coordinate_real,i,j,1)
165.
166.     m,n=solution(alpha1,alpha2,alpha12,i,j)
167.
168.     return m,n
169.
170.
171. #执行主程序
172. if __name__ == "__main__" :
173.     coordinate_new=coordinate_real
174.
175.     iterationnum=0
176.     for k in range(16):
177.         for i in range(2,8+1):
```

```
                number=createrandomnumbers(i)
                d1,beta1=problem(i,number[0])
                d2,beta2=problem(i,number[1])
                D=(d1+d2)/2
                Beta=(beta1+beta2)/2
                coordinate_new.iloc[i,0]=coordinate.iloc[i,0]-D+coordinate_new.iloc[i,0]
                coordinate_new.iloc[i,1]=coordinate.iloc[i,1]-Beta+coordinate_new.iloc[i,1]

            coordinate_real=coordinate_new
            iterationnum=iterationnum+1


        print('迭代次数为')
        print(iterationnum1)
        print('结果数据')
        print(coordinate_real1)
```

附录六、

程序 5：问题 2：锥形编队位置调整方案迭代计算程序

程序说明：基于 python 3.9.7 开发环境编译 编译解释器：Spyder

```
#加载库
import pandas as pd
import numpy as np
import math
import random
#定义圆周率
pi=3.1415926

#构建极坐标数据

#正确的情况
r=np.array([100*1.732,np.sqrt(17500),np.sqrt(17500),100,50*1.732,100,50*1.732,50,50,50*1.732,100,50,0,50,100])
theta=np.array([0,np.arctan(1/(3*1.732)),-np.arctan(1/(3*1.732)),pi/6,0,11*pi/6,pi/3,pi/6,11*pi/6,5*pi/3,pi/2,pi/2,0,3*pi/2,3*pi/2])*180/pi
coordinate=pd.DataFrame({
    "r":r,
    "theta":theta
```

```
    })

#现实的情况
r_real=np.array([100*1.732,134,128,94,93,96,85,48,53,89,102,55,3,46,92])
theta_real=np.array([0,12,-10.1,28,2,329,58,28,329,301,86,92,2,272,268])

coordinate_real=pd.DataFrame({
    "r":r_real,
    "theta":theta_real
    })


#计算零点
def findzero(function_points):   #function_points 一定要是 np.DataFrame 类型的
    m=np.argmin(abs(function_points.iloc[:,1]))
    zeros=function_points.iloc[m,0]
    return zeros


#分类讨论
def case1(alpha1,alpha2,alpha12,i,j,selection1,selection2,minum,maxum):
    #转为弧度制
    alpha1,alpha2,alpha12=alpha1*pi/180,alpha2*pi/180,alpha12*pi/180
    varphi=coordinate.iloc[j,1]*pi/180

    #解方程
    beta=np.linspace(minum,maxum,500)

    if (selection1==1):
        gramma1=alpha1-beta
    else:
        gramma1=alpha1+beta

    if (selection2==1):
        gramma2=varphi-beta
    else :
        gramma2=beta-varphi

```

```
    fun=np.sin(gramma2+alpha2)*np.sin(alpha1)*coordinate.iloc[j,0]-np.sin(gramma1)*np.sin(alpha2)*coordinate.iloc[1,0]
    points=pd.DataFrame({
        "x":beta,
        "y":fun
        })
    sol_beta=findzero(points)
    if (selection1==1):
        d=coordinate.iloc[1,0]*np.sin(alpha1-sol_beta)/np.sin(alpha1)
    elif (selection1==2):
        d=coordinate.iloc[1,0]*np.sin(alpha1+sol_beta)/np.sin(alpha1)
    sol_beta=sol_beta*180/pi
    return d,sol_beta


#对数据进行分类：
#i 代表要测量的无人机的编号，j 代表着发射信号的无人机的编号
def solution(alpha1,alpha2,alpha12,i,j):
    if (coordinate_real.iloc[i,1]>=180):
        selection1=1
    elif (coordinate_real.iloc[i,1]<=180):
        selection1=2

    if (coordinate_real.iloc[j,1]>=180):
        if (alpha1>=alpha2)&(alpha1>=alpha12):
            minum=0
            maxum=coordinate_real.iloc[j,1]*pi/180 -pi
            selection2=2
        elif (alpha2>=alpha1)&(alpha2>=alpha12):
            minum=pi
            maxum=coordinate_real.iloc[j,1]*pi/180
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
            minum=coordinate_real.iloc[j,1]*pi/180-pi
            maxum=pi
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
            minum=coordinate_real.iloc[j,1]*pi/180
            maxum=2*pi
            selection2=2

    else :
```

```
        if (alpha1>=alpha2)&(alpha1>=alpha12):
            minum=coordinate_real.iloc[j,1]*pi/180+pi
            maxum=2*pi
            selection2=1
        elif (alpha2>=alpha1)&(alpha2>=alpha12):
            minum=coordinate_real.iloc[j,1]*pi/180
            maxum=pi
            selection2=2
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i<j):
            minum=0
            maxum=coordinate_real.iloc[j,1]*pi/180
            selection2=1
        elif (alpha12>=alpha1)&(alpha12>=alpha2)&(i>j):
            minum=pi
            maxum=coordinate_real.iloc[j,1]*pi/180+pi
            selection2=2

    d,sol_beta=case1(alpha1, alpha2, alpha12, i, j, selection1, selection2, minum, maxum)
    return d,sol_beta


coordinate_real1=pd.DataFrame({
    "r":r_real,
    "theta":theta_real
    })
iterationnum1=0
coordinate_new1=coordinate_real1
while (abs(np.max(np.array(coordinate_real1-coordinate)))>=0.8):
    coordinate_new1.iloc[:,0]=(coordinate.iloc[:,0]-coordinate_real1.iloc[:,0])/4+coordinate_real1.iloc[:,0]
    coordinate_new1.iloc[:,1]=(coordinate.iloc[:,1]-coordinate_real1.iloc[:,1])/4+coordinate_real1.iloc[:,1]
    coordinate_real1=coordinate_new1
    iterationnum1=iterationnum1+1

#计算角度 bac 的大小
def angle(coordinate,a,b,c):
    cauculate1=(coordinate.iloc[a,0]*np.cos(coordinate.iloc[a,1]*pi/180)\
                -coordinate.iloc[b,0]*np.cos(coordinate.iloc[b,1]*pi/180))
```

```
133.     cauculate2=(coordinate.iloc[a,0]*np.sin(coordinate.il
   oc[a,1]*pi/180)\
134.                 -
   coordinate.iloc[b,0]*np.sin(coordinate.iloc[b,1]*pi/180))
135.     cauculate3=(coordinate.iloc[a,0]*np.cos(coordinate.il
   oc[a,1]*pi/180)\
136.                 -
   coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
137.     cauculate4=(coordinate.iloc[a,0]*np.sin(coordinate.il
   oc[a,1]*pi/180)\
138.                 -
   coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
139.     cauculate5=(coordinate.iloc[b,0]*np.cos(coordinate.il
   oc[b,1]*pi/180)\
140.                 -
   coordinate.iloc[c,0]*np.cos(coordinate.iloc[c,1]*pi/180))
141.     cauculate6=(coordinate.iloc[b,0]*np.sin(coordinate.il
   oc[b,1]*pi/180)\
142.                 -
   coordinate.iloc[c,0]*np.sin(coordinate.iloc[c,1]*pi/180))
143.     angle_1=cauculate1**2+cauculate2**2+cauculate3**2+cau
   culate4**2-cauculate5**2-cauculate6**2
144.     angle_2=2*np.sqrt(cauculate1**2+cauculate2**2)*np.sqr
   t(cauculate3**2+cauculate4**2)
145.     angle=np.arccos(angle_1/angle_2)*180/pi
146.     return angle
147.
148. #随机生成数
149. def createrandomnumbers(j):
150.     number=[]
151.     while (len(number)<2):
152.         x=random.randint(1,8)
153.         if (x not in number)&(x!=1)&(x!=j)&(x!=5)&(x!=j+4
   )&(x!=j-4):
154.             number.append(x)
155.     return number
156.
157.
158. #问题分解
159. def problem(i,j):
160.     alpha1=angle(coordinate_real,i,1,0)
161.     alpha2=angle(coordinate_real,i,j,0)
162.     alpha12=angle(coordinate_real,i,j,1)
163.
```

```
164.      m,n=solution(alpha1,alpha2,alpha12,i,j)
165.
166.      return m,n
167.
168.
169.
170.  #执行主程序
171.  if __name__ == "__main__" :
172.      coordinate_new=coordinate_real
173.
174.      iterationnum=0
175.      for k in range(16):
176.          for i in range(2,8+1):
177.              number=createrandomnumbers(i)
178.              d1,beta1=problem(i,number[0])
179.              d2,beta2=problem(i,number[1])
180.              D=(d1+d2)/2
181.              Beta=(beta1+beta2)/2
182.              coordinate_new.iloc[i,0]=coordinate.iloc[i,0]
    -D+coordinate_new.iloc[i,0]
183.              coordinate_new.iloc[i,1]=coordinate.iloc[i,1]
    -Beta+coordinate_new.iloc[i,1]
184.
185.          coordinate_real=coordinate_new
186.          iterationnum=iterationnum+1
187.
188.
189.      print('迭代次数为')
190.      print(iterationnum1)
191.      print('结果数据')
192.      print(coordinate_real1)
```