基于蒙特卡罗方法的无人机纯方位无源定位模型

摘要

无人机集群在编队飞行时，需要使用定位技术来调整相对位置，以减小可能产生的偏差从而保持预定队形。为避免外来信号干扰，同时尽可能少发送电磁信号、降低功率损耗，可使用**纯方位无源定位**的方法。本文针对这一方法进行建模分析，根据无人机的相对夹角信息，给出了有效定位与调整方案。

对于**问题 1.(1)**，首先将无人机定位模型转化为**平面几何问题**。通过对无人机方向信息 α_1、α_2 和已知编号的发射信号无人机 A、B 的位置信息建立数学等式，表示出被动接受信号无人机的一般性坐标表达式:$(\frac{K_{ac}x_a - K_{bc}x_b - y_a + y_b}{K_{ac} - K_{bc}}, \frac{K_{ac}K_{bc}(x_a - x_b) + K_{ac}y_b - K_{bc}y_a}{K_{ac} - K_{bc}})$。使用**蒙特卡罗方法**验证上述结果，通过批量生成服从**正态分布**的随机数据作为接受信号无人机的输入信息，模拟定位过程，并计算**定位误差率**，对最终结果进行误差分析。最终得到的相对误差在 **0 值**附近微小波动，证明了模型的科学性与可行性。

对于**问题 1.(2)**，首先假设增加 1 架发射信号无人机，就可实现无人机的有效定位。根据问题 1.(1) 可知只要 3 架编号已知且位置无偏差的发射信号无人机就可以实现有效定位。由于 FY00、FY01 编号已知，因此只需要确定无人机编号 T_x。将确定的 3 架发射信号无人机生成的方向信息角度对 $< \alpha_1, \alpha_2 >$ 数组与方向信息的角度对表进行**遍历比对**，可以确定无人机编号 T_x。随后随机生成服从**均匀分布**的数据作为无人机编号 T_x。再使用**蒙特卡罗方法模拟遍历比对过程**，计算得准确率稳定在 **94.5%** 左右。为优化算法，考虑误差出现条件，发现推测编号与正确编号对应的误差相近，因此通过测试数据设置**角度差值的最小限度**，增加 2 架无人机发射信号，准确率提高至 **96.8%**。考虑到**无人机要尽量少向外发射电磁波信号**，只需要增加 1 架发射信号无人机。当误差产生时，则增加 2 架发射信号无人机完成无人机的有效定位。

对于**问题 1.(3)**，首先从 FY01、FY02、…、FY09 中随机抽取 3 架无人机作为发射信号无人机，参考问题 1.(1)，得到某 1 架被动接受信号无人机的 C_3^2 种位置结果，取均值得到**推断位置**。计算剩余 6 架无人机的推断位置后，**调整无人机位置并更新迭代信息**。最后通过不断的**迭代调整**，发现无人机的角度和半径的方差逐渐趋近于**0**，说明给出的调整方案具有合理性。

对于**问题 2**，首先将 FY05、FY08、FY09 无人机单独抽出，将这 3 架无人机组成的三角形通过**迭代调整**形成正三角形。再以这个正三角形为基准，确定编队中其他无人机的**相对位置**。又考虑到模型的特殊性，可以确定每一个无人机的**理想相对位置**，并以此进行无人机的调整。最后，通过迭代模拟，发现无人机 FY05、FY08、FY09 形成的三角形边长方差逐渐趋近于**0**，说明给出的调整方案具有合理性。

关键字：蒙特卡罗方法　搜索策略　迭代调整　方差分析

一、 问题重述

1.1 问题背景

得益于无人机强大的机动性和便携性，无人机平台搭载视觉跟踪技术在路径规划、自主着陆、飞行避障等方面受到了广泛关注。[1] 随着电子干扰、隐身技术和反辐射导弹等雷达对抗技术的产生和发展，传统雷达系统已很难象过去一样发挥威力，而且其自身的生存也受到严重威胁。[2] 因此在无人机编队飞行时，采用纯方位无源定位的方法，即由编队中某几架无人机发射信号、其余无人机被动接收信号，从中提取出方向信息进行定位，来调整无人机的位置。这种方式既可以向外发送较少电磁波信号，也有利于保持编队队形，消除飞行过程中的位置偏差。

1.2 问题提出

问题 1：编队由 10 架无人机组成，形成圆形编队，其中 9 架无人机 (编号 FY01 FY09) 均匀分布在某一圆周上，另 1 架无人机 (编号 FY00) 位于圆心 (见图 1)。无人机基于自身感知的高度信息，均保持在同一个高度上飞行。

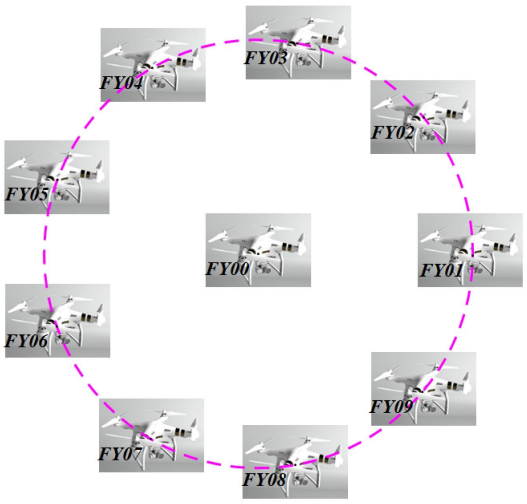

图 1　圆形无人机编队示意图

(1) 位置无偏差且编号已知的无人机 FY00(位于圆心) 和编队另 2 架无人机发射信号，其余位置略有偏差的无人机接收信号，建立被动接受信号无人机的定位模型。

(2) 已知发射信号无人机位置无偏差。某位置略有偏差的无人机收到编号 FY00 和 FY01 无人机发射的信号，则求还需要几架未知编号无人机发射信号才能实现有效定位。

(3) 按编队要求，1 架无人机位于圆心，另 9 架无人机均匀分布在半径为 100m 的圆周上。已知无人机初始位置，每次选择 FY00 的无人机和圆周上最多 3 架无人机遂行发

射信号，给出具体调整方案使 9 架无人机最终均匀发布在某个圆周上。

问题 2：仍考虑纯方位无源定位，设计其他编形队形的无人机位置调整方案。例如锥形编队队形 (直线上相邻两架无人机的间距相等，如 50m)。

二、 问题分析

2.1 问题一的分析

2.1.1 问题 1.(1) 的分析:

问题可以简化为已知圆心和圆周上两点来求其他未知点的位置。建立以 FY00 为原点，以过 FY00、FY01 的线为 x 轴，垂直 x 轴向上的射线为 y 轴的**坐标系定位模型**。两个发射信号的无人机位置为已知量，可以通过平面几何计算出未知点的坐标位置。

考虑到计算过程中产生的误差，需要通过构造数值样本检测计算结果的准确性。由于已知条件无人机位置略有偏差，我们假设无人机的位置在理想位置坐标附近呈**正态分布**，因此随机生成服从正态分布的一定规模数据，再使用**蒙特卡罗方法**模拟模型的产生过程，根据计算的误差率探究所解结果的科学性和可行性。

2.1.2 问题 1.(2) 的分析:

问题可以简化为已知圆心和圆周上一个点，还需要几个未知点可以完成其余点的位置定位。

我们先假设还需要 1 架编号未知的发射信号无人机就可以实现无人机的有效定位。参考问题 1.(1)，可知仅需要求出该发射信号无人机的编号 T_x 就可以实现无人机的有效定位。由于发射信号无人机的编号与方向信息的角度对 $<\alpha_1,\alpha_2>$ **一一对应**，随机生成服从**均匀分布**的一定规模数据，再采用**遍历算法**对得到的角度对 $<\alpha_1',\alpha_2'>$ 与理想位置角度对 $<\alpha_1,\alpha_2>$ 进行误差比对，最终确定该未知发射信号无人机的编号。若得到的准确率高，则可以证明假设成立，即还需要 1 架发射信号无人机就可以实现无人机的有效定位。同时为优化算法，可以考虑误差出现具体条件，通过增加发射信号无人机来减小误差。

2.1.3 问题 1.(3) 的分析:

问题可以简化为已知圆心，每次至多再任选三个在圆周附近的未知点来多次调整所有未知点的位置，**逐步收敛**直至未知点的位置均匀分布在一个圆周上。

为了优化模型，每次在 FY01、FY02、...、FY09 中随机抽取 3 架无人机作为信号发射机。考虑到无人机位置略有偏差，仅选择 FY00 和 3 架无人机中的任意 2 架作为信号发射机，因此对于被动接受信号无人机 C 可以得到 C_3^2 种位置坐标，取均值得到推断出的位置坐标。**逐步更新**式 (5) 的 R、θ，使得到的无人机位置不断收敛，直至 9 架无人机近似均匀分布在一个圆周上。通过计算机模拟无人机的位置变化图得到具体调整方案。

2.2 问题二的分析

问题可以简化为已知平面上任意三个点，将三个点构成的三角形调整为正三角形，并且根据这三个点向外延展得到完整的锥形图。

考虑正三角形的特殊性，首先将 A 作为接受信号无人机，将 B、C 作为发射信号无人机。把得到的方向信息与 60° 进行比较，根据比较结果对接收信号无人机进行位置的实时调整。当方向信息 $\alpha = 60°$ 时，则调整下一个无人机。经过反复多次调整使得 A、B、C 构成一个正三角形。以此正三角形为基准，能够明确得到锥形图中其他的无人机的理想位置，从而通过调整无人机位置，完成无人机锥形编队队形的调整。

三、 模型假设

1. 假设编队中无人机的高度都相同，即都位于同一水平面上。

2. 假设无人机所接收到的夹角信息不存在偏差，不考虑角度测量仪器精度有限所带来计算误差。

3. 假设无人机的微小位置偏差基本符合正态分布。

4. 假设编队无人机之间的信号均能正确传递，不考虑其他无人机或障碍物阻挡而导致信息传递失败的情况。

5. 假设模型仅考虑数学几何问题，不考虑温度、电磁波的折射与反射、风力等其它因素的影响。

四、 符号说明

符号	意义	单位
R	圆周半径	m
α_i	无人机接受的方向信息	rad
T_i	无人机 i 的编号	/
K_{ij}	过无人机 i，j 直线的斜率	/
$\overline{\Delta\theta}$	极角偏差均值	rad
$S_{\Delta\theta}$	极角偏差标准差值	rad
$\overline{\Delta\rho}$	极径偏差均值	m
$S_{\Delta\rho}$	极径偏差标准差值	m

五、 模型的建立与求解

5.1 问题 1.(1) 模型的建立与求解

5.1.1 模型建立

由于无人机均保持同一高度飞行，可以将该题抽象为平面几何。建立以 FY00 为原点，FY00、FY01 连线为 x 轴，垂直 FY00、FY01 连线向上为 y 轴的直角坐标系 (如图 2)。

假设发射信号的 2 架无人机为 A、B，编号为 T_A、T_B，任意一被动接受信号无人机为 C。现要求根据 A、B 的坐标，以及 A、B、O 与 C 连线的角度 α_1、α_2。求 C 坐标。

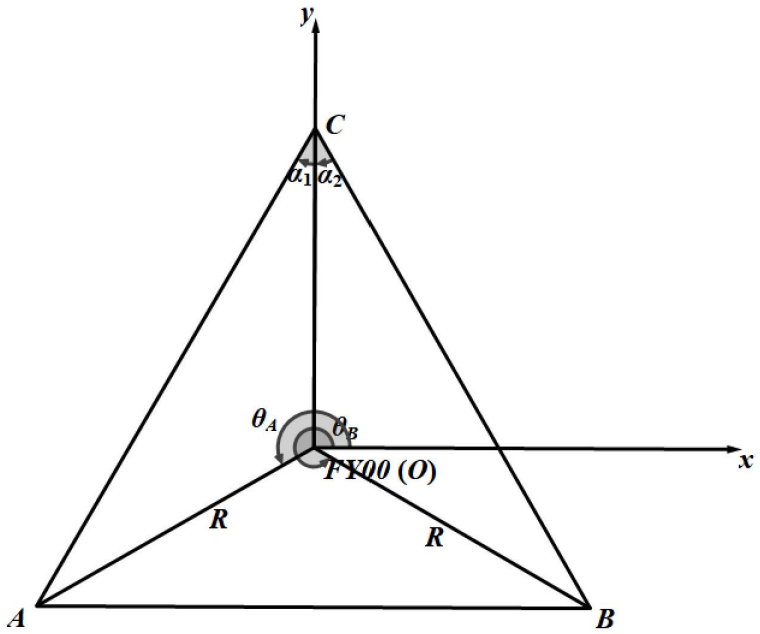

图 2　无人机的平面抽象模型

5.1.2 模型求解

考虑到无人机均匀分布在同一圆周上，可根据已知编号求出 2 架发射信号的无人机的方位角:

$$\begin{cases} \theta_A = \dfrac{2\pi}{9}(T_A - 1) \\ \\ \theta_B = \dfrac{2\pi}{9}(T_B - 1) \end{cases} \tag{1}$$

再根据方位角写出直角坐标:

$$A(Rcos\theta_A, Rsin\theta_A),\ B(Rcos\theta_B, Rsin\theta_B) \tag{2}$$

对于任意被动接收信号的无人机 C，假设其坐标为 (x_c, y_c)，则分别写出直线 AC、直线 OC、直线 BC 的斜率:

$$\begin{cases} K_{AC} = \dfrac{y_c - Rsin\theta_A}{x_c - Rcos\theta_A} \\ K_{OC} = \dfrac{y_c}{x_c} \\ K_{BC} = \dfrac{y_c - Rsin\theta_B}{x_c - Rcos\theta_B} \end{cases} \tag{3}$$

根据和差角公式，用这三条直线的斜率表示出 $tan\alpha_1$ 和 $tan\alpha_2$:

$$\begin{cases} tan\alpha_1 = \dfrac{K_{OC} - K_{AC}}{1 + K_{OC} \cdot K_{AC}} \\ tan\alpha_2 = \dfrac{K_{BC} - K_{OC}}{1 + K_{BC} \cdot K_{OC}} \end{cases} \tag{4}$$

可求解出未知点 C 的坐标:

$$\begin{cases} \boldsymbol{x_c} = \dfrac{K_{ac}x_a - K_{bc}x_b - y_a + y_b}{K_{ac} - K_{bc}} \\ \boldsymbol{y_c} = \dfrac{K_{ac}K_{bc}(x_a - x_b) + K_{ac}y_b - K_{bc}y_a}{K_{ac} - K_{bc}} \end{cases} \tag{5}$$

其中 K_{ac}、K_{bc}、K_{oc}、x_a、y_a、x_b、y_b 分别为:

$$K_{ac} = \frac{\tan^2(\alpha_1)\tan(\alpha_2)(y_a - y_b) - \tan^2(\alpha_1)x_b - \tan(\alpha_1)\tan(\alpha_2)x_b + \tan(\alpha_1)y_b + \tan(\alpha_2)y_a}{\tan^2(\alpha_1)\tan(\alpha_2)(x_a - x_b) + \tan^2(\alpha_1)y_b + \tan(\alpha_1)\tan(\alpha_2)y_b + \tan(\alpha_1)x_b + \tan(\alpha_2)x_a},$$

$$K_{bc} = \frac{\tan(\alpha_1)\tan^2(\alpha_2)(y_a - y_b) - \tan(\alpha_1)\tan(\alpha_2)x_a - \tan(\alpha_1)y_b - \tan^2(\alpha_2)x_a - \tan(\alpha_2)y_a}{\tan(\alpha_1)\tan^2(\alpha_2)(x_a - x_b) + \tan(\alpha_1)\tan(\alpha_2)y_a - \tan(\alpha_1)x_b + \tan^2(\alpha_2)y_a - \tan(\alpha_2)x_a},$$

$$K_{oc} = \frac{\tan(\alpha_1)\tan(\alpha_2)(x_a - x_b) + \tan(\alpha_1)y_b + \tan(\alpha_2)y_a}{\tan(\alpha_1)\tan(\alpha_2)(-y_a + y_b) + \tan(\alpha_1)x_b + \tan(\alpha_2)x_a},$$

$$\begin{cases} x_a = R\cos\theta_A, y_a = R\sin\theta_A \\ x_b = R\cos\theta_B, y_b = R\sin\theta_B. \end{cases}$$

5.1.3 结果分析

蒙特卡罗方法 (Monte Carlo, MC)，又称统计模拟方法或随机抽样技术，由乌拉姆和冯 · 诺伊曼在 20 世纪 40 年代中期为解决研制核武器中的计算问题首先提出并加以运用，是一种通过计算机运算能力来模拟随机过程的数值方法[3]。

考虑到接收信号无人机的位置略有偏差，本文结合实际情况与题目的限定条件，假设这种偏差较小且近似服从**正态分布**。基于该假设，本文结合问题一第 3 小问给定的表格数据，计算得到角度偏差与半径偏差的均值与标准差，由此随机生成服从正态分布的

一定规模的数据，从而使用蒙特卡罗方法，使用生成的数据信息模拟定位过程，实际应用所建立的定位模型，以评估模型的实施效果与准确性。

a) 问题建立

设待定位无人机的实际位置**相对于无偏差位置的位移**，在极坐标系下可表示为$(\Delta\rho, \Delta\theta)$，且 $\Delta\rho$ 和 $\Delta\theta$ 均近似服从正态分布，即

$$\Delta\rho \sim N(\overline{\Delta\rho}, S_{\Delta\rho}^2),\ \Delta\theta \sim N(\overline{\Delta\theta}, S_{\Delta\theta}^2),$$

其中 $\overline{\Delta\rho}$、$S_{\Delta\rho}$、$\overline{\Delta\theta}$、$S_{\Delta\theta}$ 为常数。

如表 1 所示，设定正态分布的数据:

表 1　正态分布数据

$\overline{\Delta\rho}$	$S_{\Delta\rho}$	$\overline{\Delta\theta}$	$S_{\Delta\theta}$
4.4444	5.9275	0.4444	0.1606

最终, 由此随机生成的 $\Delta\rho$、$\Delta\theta$ 符合正态分布 (如图 3、4)。

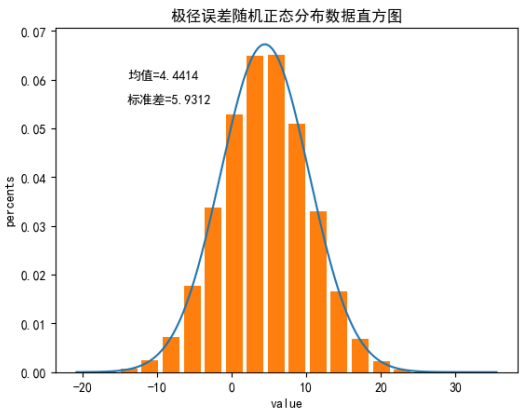

图 3　$\Delta\rho$ 的正态分布图

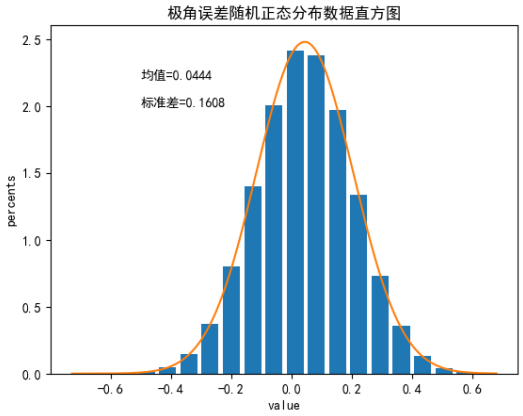

图 4　$\Delta\theta$ 的正态分布图

b) 生成随机数并计算误差

依据得到的均值与标准差，随机生成符合正态分布的偏差数据，将其赋予给随机选中的需要接受信息的无人机，使其位置略微出现偏差。同时随机选择本次除 FY00 无人机外，向其发送方向信息的另两架无人机，从而使用蒙特卡罗方法进行模拟，计算得到定位误差率，由此进行算法正确性的检验。

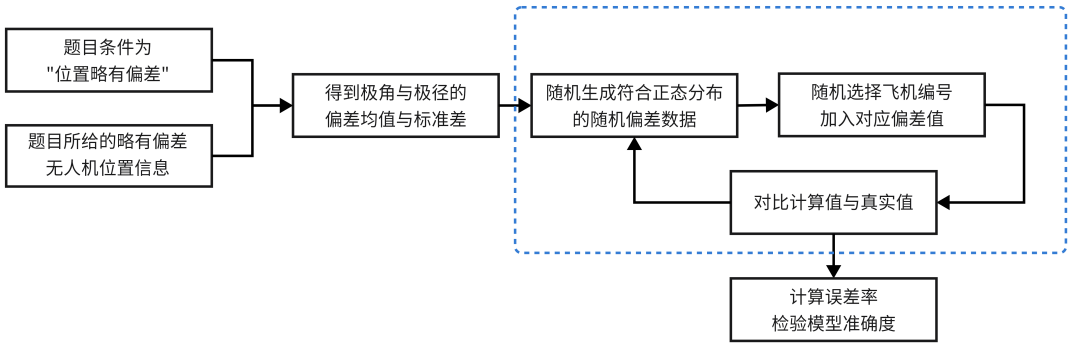

图 5　蒙特卡罗模拟算法流程

在进行了 1000 组数据的生成与模拟定位过程后，相关数据与结果如下，依据题目信息，需要接受信息的无人机 (FYC) 已知发送信息的无人机编号 (FYA 与 FYB) 与所接受到的方向信息 (α_1 和 α_2)。(部分数据见表 2)

表 2　发射与接收信号的方向信息

FYA(发射)	FYB(发射)	FYC(接收)	α_1	α_2
FY08	FY02	FY03	-0.12405	1.109455
FY08	FY09	FY06	0.817315	-0.475199
...	...	...	...	...
FY08	FY09	FY04	0.19524	0.146392
FY01	FY05	FY07	0.585961	0.677784

无人机依据已知信息，根据前文中所建立的定位模型，可以计算得到自身发生偏差后所在位置，即实现定位过程。将所得位置信息与实际的偏差坐标进行比对。(部分结果见表 3)

表 3　接收信号机位置

接收信号机编号	实际坐标 (直角坐标)	定位坐标 (直角坐标)
FY03	(8.50202,105.279612)	(8.50202,105.279612)
FY06	(-98.883609,-26.831187)	(-98.883609,-26.831187)
...	...	...
FY04	(-56.595766,87.614632)	(-56.595766,87.614632)
FY07	(-32.419756,-112.508160)	(-32.419756,-112.508160)

将生成的 1,000 组数据输入所建立的被动接受信号无人机定位模型，记录每组定位信息与实际位置信息的误差，最终计算其平均值，从而分别可以得到每组横轴坐标与纵轴坐标的相对误差。(表 4)

同时记录所有相对误差值，其分布信息见图 7，再取其平均得到横轴坐标与纵轴坐标的相对误差均值，

$$\begin{cases} \bar{\boldsymbol{x}} = 1.004601 \times 10^{-15} \\ \\ \bar{\boldsymbol{y}} = -1.417207 \times 10^{-15} \end{cases} \tag{6}$$

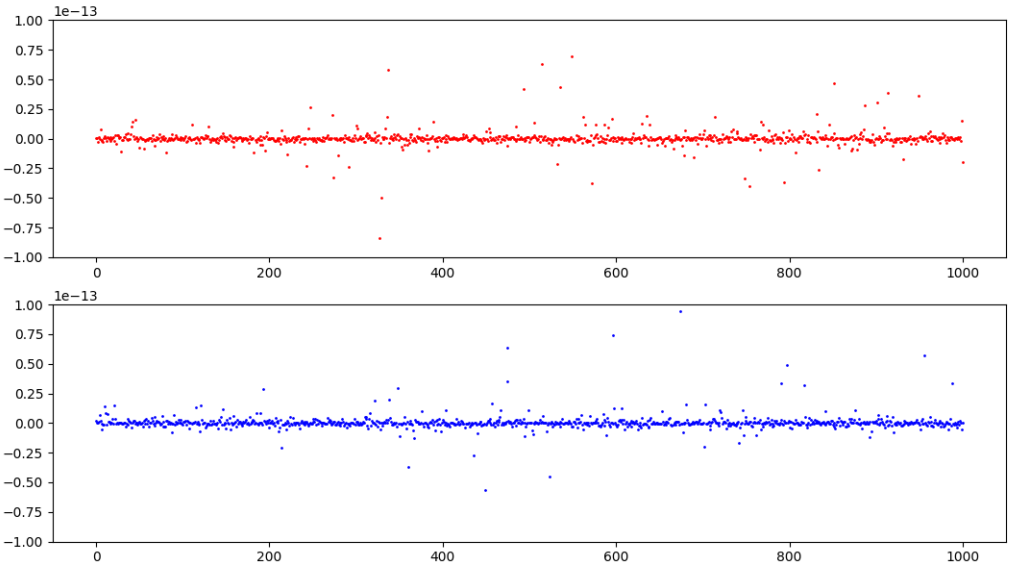

图 6　被动接收信号无人机误差分布 (上图为 x 轴坐标，下图为 y 轴坐标)

c) 误差分析

根据表 3 可知，计算值与测试值之间差异很小，两者基本完全一致。由表 4 数据，其相对误差在 **0 值**附近微小波动。依据图 6，波动值几乎完全保持在 10^{-13} 以内，从而可以说明当前所建定位模型的**科学性与可行性**。

表 4　计算结果的坐标误差率

接收信号机	横轴相对误差	纵轴相对误差
FY03	-1.42×10^{-14}	-5.40×10^{-16}
FY06	2.87×10^{-15}	9.93×10^{-11}
...	...	...
FY04	1.13×10^{-15}	-6.49×10^{-16}

5.2 问题 1.(2) 模型的建立与求解

5.2.1 模型建立

模型建立的基本流程:

根据题设，被动接受信号的无人机需要接收 FY00、FY01 以及另外若干编号未知的无人机信号。由于无人机要尽量少向外发射电磁波信号，首先考虑增加一架无人机的发射信号。此时接受信号的无人机持有 FY00、FY01 的编号和发送的角度信号，以及另外一架未知编号的无人机所发送的角度信号。

根据尚未出现偏差的无人机标准编队位置，可以得到所有可能存在的发送角度信号组合。将 FY00 标记为点 O，FY01 标记为点 A，同时将未固定的未知编号无人机标记为点 B，以及被动接受信号的无人机标记为点 C。(如图 7) 由题意，无人机每次接收的角度信号组合包括 $\angle ACO, \angle BCO, \angle BCA$, 考虑信息的必要性，我们选择记录其中所有情况下的 $\angle ACO, \angle BCO$ 作为标准角度序对 $< \alpha_1, \alpha_2 >$ 并以此建立表格。(角度制数据见表 5)

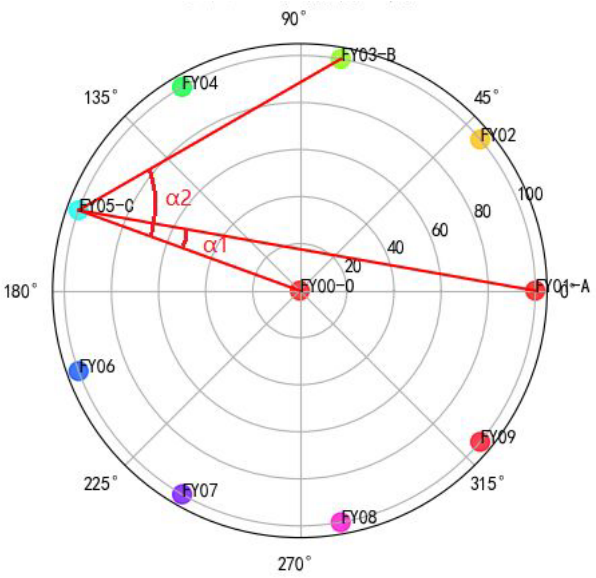

图 7　无人机标准编队示意图

依据表中数据及其对应关系，我们可以发现，发送信号的无人机编号和接收信号无人机编号的组合，与角度序对呈现一一对应的关系，也就是说，当已知当前被动接收信号的无人机编号，也同时已知其接收到的一组角度信息时，除 FY00 与 FY01 之外的另一架发送信号的无人机编号是**唯一且可以确定**的。

而当确定本身未知编号的无人机编号时，即出现了三架发射信号的无人机编号均已知的情况，即此时与问题一中第一小问情况完全相同，从而可以复用此前建立的定位模型。

而能否正确推测出无人机编号，理论上是完全可行的，实际上则需要考虑位置偏差

产生的影响，即是否存在由于位置偏差较大的情况，使接受到的角度序对匹配到错误的标准序对，从而导致推测出错误无人机编号。基于此，我们需要随机产生大量数据进行测试，即使用蒙特卡罗模拟方法，从而得到具有统计规律的成功概率，使该模型的可行性与可靠性能够得到数据上的支撑。

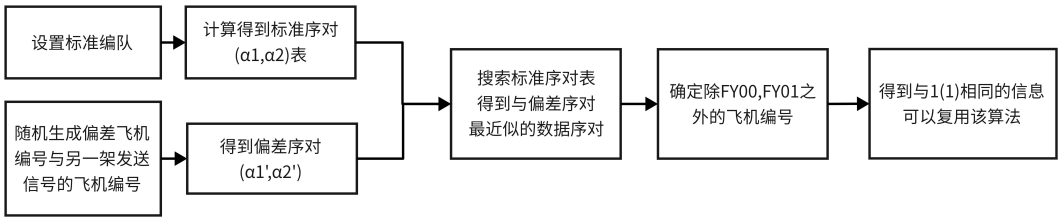

图 8　问题 1.(2) 模型建立流程

5.2.2 模型求解

首先，建立标准序对 $< \alpha_1, \alpha_2 >$ 表 (如表 5, 需要说明的是，为直观表现角度序对与编号组合之间的一一对应关系，表中展示**角度制**序对表 $< \beta_1, \beta_2 >$)。

表 5　标准角度制序对 $< \beta_1, \beta_2 >$ 数据

角度对 / ° 接收信号无人机编号 / 发射信号无人机编号	FY02	FY03	FY04	FY05	FY06	FY07	FY08	FY09
FY02	(0,0)*	(-50,70)	(-30,50)	(-10,30)	(10,10)	(30,-10)	(50,-30)	(70,-50)
FY03	(-70,-70)	(0,0)*	(-30,70)	(-10,50)	(10,30)	(30,10)	(50,-10)	(70,-30)
FY04	(-70,-50)	(-50,-70)	(0,0)*	(-10,70)	(10,50)	(30,30)	(50,10)	(70,-10)
FY05	(-70,-30)	(-50,-50)	(-30,-70)	(0,0)*	(10,70)	(30,50)	(50,30)	(70,10)
FY06	(-70,-10)	(-50,-30)	(-30,-50)	(-10,-70)	(0,0)*	(30,70)	(50,50)	(70,30)
FY07	(-70,10)	(-50,-10)	(-30,-30)	(-10,-50)	(10,-50)	(0,0)*	(50,70)	(70,50)
FY08	(-70,30)	(-50,10)	(-30,-10)	(-10,-30)	(10,-30)	(30,-70)	(0,0)*	(70,70)
FY09	(-70,50)	(-50,30)	(-30,10)	(-10,-10)	(10,-10)	(30,-50)	(50,-70)	(0,0)*

* (0,0) 代表不存在的情况，即被动接受信号无人机编号 = 发送信号信息无人机编号。

基于此种情况，出现位置偏差的无人机可以在接受信号后，首先记录下当前得到的 $< \alpha_1', \alpha_2' >$，基于该信息搜索并匹配已建立的标准角度序对表，选择**最小误差均值** (7) 对应的数据，即取最接近的标准角度序对，从而推测得到发射信号的无人机编号。

$$\mathbf{min}\{\ |\boldsymbol{\alpha_1} - \boldsymbol{\alpha_1'}| + |\boldsymbol{\alpha_2} - \boldsymbol{\alpha_2'}|\} \tag{7}$$

一次完整定位过程演绎如下:

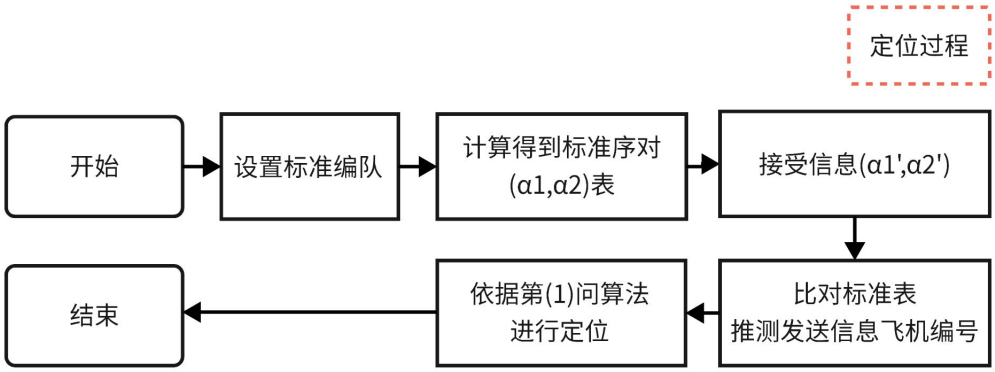

图 9　新增一架发送信号无人机的具体定位流程

需要预先说明的是，在发送具体位置信号前，我们已经计算得到标准角度序对表。同时为了更直观地展现角度序对的差值与比对过程，我们在此统一采用**角度制**表示角度信息。

首先，我们依据计算得到的偏差统计数据，随机生成符合正态分布的极角与极径偏差值:

$$\Delta\rho = 0.8196, \Delta\theta = 10.3756°$$

随机选择编号为 FY07 的无人机出现位置偏差、需要接受位置信号。当前的偏差位置为:

$$\rho = 100.8196, \theta = 250.3757°$$

随机选择另一架发送信号的无人机 **FY02**，从而计算得到无人机被动接收到的角度信息为:

$$\beta_1' = 35.0230°, \beta_2' = -15.12434°$$

此时开始搜索角度序对表，即逐一遍历接收信号无人机编号为 FY07 一列的数据，分别计算偏差值，遍历到的编号及对应偏差值如下:

表 6　发送信号无人机编号及对应偏差值

编号	角度偏差值
FY02	10.1473°
FY03	30.1473°
FY04	50.1473°
FY05	70.1473°
FY06	90.1473°
FY08	59.8986°
FY09	39.8986°

通过偏差值进行比对计算。实际上，我们也可以通过表 6 直观看出，编号 FY02 的无人机对应的角度偏差均值是最小的。由此推测：除 FY00,FY01 外，另一架发送给 FY07 方向信号的无人机编号为 **FY02**。对比此前我们已知的所选定的飞机编号，可知这**与推测出的编号一致**。

随后即可复用问题 1 的第 (1) 问所建立的定位模型，即传入 FY01 与推测编号 FY02，以及对应的角度信息，从而计算得到坐标位置:

$$\rho' = 100.8196, \theta' = 250.3757°$$

可见**与实际坐标位置一致**，从而依据题意，实现了完整的有效定位过程。

最后，为检验该模型是否能保证其准确率，我们使用蒙特卡罗模拟方法，随机生成大量数据，包括被动接受信号无人机的编号 T_x 和其位置偏差值，以及已知除发射机 T_0，T_1 之外的另一个发射信号无人机的编号 T_y，依次执行上述建立的定位过程模型，每次对比准确编号与推测编号，从而得到其推测准确率为 **94.5%** 左右。

在已建立模型的基础上，我们希望能够进一步**优化算法**，尽可能降低所产生的误差，故考虑误差出现的具体情况:

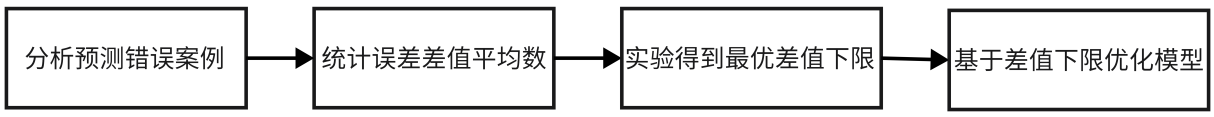

图 10　优化模型思路

当 FY09 编号无人机位置偏移至 $(\theta, \rho) = (342.43, 98.4862)$ 时，选择 FY00,FY01 与 FY04 编号无人机发射方向信号，其中包含角度对 $< \beta_1', \beta_2' >=< 84.0406°, -21.3850° >$，

按照上述定位模型进行定位，比对得到最小误差均值为 **22.6556°**，推测编号为 FY03。此时推测出现错误，实际发送信号的无人机编号为 FY04, 对应误差均值为 **25.4256°**。

可以发现的是，当前推测编号与正确编号所对应的误差均值差值是很小的。而统计 10,000 组测试数据中出现推测错误的 575 组数据，推测编号与正确编号**误差均值之差平均为 6.9389°**。

由此，我们提出**优化后的推测模型**：在比对标准角度序对表与实际角度信息后，可以计算得出最小误差均值与第二小误差均值，并取其差值，**和差值下限比较**判断是否需要再增加一架无人机。若需要增加，则再进行一次角度序对的搜索与匹配。将两次匹配的信息进行比较，取误差均值更大的一组作为推测数据，使用对应编号作为本次推测的结果。

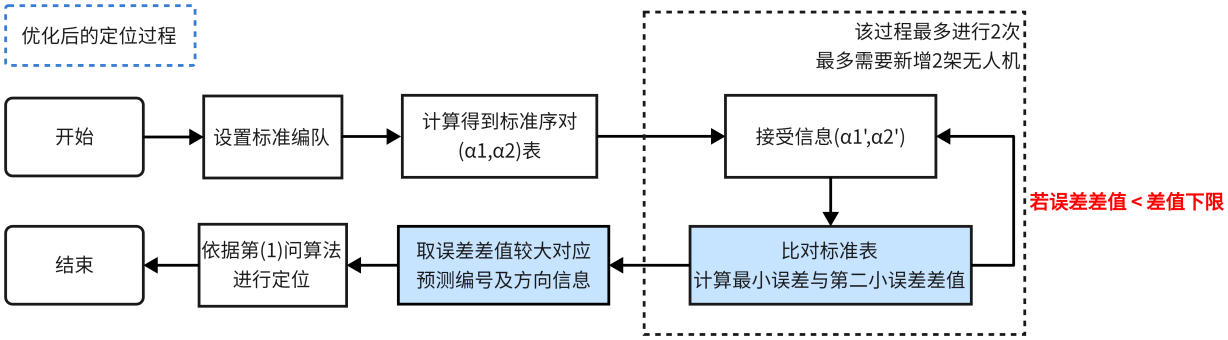

图 11　优化算法后的无人机具体定位流程

对于差值下限的选取，我们在将与均值相近的多个数据分别作为差值下限进行实验，并比对其正确率：当差值下限为 8，总错误率为 3.26%, 差值下限为 9，总错误率为 2.97%, 差值下限为 10，总错误率为 3.32%，此后错误率升高，且需要将下限设置过大会增加的计算时间考虑在内。

最终，我们设置角度差均值的最低限度为 9°。对优化后的模型进行蒙特卡罗模拟，测试 100,000 组数据后，可以得到**推测正确率提升至 96.9%**。还需要特别说明的是，此时平均所需无人机数量为 1.32。即多数情况下，其实仍然只需要增加一架无人机。

依据我们建立的模型，对问题 1.(2) 作出回答，考虑两种情况:

(1): 在需要**严格控制信号发射次数**时，如需要避免外界干扰，尽量实现电磁静默的条件下，除 FY00 和 FY01 之外，**只需增加一架**发射信号的无人机，此时使用优化前的模型 (图 9)，即可得到正确率较高的无人机编号;

(2): 当需要尽可能**保持定位的准确性**，可以适当多的发射电磁波信号时，除 FY00 和 FY01 之外，可能**需要增加最多两架**发射信号的无人机，此时使用优化后的模型 (图 11)，可以得到正确率更高的无人机编号。

最后，复用问题 1.(1) 建立的模型，均可实现无人机的有效定位。

5.2.3 结果分析

由于 FY02、FY03、…、FY09 的编号是均匀分布的，随机生成服从均匀分布的一定规模的编号数据，同时依据实际情况合理地生成服从正态分布的偏差值，使用蒙特卡罗方法对大量数据进行模拟，即使用生成的数据信息模拟定位过程，将所建立的定位模型投入实际应用中，从而评估模型的实施效果与准确性。

根据遍历，对比可以得到每一组数据所得推测编号的正确与否，从而计算得出，在使用蒙特卡罗方法的情况下，对比发射信号无人机的正确次数与测试次数 (如图 12)。

最终通过统计得到的正确次数，可以计算出准确率:

$$e = \frac{\text{正确次数}}{\text{测试次数}}$$

通过推测 100,000 次发射信号无人机编号数据得到，$e = 0.94479$。

由上述实验结果可知，准确率基本稳定在 **94.5%** 左右，即误差率控制在 6% 以内，当前数学模型具有一定的可靠性。故得出结论，假设成立，即还需要另外一**架**发射信号无人机就可以实现无人机的有效定位。

但不可否认的是，仍然存在会产生误差的情况。由此，我们考虑误差产生的实际情况，并在此基础上建立能够使正确率得到进一步优化的模型，即最多可能需要另外两架发射信号的无人机，实现准确率更高的定位。再次通过推测 100,000 次发射信号无人机编号数据得到结果，$e = 0.96785$。

由此可知，准确率**提升至 96.8%** 左右，该数据说明模型的优化取得了一定的效果，最终得到的模型具有更高的可靠性与准确性。

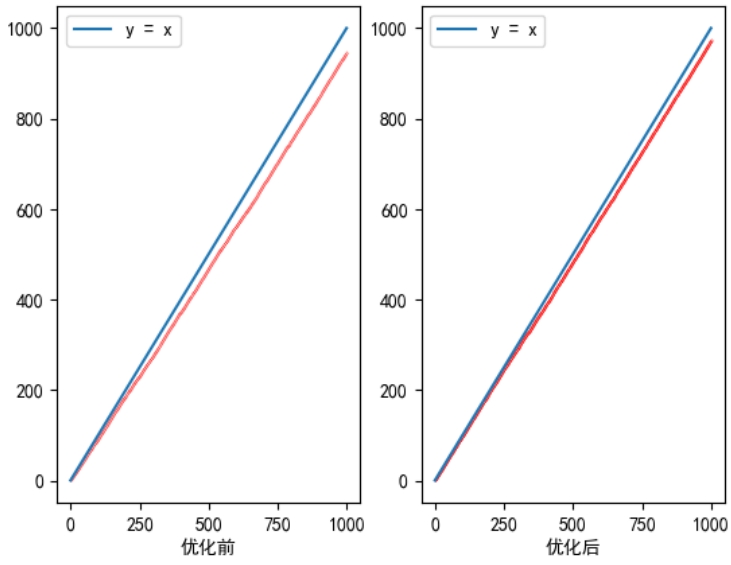

图 12　正确次数与测试次数比对

5.3 问题 1.(3) 模型的建立与求解

5.3.1 模型建立

由于无人机均保持同一高度飞行，可以将该题抽象为平面几何。建立以 FY00 为原点，FY00、FY01(理想位置) 连线为 x 轴，垂直 FY00、FY01(理想位置) 连线向上为 y 轴的直角坐标系。

由题目所给条件可知，初始时推断位置与实际位置存在一定差异，但差异较小，因此可设定初始条件:

$$\begin{cases} \text{推断半径}: R_0 = 100m, \\ \\ \text{FY01 方位角的推断值}: \theta_0 = 0rad. \end{cases} \tag{8}$$

在每一次选取发射信号的无人机调整位置后，更新推断半径与 FY01 方位角的推断值。对无人机的位置不断进行迭代调整，直至 9 架无人机均匀分布在某一个圆周上。

对以上部分名词实际意义的解释如下所示:

理想位置: 根据推断半径，无人机应当调整到的目标位置。

实际位置: 无人机所处的真实位置，这对于待调整位置的无人机是完全未知的。

推断位置: 根据无人机接收到的夹角信息进行定位后，所得到的位置坐标。

推断半径 R_0: 对圆周上所有无人机的推断位置到 FY00 的距离求均值得到的半径。

FY01 方位角的推断值 θ_0: 根据 FY01 的推断位置所计算出来的方位角。

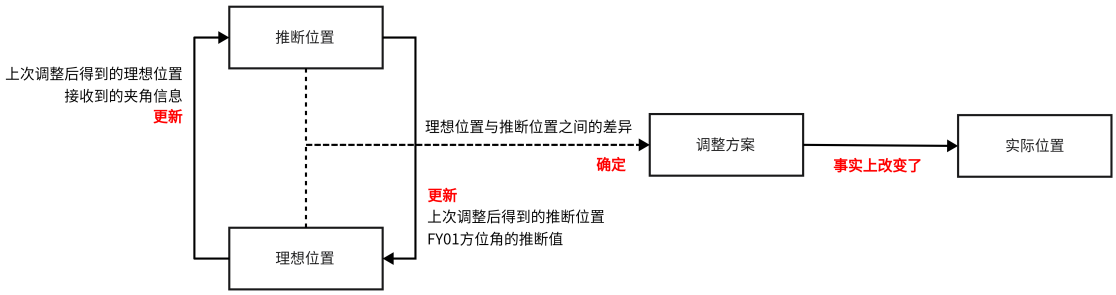

图 13　问题 1.(3) 位置调整流程

5.3.2 模型求解

对于每一轮迭代调整，从 FY01、FY02、…、FY09 中随机抽取 3 个作为发射信号的无人机，记为 S_1，S_2，S_3。对剩下的 6 架被动接收信号无人机进行位置调整，每次调整一架 (记为 C)。记 FY00 为 S_0。发射信号的无人机组合有如下 3 种情况:

a) $\boldsymbol{S_0}$、$\boldsymbol{S_1}$、$\boldsymbol{S_2}$

b) $\boldsymbol{S_0}$、$\boldsymbol{S_1}$、$\boldsymbol{S_3}$

c) S_0、S_2、S_3

对上述每一种组合，使用问题 1.(1) 建立的模型求解出 C 的坐标 (共有 3 组)，对这 3 组坐标求取平均值，可以得到 C 的推断位置 (记位矢为 $\overrightarrow{r_{c_1}}$)。根据 C 的标号、FY01 方位角的推断值 θ_0，可以求出 C 的理想位置 (记位矢为 $\overrightarrow{r_{c_1}}'$)。

$$\theta'_{C_1} = \frac{2\pi}{9}(T_c - 1) + \theta_0 \tag{9}$$

$$\overrightarrow{r_{c_1}}' = (R_0 cos\theta'_{C_1}, R_0 sin\theta'_{C_1}) \tag{10}$$

根据理想位矢与推断位矢之间的差异，可求得需要调整的位移:

$$\overrightarrow{\Delta r} = \overrightarrow{r_{c_1}}' - \overrightarrow{r_{c_1}} \tag{11}$$

更新推断半径 R，FY01 方位角的推断值 θ_0。**R 为 6 架被动接收信号无人机的推断半径平均值，θ_0 为调整位置后 FY01 的方位角**。通过反复**迭代调整**，可以使 9 架无人机与 FY00 之间距离的方差、与 FY00 连线夹角的方差逐渐收敛为 0，最终达到均匀分布在某一个圆周上的目的。

根据题目所给初始数据进行迭代模拟，得到部分调整过程如下所示 (如图 14)。

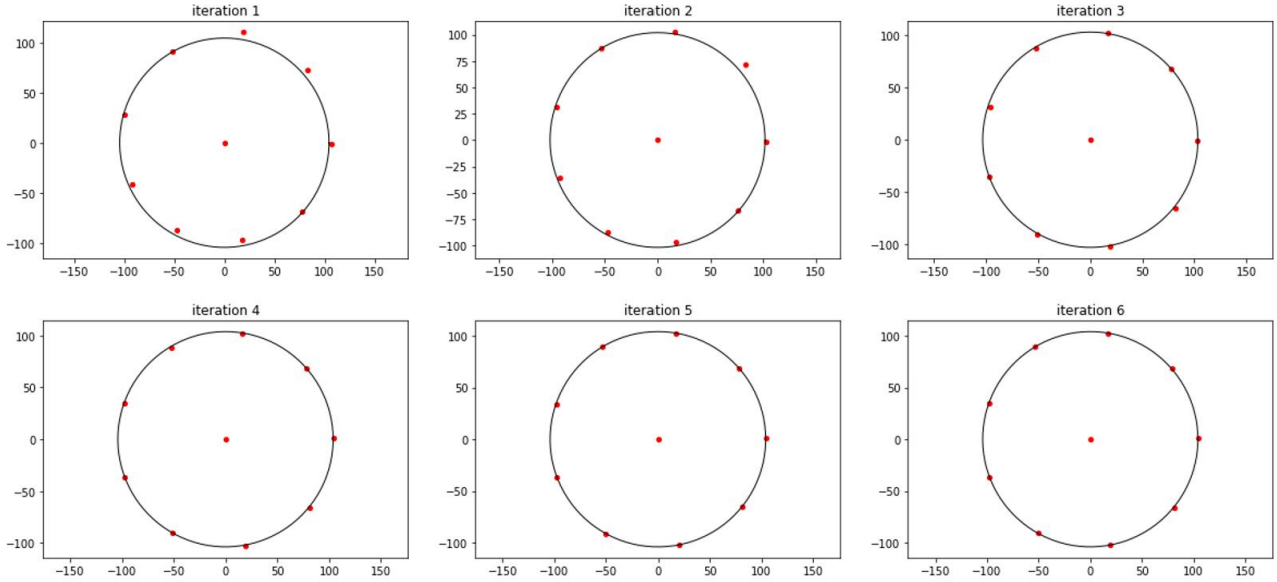

图 14　第 1~6 轮迭代调整的位置变化图

5.3.3 结果分析

由于推断坐标与实际坐标之间存在微小偏差，因此在每一轮迭代过程中，不能保证 9 架无人机绝对地均匀分布在同一圆周上。对 9 架无人机的夹角和半径进行误差分析。(部分数据见表 7)

表 7　每次调整无人机夹角和半径的方差

迭代轮数	相邻无人机夹角方差	无人机与圆心距离的方差
1	0.000326	2.368556
2	0.000305	2.321048
3	0.000544	2.597328
...	...	
49	1.036727e-19	1.971600e-08
50	7.388761e-20	2.028685e-08

因此可以根据所求方差得到角度和半径的方差随迭代次数增加的变化 (如图 15)，通过判断方差是否逐步收敛于 0，探究当前数学模型的可靠性。

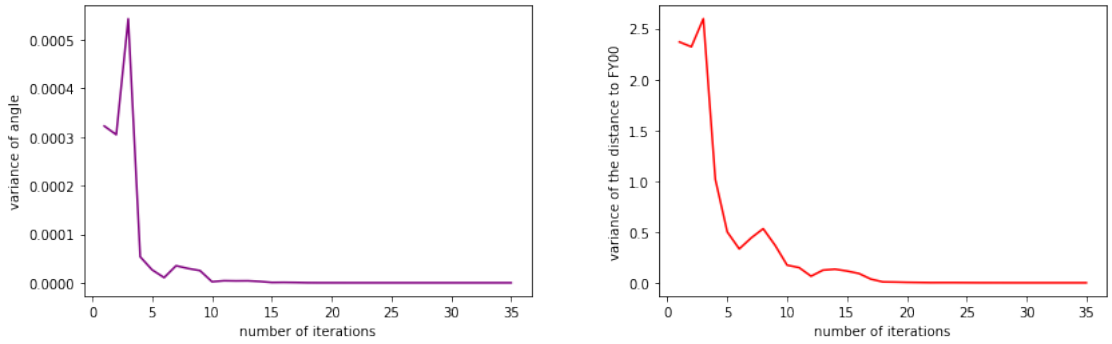

图 15　角度和半径的方差随迭代次数的变化曲线

由上述实验结果可知，角度的方差逐渐收敛至 10^{-20}，半径的方差逐步收敛至 10^{-8}。均趋近于 0。说明当前数学模型具有一定的可靠性。证明了调整方案的可行性。

5.4 问题 2 模型的建立与求解

5.4.1 模型建立

将无人机的锥形编队抽象到平面几何的问题上 (如图 16)。首先以 T_5、T_8、T_9 三个无人机为基准，将这三个无人机调整为正三角形。

因为这 3 架无人机的位置并不确定，所以随机规定 A、B、C 三个点，并作出其中一个角的角平分线 (如图 17)。

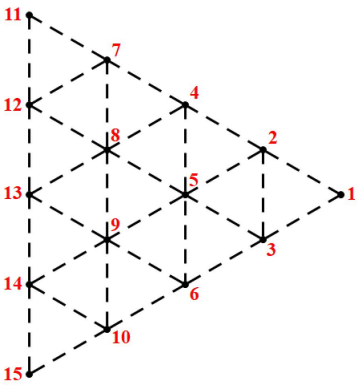

图 16　无人机锥形编队平面图

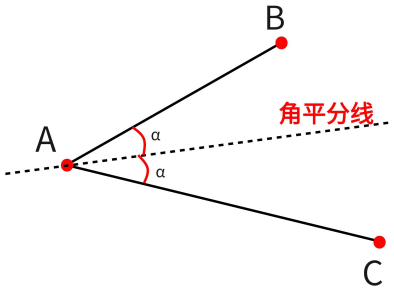

图 17　3 架无人机调整角度模型

5.4.2 模型求解

首先，A 作为接受信号无人机，B、C 作为发射信号无人机时，

a) 当 $\angle BAC < 60°$ 时，A 向 $\angle BAC$ 的角平分线方向调整。

b) 当 $\angle BAC > 60°$ 时，A 向 $\angle BAC$ 的角平分线反方向调整。

c) 当 $\angle BAC = 60°$ 时，则调整下一个无人机。

经过反复多次调整使得 A、B、C 构成一个正三角形。

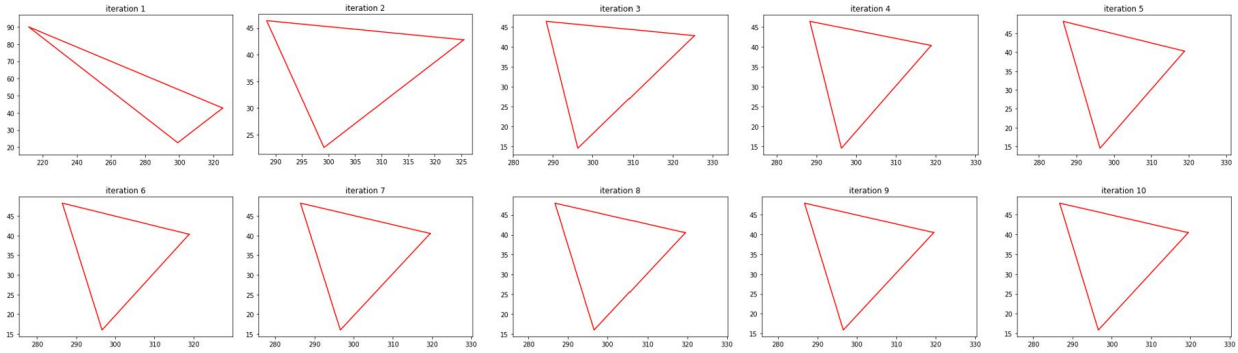

图 18　3 架无人机迭代形成正三角形的过程

随后，以此正三角形为基准，建立以线段 T_8、T_9 中点与 T_5 连线为 x 轴，T_8、T_9 连线为 y 轴的直角坐标系。由此可以得到 T_5、T_8、T_9 三者在该坐标系下的**实际位置**。

由于问题 1.(1) 可知，已知 3 架无人机的实际位置和夹角信息可以定位其它任意一架无人机，确定它的**实际位置**。同时考虑到编队队形的特殊性，可以根据正三角形的几何关系得到锥形图中其他的无人机的**理想位置**。

最后，使用与问题 1.(3) 相类似的方法，通过比对**理想位置**与**实际位置**之间的差异，确定调整方案。反复对位置略有偏差的无人机进行上述操作，可以完成无人机锥形编队队形的调整，消除飞行过程中可能产生的位置偏差。

5.4.3 结果分析

由于被动接受信号无人机的位置调节存在微小偏差，因此在形成正三角形的迭代过程中，不能保证 3 架无人机刚好形成正三角形。在迭代过程中，对 3 架无人机组成的三角形的边长 l_1、l_2、l_3 进行方差分析 (如图 19)。

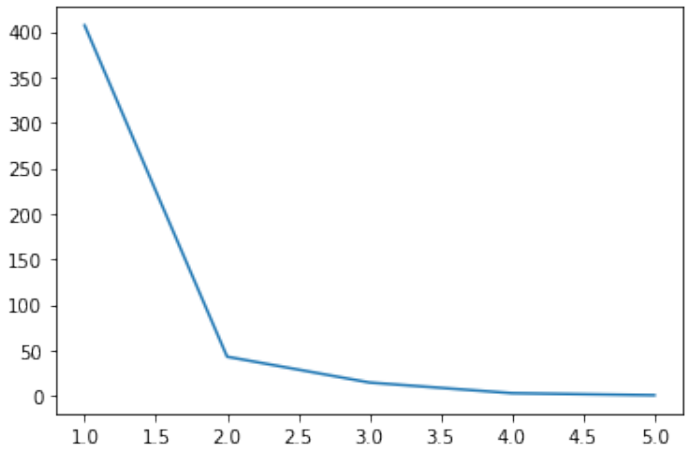

图 19　迭代过程中三角形三边长度方差的变化曲线

由图可知，随着迭代次数的增加，三角形三边长度的方差不断趋近于**0**，这说明所建立的调整方案具有合理性与可行性。

六、 模型的评价与改进

6.1 模型的优点

(1) 模型将一个实际三维问题简化为一个平面几何问题，有利于问题的理解与解决。
(2) 问题 1.(1) 所建立的模型与方法可以充分复用到其他的问题里，提高了模型的利用率和普适度。
(3) 问题 1.(2) 的模型运用搜索策略时，若只增加 1 架发射信号无人机，编号的准确率为 **94.5%**。当误差超过限度，则增加 2 架发射信号无人机，编号的准确率为 **96.8%**。有效解决减少无人机发射电磁波信号与提高定位准确性的冲突问题。
(4) 问题 1.(3) 的模型引入了**理想位置**、**实际位置**、**推断位置**等概念，有效地解决了无人机初始位置的微小偏差所带来的定位困难问题。同时，采用迭代调整方式，使距离偏差与角度偏差逐渐收敛为 0，模型较为稳定。

6.2 模型的不足之处

(1) 通过搜索策略所确定的编号，会有极小概率产生错误，从而导致定位结果存在误差。

(2) 问题 1.(3) 的迭代收敛只能使 9 架无人机无限逼近均匀分布在某一个圆周上的情况，不能保证该圆周可以满足某一特定的半径值要求。
(3) 问题 2 所给出的模型只适用于已知编号的情景，且对于立体的编队队形不适用。

6.3 模型的改进与推广方向

(1) 由于遍历求值不能很好地解决所求编号与理想值的冲突，所以我们可以考虑在增加 2 架无人机仍产生误差的情况下增加 3 架无人机定位，以进一步提高准确率。
(2) 由于迭代收敛确定调整方案只是逼近最优解，我们可以根据选取不同发射机所得到的准确性，为每一个无人机的推断位置设定不同的权重以计算得到理想位置，获得更加优化的解。
(3) 除无人机编队队形的调整以外，本文所提出的模型还可以应用到许多其它方面，如卫星、雷达、船舶的定位。本文所使用的搜索策略、迭代调整方法，以及基于蒙特卡洛模拟的模型评估手段，可以为相关领域的工作提供借鉴与参考。

七、 参考文献

[1] 林淑彬, 吴贵山, 姚文勇, 杨文元. 基于光照自适应动态一致性的无人机目标跟踪 [J/OL]. 智能系统学报,2022,09(18):1-12.

[2] 修建娟, 何友, 修建华. 多目标纯方位定位和跟踪 [J]. 现代雷达,2004(08):45-48.

[3] 张亚飞. 蒙特卡罗模拟及其基本应用 [J]. 电脑知识与技术,2013,9(06):1435-1437.

八、 附录

附录一: 支持材料文件列表

```
problem_1_1_ans.py
problem_1_1_monteCarlo.py
problem_1_2_calculate_error.py
problem_1_2_optimization_model.py
problem_1_2_original_model.py
problem_1_3.py
problem_2.py
问题1.(1)发射与接收信号的方向信息.xlsx
问题1.(1)计算结果的坐标误差率.xlsx
问题1.(1)接收信号无人机的位置信息.xlsx
问题1.(2)错误推测编号情况.txt
问题1.(3)每次调整后各无人机到FY00实际距离的方差.txt
问题1.(3)每次调整后无人机两两之间实际夹角的方差.txt
问题2对某一随机生成三角形进行调整时的方差变化情况.txt
```

附录二: 代码

题目 1.(1) 解被动接收信号无人机位置

```
import sympy as sp
import numpy as np
from sympy import *
from numpy import *
init_printing(use_unicode=True)

x,y = symbols('x y')
tan_talpha1, tan_talpha2 = symbols('tan_talpha1 tan_talpha2')
k_ac, k_oc, k_bc = symbols('k_ac k_oc k_bc')
x_a, y_a = symbols('x_a y_a')
x_b, y_b = symbols('x_b y_b')
R = Symbol('R')

f1 = k_ac - (y-y_a)/(x-x_a)
f2 = k_bc - (y-y_b)/(x-x_b)
f3 = k_oc - y/x
f4 = tan_talpha1 - (k_oc-k_ac)/(1+k_oc*k_ac)
f5 = tan_talpha2 - (k_bc-k_oc)/(1+k_bc*k_oc)

ans = sp.solve([f1,f2,f3,f4,f5],[x,y,k_ac,k_oc,k_bc])
print(ans)
print(sp.solve([f1,f2],[x,y]))
```

题目 1.(1) 蒙特卡罗算法关键代码

```

theta_k = np.arange(0,2*math.pi, 2*math.pi/9) # FY01-09无偏差坐标极角度
# 给定半径
R = 100
rho_k = np.zeros(9)
rho_k.fill(R)         # FY01-FY09极径
index_k = np.arange(0,9) # 设置FY0K下标


def product_random_difference():
    # 参照表中所给数据
    real_location_theta = [0, 40.10, 80.21, 119.75, 159.86, 199.96, 240.07, 280.17, 320.28]
    real_location_rho = [100, 98, 112, 105, 98, 112, 105, 98, 112]
    # 理想极坐标
    ideal_location_theta = arange(0,360,360/9)
    ideal_location_rho = np.zeros(9)
    ideal_location_rho.fill(100)

    mean_dif_theta = np.mean(real_location_theta-ideal_location_theta)
    std_dif_theta = np.std(real_location_theta-ideal_location_theta)

    mean_dif_rho = np.mean(real_location_rho-ideal_location_rho)
    std_dif_rho  = np.std(real_location_rho-ideal_location_rho)

    dif_theta = np.random.normal(loc=mean_dif_theta,scale=std_dif_theta)
    dif_rho  = np.random.normal(loc=mean_dif_rho, scale=std_dif_rho)
    print("随机偏差值: ",dif_rho,dif_theta)
    return [dif_rho,dif_theta]

def monte_carlo(dif_rho,dif_theta):
# 使FY0X略有偏差
# 依据题目给定数据参考偏差范围(最好给定正态分布随机数)
# 实现正态分布随机数
    index_x = random.choice(index_k) # 随机选择FY0X编号
    theta_x = theta_k[index_x] + dif_theta # 产生误差
    rho_x  = rho_k [index_x] + dif_rho
```

```
37
38      print("出现位置偏差的飞机: FY0",index_x+1,":",theta_x,rho_x)
39
40      while True:
41          index_a = random.choice(index_k)
42          if index_a != index_x:
43              break                       # 随机选择FY0A
44      while True:
45          index_b = random.choice(index_k)
46          if index_b != index_x:
47              if index_b != index_a:
48                  break                   # 随机选择FY0B
49      print("除FY00外发送信号的飞机:FY0",index_a+1,"FY0",index_b+1)
50
51      # 求出实际的tan_alpha1 & tan_alpha2
52      x_a = R*cos(theta_k[index_a])
53      y_a = R*sin(theta_k[index_a])
54      x_b = R*cos(theta_k[index_b])
55      y_b = R*sin(theta_k[index_b])
56      x_c = rho_x*cos(theta_x)
57      y_c = rho_x*sin(theta_x)
58
59      true_ac = (y_c-y_a) / (x_c-x_a)
60      true_bc = (y_c-y_b) / (x_c-x_b)
61      true_oc = y_c/x_c
62      talpha1 = atan((true_oc-true_ac)/(1+true_oc*true_ac))
63      talpha2 = atan((true_bc-true_oc)/(1+true_bc*true_oc))
64
65      x,y             = symbols('x y')
66      rho_c,theta_c   = symbols('rho_c theta_c')
67      k_ac, k_oc, k_bc = symbols('k_ac k_oc k_bc')
68
69      # 还原FY0X定位过程
70      # 直角坐标系解
71      f1 = k_ac - (y-y_a)/(x-x_a)
72      f2 = k_bc - (y-y_b)/(x-x_b)
73      f3 = k_oc - y/x
74      f4 = tan(talpha1) - (k_oc-k_ac)/(1+k_oc*k_ac)
75      f5 = tan(talpha2) - (k_bc-k_oc)/(1+k_bc*k_oc)
76
77      print("偏差飞机实际位置:",x_c,y_c)
78      ans=sp.solve([f1,f2,f3,f4,f5],[x,y,k_ac,k_bc,k_oc])
79      print("偏差飞机定位位置:",ans[0][0],ans[0][1])
80      error_x = (ans[0][0]-x_c)/x_c
81      error_y = (ans[0][1]-y_c)/y_c
82      return [error_x,error_y]
83
84  # 按照正态分布产生误差
85  error_list = []
86  size=1001 #随机数数量
87  for i in range(size):
88      dif = product_random_difference()
89      # 使用蒙特卡洛模拟算法计算误差率
90      error_list.append(monte_carlo(dif[0],dif[1]))
91
92  # 绘制阵列图
93  plt.show()
94
95  # 计算
96  error_x = []
97  error_y = []
98  for element in error_list:
99      error_x.append(element[0])
100     error_y.append(element[1])
```

```

ax1 = plt.subplot(211)
ax1.scatter(np.arange(0,size,1), error_x, color='red',s=1)
ax1.set_ylim(bottom=-1e-13,top=1e-13)
ax2 = plt.subplot(212)
ax2.scatter(np.arange(0,size,1), error_y, color='blue',s=1)
ax2.set_ylim(bottom=-1e-13,top=1e-13)
plt.show()
mean_err_x = np.mean(error_x)
mean_err_y = np.mean(error_y)
print("Xc误差率均值:",mean_err_x,"Yc误差率均值",mean_err_y)
```

题目 1.(2) 遍历推测算法

```

# 令除FY00之外的飞机为A,B, 出现位置偏差的飞机为C。
#  ACB范围[-140,-120,-100,-80,-60,-40,-20,20,40,60,80,100,120,140]
# 计算 ACO, BCO的对应关系(alpha_1,alpha_2)
# 需要注意的是, 当C点为 的顶点(方向向左), 且CO在 BCA中间, BC在上, AC在下, 此时alpha_1,alpha_2为正值

# 得到标准位置处的(alpha1,alpha2)序对
def set_alpha(num_a, num_b, num_c):
    index_a = num_a-1
    index_b = num_b-1
    index_c = num_c-1

    theta_k = np.arange(0,2*math.pi, 2*math.pi/9) # FY01-09无偏差坐标极角度
    R = 100               # 给定半径,假设为100
    rho_k = np.zeros(9)
    rho_k.fill(R)         # FY01-FY09极径

    # 求出实际的tan_alpha1 & tan_alpha2
    x_a = R*cos(theta_k[index_a])
    y_a = R*sin(theta_k[index_a])
    x_b = R*cos(theta_k[index_b])
    y_b = R*sin(theta_k[index_b])
    x_c = R*cos(theta_k[index_c])
    y_c = R*sin(theta_k[index_c])

    true_ac = (y_c-y_a) / (x_c-x_a)
    true_bc = (y_c-y_b) / (x_c-x_b)
    true_oc = y_c/x_c

    talpha1 = atan((true_oc-true_ac)/(1+true_oc*true_ac))
    talpha2 = atan((true_bc-true_oc)/(1+true_bc*true_oc))

    return [talpha1,talpha2]

# 遍历得到(alpha1,alpha2)的二位数据表,且确定其中一架飞机为FY01
def get_alpha_diagram():
    diagram=[[0 for i in range(8)] for i in range(8)]
    for i in range(2,10):
        for j in range(2,10):
            if i == j :
                diagram[i-2][j-2]=[0,0]
                continue
            diagram[i-2][j-2]=set_alpha(1,i,j)
    return diagram

# 计算实际alpha与标准alpha差异值
def get_dif(idea,true):
    if idea>true :
        return idea-true
    else:
```

```
        return true-idea

# 当给定编号的偏差飞机得到(alpha1,alpha2)弧度制信息时，进行信息比对，取误差最小的编号序列即可
def find_b(alpha1,alpha2,num_c):
    num_b=0
    min_dif=100.0 # 最小误差，初始值设为最大值
    alpha_diagram=get_alpha_diagram()
    for i in range(0,8):
        if i==num_c-2:
            continue
        ideal_alpha1 = alpha_diagram[i][num_c-2][0]
        ideal_alpha2 = alpha_diagram[i][num_c-2][1]
        cur_dif = get_dif(ideal_alpha1,alpha1)+get_dif(ideal_alpha2,alpha2)
        print("FY0",i+2,":",math.degrees(cur_dif))
        if cur_dif < min_dif:
            min_dif=cur_dif
            num_b=i+2

    return num_b

# Monte Carlo验证
# 按照正态分布产生偏差值
def product_random_difference():
    # 参照表中所给数据
    real_location_theta = [0, 40.10, 80.21, 119.75, 159.86, 199.96, 240.07, 280.17, 320.28]
    real_location_rho = [100, 98, 112, 105, 98, 112, 105, 98, 112]
    # 理想极坐标
    ideal_location_theta = arange(0,360,360/9)
    ideal_location_rho = np.zeros(9)
    ideal_location_rho.fill(100)

    mean_dif_theta = np.mean(real_location_theta-ideal_location_theta)
    std_dif_theta = np.std(real_location_theta-ideal_location_theta)

    mean_dif_rho = np.mean(real_location_rho-ideal_location_rho)
    std_dif_rho  = np.std(real_location_rho-ideal_location_rho)

    dif_theta = np.random.normal(loc=mean_dif_theta,scale=std_dif_theta)
    dif_rho  = np.random.normal(loc=mean_dif_rho, scale=std_dif_rho)
    print("随机偏差值: ",dif_rho,math.degrees(dif_theta))
    # return [0,0]
    return [dif_rho,dif_theta]

# 使用Monte_carlo产生弧度信息与发送信号的飞机编号
# 与问题1(1)基本一致
def monte_carlo(dif_rho,dif_theta):
# 使FY0X略有偏差
# 依据题目给定数据参考偏差范围(最好给定正态分布随机数)
# 实现正态分布随机数
    theta_k = np.arange(0,2*math.pi, 2*math.pi/9) # FY01-09无偏差坐标极角度
    # 给定半径
    R = 100
    rho_k = np.zeros(9)
    rho_k.fill(R)         # FY01-FY09极径
    index_k = np.arange(0,9) # 设置FY0K下标
    index_x = random.choice(np.arange(1,9)) # 随机选择FY0X编号
    theta_x = theta_k[index_x] + dif_theta # 产生误差
    rho_x  = rho_k [index_x] + dif_rho

    print("出现位置偏差的飞机: FY0",index_x+1,"当前位置:",math.degrees(theta_x),rho_x)
    error_c=index_x+1

    index_a=0                          #固定A飞机为FY01
    while True:
```

```
        index_b = random.choice(index_k)
        if index_b != index_x:
            if index_b != index_a:
                break                    # 随机选择FY0B
    launch_b=index_b+1

    # 求出实际的tan_alpha1 & tan_alpha2
    x_a = R*cos(theta_k[index_a])
    y_a = R*sin(theta_k[index_a])
    x_b = R*cos(theta_k[index_b])
    y_b = R*sin(theta_k[index_b])
    x_c = rho_x*cos(theta_x)
    y_c = rho_x*sin(theta_x)

    true_ac = (y_c-y_a) / (x_c-x_a)
    true_bc = (y_c-y_b) / (x_c-x_b)
    true_oc = y_c/x_c

    talpha1 = atan((true_oc-true_ac)/(1+true_oc*true_ac))
    talpha2 = atan((true_bc-true_oc)/(1+true_bc*true_oc))

    return [talpha1, talpha2, error_c, launch_b]

size=100
correct=0 #
for i in range(size):
    dif = product_random_difference()
    # 使用蒙特卡洛模拟算法计算误差率
    info=(monte_carlo(dif[0],dif[1])) # 获取弧度信息与既定编号:info[alpha1,alpha2,num_c,num_b]

    num_b=find_b(info[0],info[1],info[2])
    print("推测编号:",num_b,"实际编号",info[3])
    if num_b==info[3]:
        correct+=1
percent=correct/size
print("推测编号正确率: ", percent)
```

题目 1.(2) 优化后的编号推测代码

```
# 输入待定位无人机编号num_c与角度信息(alpha1,alpha2)(弧度制)
# 返回推测编号过程中最小误差均值与第二小误差均值之差
def get_subdif(alpha1,alpha2,num_c):
    # print(math.degrees(alpha1),math.degrees(alpha2))
    print("alpha1:",math.degrees(alpha1),"alpha2:",math.degrees(alpha2))
    num_b=0
    min_dif=100.0 # 最小误差, 初始值设为最大值
    second_dif=100.0 #记录第二小的误差
    alpha_diagram=get_alpha_diagram()
    for i in range(0,8):
        if i==num_c-2:
            continue
        ideal_alpha1 = alpha_diagram[i][num_c-2][0]
        ideal_alpha2 = alpha_diagram[i][num_c-2][1]
        cur_dif = get_dif(ideal_alpha1,alpha1)+get_dif(ideal_alpha2,alpha2)
        print("FY0",i+2,":",math.degrees(cur_dif))
        if cur_dif < min_dif:
            second_dif=min_dif
            min_dif=cur_dif
            num_b=i+2
        else :
            if cur_dif < second_dif:
                second_dif=cur_dif

    print("最小差值: ",math.degrees(min_dif),"第二小差值: ",math.degrees(second_dif))
```

```
    print("差值之差",math.degrees(get_dif(min_dif,second_dif)))

    return [num_b,get_dif(min_dif,second_dif)]

# 优化后的推测编号过程的蒙特卡罗测试
# 输入推测的次数
def find_process(times):
    error=0 # 错误推测次数
    dif_limit=math.radians(9) # 设置的差值下限
    for i in range(times):
        flag=np.zeros(9) # 发射无人机的选择标记
        flag[0]=1      # 标记FY01
        dif = product_random_difference() # 与问题1.1相同的方式产生位置误差
        info_c=choose_c(dif[0],dif[1]) # 随机选择待定位的无人机添加误差
        flag[info_c[0]-1]=1           # 标记接收机
        num_b=choose_b(flag)          # 依据标记选择无人机发送信号
        alpha_pair=cal_true_alpha(info_c[0],info_c[1],info_c[2],num_b)

        predict_b=find_b(alpha_pair[0],alpha_pair[1],info_c[0])
        if predict_b[1] < dif_limit:   # 差值小于差值下限
            flag[num_b-1]=1            # 标记当前选择的无人机编号
            num_b_copy=choose_b(flag)  # 再次选择一架无人机
            alpha_pair_copy=cal_true_alpha(info_c[0],info_c[1],info_c[2],num_b_copy)
            predict_copy=find_b(alpha_pair_copy[0],alpha_pair_copy[1],info_c[0])
            if predict_b[1] < predict_copy[1] : # 选择差值较大的推测组
                predict_b=predict_copy
                num_b=num_b_copy
        if predict_b[0]!=num_b :
            error+=1

    print("总错误率: ",error/times)
    print("正确率: ",1-error/times)
    return
```

题目 1.(3) 模拟位置调整代码

```
import math
from random import sample

import numpy as np
import pandas
from matplotlib import pyplot as plt
from sympy import symbols, solve

# 记录历次迭代的实际误差
radius_variance_history = []
angle_variance_history = []

# 初始化推断半径及推断角度
radius_inferred = 100
theta_1_inferred = 0

# 初始化实际位置与理想位置
real_locations_polar = [(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)]
correct_locations_polar = [(0, 0), (100, 0), (100, 40), (100, 80), (100, 120), (100, 160),
    (100, 200),
                           (100, 240),
                           (100, 280), (100, 320)]
real_locations_cartesian = []
correct_locations_cartesian = []
for location in real_locations_polar:
```

```
    x = location[0] * math.cos(location[1] / 180 * math.pi)
    y = location[0] * math.sin(location[1] / 180 * math.pi)
    real_locations_cartesian.append([x, y])
for location in correct_locations_polar:
    x = location[0] * math.cos(location[1] / 180 * math.pi)
    y = location[0] * math.sin(location[1] / 180 * math.pi)
    correct_locations_cartesian.append([x, y])

# 实际位置
real_locations_df = pandas.DataFrame(real_locations_cartesian, columns=['x', 'y'])
# 理想位置
correct_locations_df = pandas.DataFrame(correct_locations_cartesian, columns=['x', 'y'])
# 推断位置
inferred_locations_df = pandas.DataFrame(correct_locations_cartesian, columns=['x', 'y'])


# 根据最新的推断半径和推断角度更新理想位置
def update():
    for i in range(1, 10):
        correct_locations_df.iloc[i, 0] = radius_inferred * np.cos((i - 1) / 9 * 2 * math.pi +
            theta_1_inferred)
        correct_locations_df.iloc[i, 1] = radius_inferred * np.sin((i - 1) / 9 * 2 * math.pi +
            theta_1_inferred)


# 求解基本模型
# id_a, id_b, id_c分别为A, B, C三点的编号
def solve_basic_model(id_a, id_b, id_c):
    # 根据实际位置求解tan_alpha1与tan_alpha2, 作为观测到的已知输入
    k_oc_real = real_locations_df.iloc[id_c, 1] / real_locations_df.iloc[id_c, 0]
    k_ac_real = (real_locations_df.iloc[id_c, 1] - real_locations_df.iloc[id_a, 1]) / (
            real_locations_df.iloc[id_c, 0] - real_locations_df.iloc[id_a, 0])
    k_bc_real = (real_locations_df.iloc[id_c, 1] - real_locations_df.iloc[id_b, 1]) / (
            real_locations_df.iloc[id_c, 0] - real_locations_df.iloc[id_b, 0])
    tan_alpha1 = (k_oc_real - k_ac_real) / (1 + k_oc_real * k_ac_real)
    tan_alpha2 = (k_bc_real - k_oc_real) / (1 + k_bc_real * k_oc_real)

    x_c, y_c = symbols("x_c y_c")
    k_ac, k_oc, k_bc = symbols("k_ac k_oc k_bc")
    f1 = k_ac - (y_c - correct_locations_df.iloc[id_a, 1]) / (x_c -
        correct_locations_df.iloc[id_a, 0])
    f2 = k_oc - y_c / x_c
    f3 = k_bc - (y_c - correct_locations_df.iloc[id_b, 1]) / (x_c -
        correct_locations_df.iloc[id_b, 0])
    f4 = tan_alpha1 - (k_oc - k_ac) / (1 + k_oc * k_ac)
    f5 = tan_alpha2 - (k_bc - k_oc) / (1 + k_bc * k_oc)
    ans = solve([f1, f2, f3, f4, f5], [x_c, y_c, k_ac, k_oc, k_bc])
    return ans[0][0], ans[0][1]


# 求角度的偏差
# 仅用于误差估算, 不可用于调整方案
def calculate_real_angle_variance():
    theta_list = []
    for i in range(1, 10):
        if i == 9:
            j = 1
        else:
            j = i + 1
        x_i = real_locations_df.iloc[i, 0]
        y_i = real_locations_df.iloc[i, 1]
        x_j = real_locations_df.iloc[j, 0]
        y_j = real_locations_df.iloc[j, 1]
        tan_i = y_i / x_i
```

```
        tan_j = y_j / x_j
        tan_theta = (tan_j - tan_i) / (1 + tan_j * tan_i)
        theta_list.append(math.atan(tan_theta))
    print("theta list:", theta_list)
    return np.var(np.array(theta_list))


# 求与圆心真实距离的均值及方差
# 仅用于误差估算，不可用于调整方案
def calculate_real_average_variance():
    x = np.array(real_locations_df.iloc[1:, 0], dtype=float)
    y = np.array(real_locations_df.iloc[1:, 1], dtype=float)
    distance = np.sqrt(x ** 2 + y ** 2)
    return np.average(distance), np.std(distance)


# 用于计算推断半径及angle
def calculate_inferred_radius_angle():
    x = np.array(inferred_locations_df.iloc[1:, 0], dtype=float)
    y = np.array(inferred_locations_df.iloc[1:, 1], dtype=float)
    distance = np.sqrt(x ** 2 + y ** 2)
    return np.average(distance), math.atan(inferred_locations_df.iloc[1, 1] /
        inferred_locations_df.iloc[1, 0])


iteration_times = 50
for iteration in range(iteration_times):

    total_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

    # 发射信号的无人机编号集合
    launch_list = sample(total_list, 3)

    # 接收信号的无人机编号集合
    for element in launch_list:
        total_list.remove(element)
    receive_list = total_list

    # 对接收信号的无人机进行位置调整
    for receiver in receive_list:
        x_inferred_1, y_inferred_1 = solve_basic_model(launch_list[0], launch_list[1], receiver)
        x_inferred_2, y_inferred_2 = solve_basic_model(launch_list[0], launch_list[2], receiver)
        x_inferred_3, y_inferred_3 = solve_basic_model(launch_list[1], launch_list[2], receiver)
        x_inferred = (x_inferred_1 + x_inferred_2 + x_inferred_3) / 3
        y_inferred = (y_inferred_1 + y_inferred_2 + y_inferred_3) / 3
        x_adjusted = real_locations_df.iloc[receiver, 0] + correct_locations_df.iloc[receiver,
            0] - x_inferred
        y_adjusted = real_locations_df.iloc[receiver, 1] + correct_locations_df.iloc[receiver,
            1] - y_inferred

        real_locations_df.iloc[receiver, 0] = x_adjusted
        real_locations_df.iloc[receiver, 1] = y_adjusted
        inferred_locations_df.iloc[receiver, 0] = x_inferred
        inferred_locations_df.iloc[receiver, 1] = y_inferred

    plt.scatter(real_locations_df.iloc[:, 0], real_locations_df.iloc[:, 1], color="red", s=17)

    # 计算与原点距离的平均值(即实际半径)和方差
    radius_real, radius_variance_real = calculate_real_average_variance()
    # 计算两两飞行器夹角的方差
    angle_variance = calculate_real_angle_variance()
    # 将上述两种方差添加到历史记录表中，便于之后作图
    radius_variance_history.append(radius_variance_real)
    angle_variance_history.append(angle_variance)
```

```
148
149     theta = np.linspace(0, 2 * np.pi, 200)
150     x_real = radius_real * np.cos(theta)
151     y_real = radius_real * np.sin(theta)
152     plt.plot(x_real, y_real, color="black", linewidth=1)
153     #     x_inferred = radius_inferred * np.cos(theta)
154     #     y_inferred = radius_inferred * np.sin(theta)
155     #     plt.plot(x_inferred, y_inferred, color= "yellow")
156     plt.axis("equal")
157
158     # 更新推断半径和角度
159     radius_inferred, theta_1_inferred = calculate_inferred_radius_angle()
160
161     print("radius_real, radius_variance_real: ", radius_real, radius_variance_real)
162     print("radius_inferred: ", radius_inferred)
163     print("theta_1_inferred: ", theta_1_inferred)
164     print("angle_variance: ", angle_variance)
165     plt.title("iteration %d" % (iteration + 1))
166     plt.show()
167     update()
168     print(correct_locations_df)
169     print("iteration %d finished\n" % (iteration + 1))
170
171 plt.plot(range(1, (iteration_times + 1)), radius_variance_history)
172 plt.plot(range(1, (iteration_times + 1)), angle_variance_history)
173 plt.show()
174
175 plt.plot(range(1, (35 + 1)), radius_variance_history[0:35], color="red")
176 plt.xlabel("number of iterations")
177 plt.ylabel("variance of the distance to FY00")
178 plt.figure(figsize=(24, 32))
179 plt.show()
180
181 plt.plot(range(1, (35 + 1)), angle_variance_history[0:35], color="purple")
182 plt.xlabel("number of iterations")
183 plt.ylabel("variance of angle")
184 plt.figure(figsize=(24, 32))
185 plt.show()
```

题目 2 模拟正三角形的位置调整代码

```
1  import math
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import pandas as pd
6
7  d = 50
8  dif_theta = 10
9  dif_rho = 0.2
10
11
12 def calculate_side_length(a, b):
13     delta_x = real_locations_df.iloc[a - 1, 0] - real_locations_df.iloc[b - 1, 0]
14     delta_y = real_locations_df.iloc[a - 1, 1] - real_locations_df.iloc[b - 1, 1]
15     return math.sqrt(delta_x ** 2 + delta_y ** 2)
16
17
18 # 计算相邻两点之间距离的方差(所有点)
19 def calculate_variance():
20     side_length_list = [calculate_side_length(1, 2), calculate_side_length(1, 3),
           calculate_side_length(2, 3),
21                         calculate_side_length(2, 4), calculate_side_length(2, 5),
                               calculate_side_length(3, 5),
```

```
                    calculate_side_length(3, 6), calculate_side_length(4, 5),
                        calculate_side_length(5, 6),
                    calculate_side_length(4, 7), calculate_side_length(4, 8),
                        calculate_side_length(5, 8),
                    calculate_side_length(5, 9), calculate_side_length(6, 9),
                        calculate_side_length(6, 10),
                    calculate_side_length(7, 8), calculate_side_length(8, 9),
                        calculate_side_length(9, 10),
                    calculate_side_length(7, 11), calculate_side_length(7, 12),
                        calculate_side_length(8, 12),
                    calculate_side_length(8, 13), calculate_side_length(9, 13),
                        calculate_side_length(9, 14),
                    calculate_side_length(10, 14), calculate_side_length(10, 15),
                        calculate_side_length(11, 12),
                    calculate_side_length(12, 13), calculate_side_length(13, 14),
                        calculate_side_length(14, 15)]
    return np.var(side_length_list)


# 计算相邻两点之间距离的方差(中间9个点)
def calculate_variance_1():
    side_length_list = [calculate_side_length(5, 8), calculate_side_length(8, 9),
        calculate_side_length(9, 5),
                    calculate_side_length(8, 4), calculate_side_length(4, 5),
                        calculate_side_length(13, 8),
                    calculate_side_length(13, 9), calculate_side_length(6, 9),
                        calculate_side_length(6, 5)]
    return np.var(side_length_list)


# 计算相邻两点之间距离的方差(中间3个点)
def calculate_variance_2():
    side_length_list = [calculate_side_length(5, 8), calculate_side_length(8, 9),
        calculate_side_length(9, 5)]
    return np.var(side_length_list)


# 求解从vector1旋转到vector2的夹角(逆时针为正)
# vector1: (x1, y1)
# vector2: (x2, y2)
def calculate_vector_angle(vector1, vector2):
    v1 = np.array(vector1, dtype=np.float64)
    v2 = np.array(vector2, dtype=np.float64)
    norm_product = np.linalg.norm(v1) * np.linalg.norm(v2)
    # 叉乘求解rho
    rho = np.arcsin(np.cross(v1, v2) / norm_product)
    # 点乘求解theta
    theta = np.arccos(np.dot(v1, v2) / norm_product)
    if rho < 0:
        return - theta
    else:
        return theta


correct_locations = [[2 * math.sqrt(3) * d, 0],
                     [3 / 2 * math.sqrt(3) * d, d / 2],
                     [3 / 2 * math.sqrt(3) * d, -d / 2],
                     [math.sqrt(3) * d, d],
                     [math.sqrt(3) * d, 0],
                     [math.sqrt(3) * d, -d],
                     [1 / 2 * math.sqrt(3) * d, 3 / 2 * d],
                     [1 / 2 * math.sqrt(3) * d, d / 2],
                     [1 / 2 * math.sqrt(3) * d, - d / 2],
                     [1 / 2 * math.sqrt(3) * d, - 3 / 2 * d],
```

```
                  [0, 2 * d],
                  [0, d],
                  [0, 0],
                  [0, -d],
                  [0, -2 * d]
                  ]

correct_locations[4] = [100, 0]
correct_locations[7] = [-80, 20]
correct_locations[8] = [0, 0]

# 初始化理想位置
correct_locations_df = pd.DataFrame(correct_locations)

# 初始化实际位置
real_locations = []
for location in correct_locations:
    real_locations.append([location[0] + dif_rho * math.cos(dif_theta), location[1] + dif_rho *
        math.sin(dif_theta)])

real_locations_df = pd.DataFrame(real_locations)


# id_a, id_b, id_c
# id_a为待调整的无人机
# 调整夹角为某一特定值(默认60°)
def adjust_angle_to_certain_degree(id_a, id_b, id_c, degree=60):
    id_a = id_a - 1
    id_b = id_b - 1
    id_c = id_c - 1

    vector_ab_real = np.array([real_locations_df.iloc[id_b, 0] - real_locations_df.iloc[id_a,
        0],
                               real_locations_df.iloc[id_b, 1] - real_locations_df.iloc[id_a, 1]])
    vector_ac_real = np.array([real_locations_df.iloc[id_c, 0] - real_locations_df.iloc[id_a,
        0],
                               real_locations_df.iloc[id_c, 1] - real_locations_df.iloc[id_a, 1]])
    vector_bisector_real = (vector_ab_real + vector_ac_real) / 2
    vector_bisector_real = vector_bisector_real / np.linalg.norm(vector_bisector_real)

    real_angle = calculate_vector_angle(vector_ab_real, vector_ac_real) / math.pi * 180
    if real_angle > degree:
        while real_angle > degree:
            real_locations_df.iloc[id_a, 0] -= vector_bisector_real[0] / 100
            real_locations_df.iloc[id_a, 1] -= vector_bisector_real[1] / 100
            vector_ab_real = np.array([real_locations_df.iloc[id_b, 0] -
                real_locations_df.iloc[id_a, 0],
                                       real_locations_df.iloc[id_b, 1] -
                                           real_locations_df.iloc[id_a, 1]])
            vector_ac_real = np.array([real_locations_df.iloc[id_c, 0] -
                real_locations_df.iloc[id_a, 0],
                                       real_locations_df.iloc[id_c, 1] -
                                           real_locations_df.iloc[id_a, 1]])
            vector_bisector_real = (vector_ab_real + vector_ac_real) / 2
            vector_bisector_real = vector_bisector_real / np.linalg.norm(vector_bisector_real)
            real_angle = calculate_vector_angle(vector_ab_real, vector_ac_real) / math.pi * 180
    elif real_angle < degree:
        while real_angle < degree:
            real_locations_df.iloc[id_a, 0] += vector_bisector_real[0] / 100
            real_locations_df.iloc[id_a, 1] += vector_bisector_real[1] / 100
            vector_ab_real = np.array([real_locations_df.iloc[id_b, 0] -
                real_locations_df.iloc[id_a, 0],
                                       real_locations_df.iloc[id_b, 1] -
                                           real_locations_df.iloc[id_a, 1]])
```

```
            vector_ac_real = np.array([real_locations_df.iloc[id_c, 0] -
                real_locations_df.iloc[id_a, 0],
                                       real_locations_df.iloc[id_c, 1] -
                                           real_locations_df.iloc[id_a, 1]])
            vector_bisector_real = (vector_ab_real + vector_ac_real) / 2
            vector_bisector_real = vector_bisector_real / np.linalg.norm(vector_bisector_real)
            real_angle = calculate_vector_angle(vector_ab_real, vector_ac_real) / math.pi * 180


plt.plot(real_locations_df.iloc[[4, 7, 8, 4], 0], real_locations_df.iloc[[4, 7, 8, 4], 1],
    color="red")
plt.axis("equal")
plt.show()
for i in range(100):
    adjust_angle_to_certain_degree(5, 8, 9)

    plt.plot(real_locations_df.iloc[[4, 7, 8, 4], 0], real_locations_df.iloc[[4, 7, 8, 4], 1],
        color="red")
    plt.axis("equal")
    plt.show()

    adjust_angle_to_certain_degree(8, 9, 5)

    plt.plot(real_locations_df.iloc[[4, 7, 8, 4], 0], real_locations_df.iloc[[4, 7, 8, 4], 1],
        color="red")
    plt.axis("equal")
    plt.show()

    adjust_angle_to_certain_degree(9, 5, 8)

    plt.plot(real_locations_df.iloc[[4, 7, 8, 4], 0], real_locations_df.iloc[[4, 7, 8, 4], 1],
        color="red")
    plt.axis("equal")
    plt.show()
```