中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

学　　校	中国海洋大学
参赛队号	19104230040
队员姓名	1.秦昌帅
	2.王永站
	3.宋国菊

中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

题 目　　多约束条件下飞行器航迹快速规划

摘　　要：

本文讨论了飞行器在多个约束条件下的航迹规划，并针对具体约束条件，找出了对应的所有校正点的编号、坐标以及总航迹路程，并画出了飞行器在空间中的飞行路径。

针对问题一：

首先根据所给的约束条件，飞行器是在各个校正点之间做垂直误差和水平误差的交替校正，其两点之间的运动轨迹为直线。飞行器从 A 点出发，然后利用校正时误差所允许的范围，判断第一个要经过的水平或垂直校正点是否位于以 A 点为球心的一个球形范围内，接下来利用 A*算法与贪心算法的结合，通过设置 A*的代价函数引入启发信息来找出这一范围内满足航迹最短的校正点，然后从该点出发，重复上述操作，并利用交替校正的方式判断下一个校正点的类型，找出下一个校正点，直到最后一个校正点到终点 B 的距离在相应的误差允许范围内。综合考虑校正点个数和航迹长度，发现对于附件 1，飞行器先进行水平误差校正更好，此时飞行器航迹将经过 10 个校正点，这 10 个校正点编号分别为[521, 64, 80, 237, 282, 33, 11, 403, 594, 501]，总航迹长度为 108436.66299130135m。对于附件 2，综合考虑校正点个数和航迹长度，比较分析得出先进行垂直误差校正更好，此时飞行器航迹将经过 10 个校正点，这 10 个校正点编号分别为[105, 188, 309, 305, 123, 49, 160, 92, 93, 61]，总航迹长度为 109265.6035968887m。

	先经过的校正点类型	经过的校正点个数	航迹总长度/m
附件 1	垂直误差校正点	10	110055.78535055582
	水平误差校正点	10	108436.66299130135
附件 2	垂直误差校正点	10	109265.6035968887
	水平误差校正点	11	107743.15210357278

针对问题二：

采用和问题一相同的做法，只是还需要考虑飞行器受到了转弯半径的影响，除到达第一个校正点之外，在其余两点之间不再是简单的直线运动轨迹，而是一段弧线和直线相连接的运动轨迹，利用空间几何的相关知识进行分析，求出来新的两个点之间的航迹长度，并带到基于问题一的 A*算法中进行改进，得到结论：对于附件 1，选择先垂直误差校正的方式更好，一共经过 10 个校正点，这 10 个校正点编号分别为[503, 200, 136, 80, 237, 278, 375, 172, 340, 277]，总航迹长度为 109030.77832139329m。对于附件 2，选择先进行垂直误差校正的更好，一共经过 10 个校正点，这 10 个校正点编号分别为[105, 188, 309, 305, 123, 49, 160, 92, 93, 61]，总航迹长度为 108272.80202221114m。

	先经过的校正点类型	经过的校正点个数	航迹总长度/m
附件 1	垂直误差校正点	10	109030.77832139329
	水平误差校正点	10	109507.23034500123
附件 2	垂直误差校正点	10	108272.80202221114
	水平误差校正点	11	109435.3781544632

针对问题三：

通过生成随机数的思想来解决概率问题，如果在可能出现问题的点校正失败，则改变其限制条件。多次实验，随机生成随机数，本文中随机取一组代表其航迹路线。对于附件 1，此时飞行器经过 11 个校正点，校正点编号分别为[503, 200, 136, 80, 237, 278, 375, 172, 340, 277, 370]，总航迹长度为 112367.5025846904m，在此规划路径中，飞行器一共经过两个误差校正失败点，分别是水平误差校正点 172 和垂直误差校正点 340，此时飞行器能够成功到达终点的概率为 96%。对于附件 2，此时飞行器经过 10 个校正点，校正点编号分别为 [105, 188, 222, 230, 123, 49, 160, 92, 93, 61]，总航迹长度为 111087.66363688158m，在此规划路径中，飞行器一共经过两个误差校正失败点，均为水平误差校正点，其编号分别为 188 和 92，此时飞行器能够成功到达终点的概率为 96%。

关键词：A*算法、贪心算法、动态规划、航迹规划、交替校正

目录

一、问题重述

1.1 研究背景

随着航空航天技术的快速发展，智能飞行器在复杂环境下的航迹快速规划问题已经成为一个重要课题。由于目前飞行器自身系统结构的限制，飞行器在定位时其定位系统的准确性还有待提高，定位误差会变得越来越大，一旦飞行器定位误差超过一定界限，将会导致其任务失败。飞行器在飞行过程中还会受到油耗、到达时间、地形环境、天气等影响，研究飞行器航迹规划优化控制问题，就要综合考虑可能出现的问题，寻求最优的解决方案，对于目标约束条件，寻求最优的解决方案，因此，寻求飞行器最有航迹问题成为一个重要问题。

1.2 研究问题

随着航行距离的加大，垂直误差和水平误差也会越来越大，这就要求飞行器在飞行过程中不断进行校正，并且只有当垂直误差和水平误差小于一定值时，飞行器才能按照规定航迹飞行，一旦误差过大，飞行器将无法按照规定路径飞行。

（1）对于附件 1 和附件 2 中的数据，飞行器航迹需满足条件（1）-（7），题目中所要求的各个参数已给出，需寻找一条航迹规划路径使得飞行器航迹长度尽可能小，经过的校正区域也要尽可能少。

（2）针对附件 1 和附件 2 中的数据，参数与问题 1 中所给参数相同，此时在需要满足问题一中所给条件（1）-（7）的同时，由于飞行器转弯时收到结构和控制系统的影响，无法完成即时转弯，飞行器最小转弯半径已给出，需要飞行器在飞行时考虑转弯半径的的影响下，寻找飞行器最优航迹路线规划。

（3）由于地形原因，垂直校正点和水平校正点在航迹规划前就已经确定，当飞行器飞到垂直校正点之后，理想情况下垂直误差将会被校正为 0；同样的，当飞行器飞到水平校正点之后，理想情况下水平误差将会被校正为 0。因此，考虑到以上约束条件，我们需要在飞行器到达终点的概率尽可能大的基础上对飞行器从出发点到目的地进行规划。

二、问题分析

2.1 问题一的分析

问题一是规划满足条件（1）-（7）的飞行器航迹路线[1]，要求综合两个优化目标，航迹长度尽可能小以及经过校正区域校正的次数尽可能少。基于飞行器在误差校正时出现的垂直误差和水平误差，且校正垂直误差和水平误差时需要满足的参数条件，本文分析出飞行器在校正误差时需垂直误差和水平误差交替校正，否则飞行器误差过大，将无法按照规划路径飞行，且无法到达目的地。并且为了缩小航迹长度，飞行器在两个校正点之间做直线运动。

首先根据给出可以进行校正的允许误差范围计算出每两个校正点之间的最大距离，从而找到下个校正点坐标的初步范围，再通过判断下个校正点的类型进一步缩小校正点的范围。然后利用 A*算法，估价函数为在这个范围内所有校正点与上个校正点距离与到终点 B 的距离之和。通过比较所有校正点的距离和找出这个范围内最合适的校正点。通过迭代法，再利用飞行器从最后一个校正点到达终点的误差θ，计算出最后一个校正点与终点 B 的距离范围，从而找出所有合适的校正点。

通过比较先进行垂直校正和先进行水平校正的结果，得出最适合的校正点坐标和总航迹长度。

2.2 问题二的分析

问题二是在问题一的基础上增加了飞行器最小转弯半径的约条件束[2]，这样就使得飞行器在两个校正点之间的航迹不再是简单的直线，为了使航迹长度尽可能小，本文分析出飞行器在经过上一个校正点时先以切线方向进入以最小转弯半径为半径的圆中，沿着圆弧运动，再通过一个切点以切线的方向朝下一个校正点做直线运动。

问题二的解法与问题一相同，只是在两个校正点之间的距离的解法和表达式上出现了区别。所以问题二的重点就在于计算两个校正点之间的航迹长度，从而对问题一的解法形成新的约束，找出新的符合要求的校正点。

为了计算出两个校正点的距离，先将它们的运动轨迹投影到二维平面上，通过计算出二维平面上的轨迹长度然后将其扩展到空间中，运用空间几何的相关知识，求出来两个校正点在空间中的最小航迹长度。再通过问题一的算法，找出飞行器经过的所有校正点，并算出总航迹长度。

2.3 问题三的分析

问题三是随着时间的改变，飞行器的飞行环境也会随着动态变化，在此状态下飞行器在进行误差校正时，可能无法达到理想的校正状态。飞行器飞到可能出现问题的点时如果能够将飞行器的水平误差或者垂直误差校正为 0，则校正成功；如果校正失败，校正后的剩余误差会变为$\min(\text{error},5)$。飞行器在经过问题点时校正成功的概率为 80%，校正失败的概率为 20%。飞行器需要在满足航迹约束（1）-（7）的条件下，使得飞行器的航迹长度尽可能小以及经过校正区域校正的次数尽可能少，加上成功到达终点的概率也要尽可能大，寻求最优解。因为飞行器经过问题点时即可知道该点校正成功与否，而且这是一个概率问题，于是可以用随机数的思想来解决。计算机随机生成数，可以假设计算机生成的数在一

个集合中，集合中的一个子集中的点出现的概率为 80%，则其补集中的点出现的概率为 20%，通过随机数可以计算出飞行器的航迹经过问题点能够成功到达终点的概率。

三、符号定义与说明

符号	意义
(x_i, y_i, z_i)	经过第i个校正点坐标
A_i	经过的第i个校正点
r_i	圆半径
$\mathrm{G}_i = d(\mathrm{A}_i, \mathrm{A}_{i+1})$	A_i点与A_{i+1}点的欧式距离
$\mathrm{H}_i = d(\mathrm{A}_i, \mathrm{B})$	A_i点与B点的欧式距离
$[\omega_i(a), \omega_i(b)]$	$\omega_i(a)$为垂直误差，$\omega_i(b)$为水平误差
s	航迹总长度
l	弧长
e	校正失败后的剩余误差
ξ	计算机生成的随机数

四、模型假设

1、不考虑飞行器的大小、形状、质量等因素，把飞行器简化为一个质点。
2、不考虑飞行器的速度和飞行时间。
3、飞行器在经过问题一问题二给出的校正点时均能正常进行校正。
4、在误差允许的范围内，飞行器能按预定轨迹进行飞行。
5、飞行器在航迹规划时只受到题中所给条件的约束。
6、飞行器在经过表中给出的校正点不进行校正的情况时，不把这个点视为校正点。

五、模型的建立与求解

5.1 问题一的模型建立与求解

5.1.1 数据预处理

对于附件中的数据，由于 A 点的初始位置为不在坐标原点，为了便于计算，通过坐标变换(1)，将 A 点看作坐标原点(0,0,0)，其余点通过坐标变换变换成新的坐标。

$$\begin{cases} x' = x - x_A \\ y' = y - y_A \\ z' = z - z_A \end{cases} \tag{1}$$

5.1.2 基于改进的 A*算法与贪心算法三维寻路

综合考虑 Voronoi 图法[3][4]、A*算法[5]、粒子群算法[6]、遗传算法[7]我们最终选用了 A*算法。A*算法最大的特点是通过设置代价函数的方式引入了启发信息,在当前节点选择下一节点的时，A*算法会通过代价函数来计算所有待扩展节点的代价值，然后选择代价最小的节点进行扩展，从而 A*算法能够获得全局最优解。

传统 A*算法基本步骤如图 1 所示：

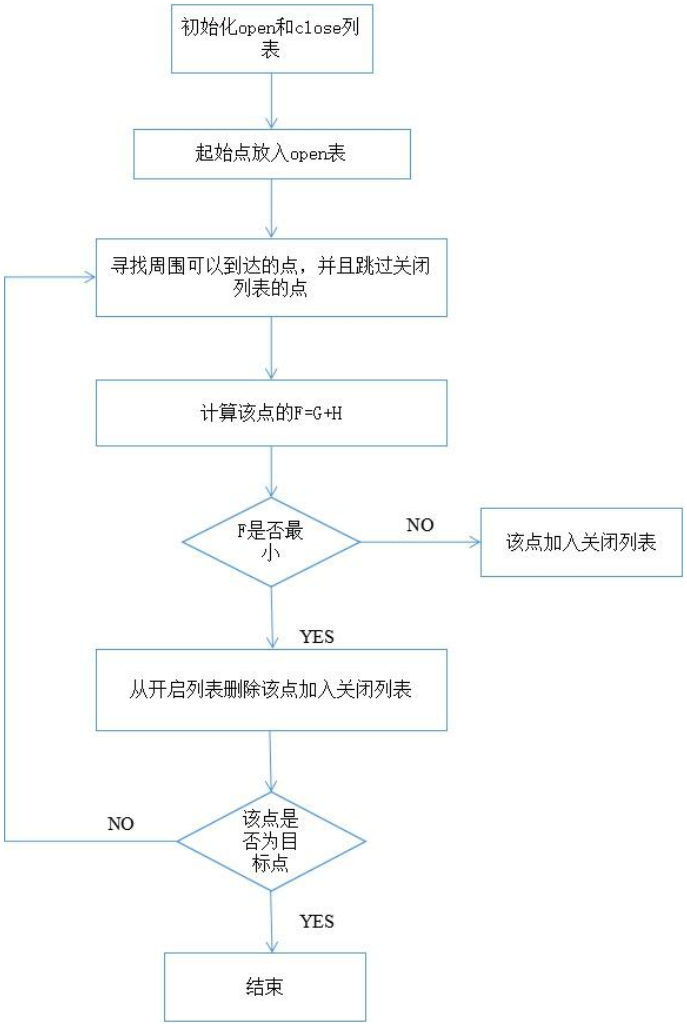


图 1 传统 A*算法的求解步骤

传统的 A*算法由于缺乏方向引导而导致效率低下的问题，我们综合 A*算法与贪心算法，将其拓展到本题的三维空间中，设计了一种动态引导 A*算法。在传统 A*算法以及题目的基础上，限定目标点（限定下一个目标点是水平校正点或者垂直校正点），采用剔除一些目标点减小搜索空间，去构造新的代价函数。将引导点的切换与题目中的要求的校正点的类型结合起来，进行具有端点方向约束的航迹规划[8]，通过设置合适的引导点参数，该方法可以规划出从起始点的特定方向出发、并沿指定方向到达目标的飞行航迹，提升了效率，优化了路径规划结果。

5.1.3 问题一模型的建立

由于飞行器每飞行 1m，垂直误差和水平误差都会随之增加 δ 个单位，A 点距离 B 点的距离明显超过了误差允许的范围，所以飞行器从 A 点到 B 点必经过若干次校正点。

飞行器从 A 点出发，可以先经过垂直校正点，也可以先经过水平校正点。即从 A 点出发的路线可以分为两种情况。同时为使飞行器经过的校正点尽可能少，本文认为在经过一次垂直或水平误差校正点之后，下次将不再经过同样的校正点，即飞行器进行的是交替误差校正的飞行。（如果经过两次相同的校正点，两次校正点的距离较近的话，则飞行器可以一次到达第二个校正点；如果两个校正点的距离较远，则另外一个误差就会过大，为使另一个误差的到校正，还是需要经过另一种类型的校正点，此时相比交替校正的过程，校正点的个数较多）。

同时为了使飞行器的航迹长度相对较小，本文认为飞行器在经过上个校正点和下个校正点以及 A，B 时均是采取直线飞行的方式，故飞行器的飞行航迹为折线。

假设 A 点坐标为 (x_A, y_A, z_A)，B 点坐标为 (x_B, y_B, z_B)。为了便于运算，将 A 点记为 A_0 点，中间经过的第 i 个校正点的坐标为 (x_i, y_i, z_i)。分别算出两种情况下的飞行器飞行路径，再比较两种路径在优化目标下哪种路径最优。

（1）从 A 点出发，先经过垂直误差校正点：

由于进行垂直误差校正时，要求飞行器的垂直误差不大于 α_1 个单位，水平误差不大于 α_2 个单位。所以，以 A 点为球心，$r_1 = \min\left(\frac{\alpha_1}{0.001}, \frac{\alpha_2}{0.001}\right)$ 为球半径画一个球，如图 2 所示。

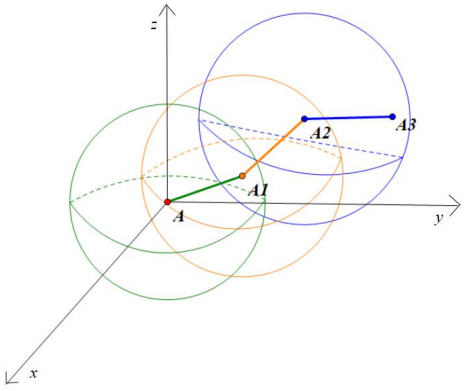


图 2 以 A_i 为球心的范围示意图

只有当该点在球内时，飞行器才能进行垂直误差校正，否则，误差过大，飞行器将无法飞到终点。所以可以通过这个条件对垂直误差校正点进行筛选，即垂直误差校正点 A_1 的

坐标(x_1, y_1, z_1)应满足

$$d(A_0,A_1)=\sqrt{(x_1-x_A)^2+(y_1-y_A)^2+(z_1-z_A)^2}<r_1 \tag{2}$$

同时求出的点 A_1 与 B 点的距离

$$d(A_1,B)=\sqrt{(x_1-x_B)^2+(y_1-y_B)^2+(z_1-z_B)^2} \tag{3}$$

A*算法代价函数为

$$\mathrm{F}=\mathrm{G}+\mathrm{H} \tag{4}$$

为了方便表述每次的比较过程，这里

$$\mathrm{G}_i=d(\mathrm{A}_{i\text{-}1},\mathrm{A}_i),\mathrm{H}_i=d(\mathrm{A}_i,\mathrm{B}),\ \mathrm{F}=d_i \tag{5}$$

为使求出的点 A_1 距离 A 点与 B 点距离之和最小，即需要寻找的垂直校正点 A_1 满足

$$d_1=d(\mathrm{A}_0,\mathrm{A}_1)+d(\mathrm{A}_1,\mathrm{B}) \tag{6}$$

最小。

在此限制条件下求出垂直误差校正点 A_1 的坐标(x_1,y_1,z_1)，找出该点在附件中的位置编号。按照垂直误差与水平误差交替的方法来消除误差，假设第i步消除了垂直误差，此时飞行器的误差仅为水平误差，而且仅为第i步新产生的水平误差$d(\mathrm{A}_{i-1},\mathrm{A}_i)\times\delta$（之前的水平误差在经过第$i-1$个水平校正点时变为 0），则此时的垂直和水平误差为：

$$\omega_i=(0,d(\mathrm{A}_{i-1},\mathrm{A}_i)\times\delta)$$

在进行第$i+1$步时，以A_i点为球心，此时飞行器已经存在了上述误差，再经过下个水平校正点A_{i+1}时又会产生新的误差c_i，所以半径的寻找发生了变化，飞行器要进行水平误差校正时，需要满足的条件为垂直误差c_i不大于β_1个单位，水平误差满足$d(\mathrm{A}_{i-1},\mathrm{A}_i)+c_i<\beta_2$，故此时要进行水平校正时，需要满足的条件变为：

$$r_{i+1}=\min\left\{\frac{\beta_1}{\delta},\frac{\beta_2-d(\mathrm{A}_{i-1},\mathrm{A}_i)\times\delta}{\delta}\right\} \tag{7}$$

为球半径，寻找在球内的点，使$d_{i+1}=d(\mathrm{A}_i,\mathrm{A}_{i+1})+d(\mathrm{A}_{i+1},\mathrm{B})$最小，且该点为水平误差校正点，求出$\mathrm{A}_{i+1}(x_{i+1},y_{i+1},z_{i+1})$，此时误差$\omega_{i+1}=(d(\mathrm{A}_i,\mathrm{A}_{i+1})\times\delta,0)$；再进行$i+2$步时，以$\mathrm{A}_{i+1}$点为球心，此时飞行器只存在垂直误差，飞行器在$\mathrm{A}_{i+1}$到$\mathrm{A}_{i+2}$新产生的误差为$c_{i+1}$，所以飞行器在到达垂直校正点$\mathrm{A}_{i+2}$时，需要满足垂直误差$d(\mathrm{A}_i,\mathrm{A}_{i+1})\times\delta+c_{i+1}<\alpha_1$，水平误差满足$c_{i+1}<\alpha_2$，此时的半径应为

$$r_{i+2}=\min\left\{\frac{\alpha_1-d(\mathrm{A}_i,\mathrm{A}_{i+1})\times\delta}{\delta},\frac{\alpha_2}{\delta}\right\},\tag{8}$$

寻找在球内的点，使$d_{i+2}=d(\mathrm{A}_{i+1},\mathrm{A}_{i+2})+d(\mathrm{A}_{i+2},\mathrm{B})$最小，且该点为垂直误差校正点，求出$\mathrm{A}_{i+2}$，此时误差$\omega_{i+2}=(0,d(\mathrm{A}_{i+1},\mathrm{A}_{i+2})\times\delta)$，以此类推.....

由于飞行器到达终点B时只需水平误差和垂直误差均小于θ个单位，故当在经过最后一个校正点A_n之后，存在的误差$d(\mathrm{A}_{n-1},\mathrm{A}_n)\times\delta$和飞向B点产生的误差$d(\mathrm{A}_n,\mathrm{B})\times\delta$在满足

$$d(\mathrm{A}_{n-1},\mathrm{A}_n)\times\delta+d(\mathrm{A}_n,\mathrm{B})\times\delta<\theta\tag{9}$$

时飞行器可以从A_n校正点直接飞到B点即终点，飞行器的航迹即可求出。

（2）从A点出发，先经过水平误差校正点：

同先经过垂直误差校正点的思想一样，不同的是先进行水平误差校正时，要求飞行的垂直误差不大于β_1个单位，水平误差不大于β_2个单位。以A点为球心$r_1=\min\left(\frac{\beta_1}{0.001},\frac{\beta_2}{0.001}\right)$为球半径画一个球，只有在球内水平误差校正点可以满足条件，即水平误差校正点A_1的坐标(x_1,y_1,z_1)应满足式(2)，应用式(3)求出的点A_1与B点的距离$d(A_1,B)$。

求出的点A_1应使其与点A与其与点B点之间的距离之和为最小值，由式(5)，令$\mathrm{G}_0=d(\mathrm{A}_0,\mathrm{A}_1)$，$\mathrm{H}_0=d(\mathrm{A}_1,\mathrm{B})$，使$\mathrm{F}=\mathrm{G}+\mathrm{H}$得取得最小值，即式(6)最小。在此限制条件下求出水平误差校正点A_1的坐标，找出该点在附件中的位置编号。

同理，按照垂直误差与水平误差交替的方法来消除误差，假设第i步消除了垂直误差，则下一步需要找水平校正点，步骤同上，直至找出最后一个校正点。

5.1.4 问题一的求解

（1）对于附件一中的数据，由于A点的坐标为$(0,50000,5000)$，为简化运算，运用式(1)坐标变换，有

$$\begin{cases}x'=x-0\\y'=y-50000\\z'=z-5000\end{cases},$$

将附件1中的坐标看作以A为坐标原点的空间坐标系中，运用A*算法，假设先进行垂直误差校正，得出飞行器从A点出发到B点中间需要经过10个校正点，这10个校正点在附件一中的编号分别为[503, 200, 136, 80, 237, 278, 375, 172, 340, 277]，总航迹长度为$s=110055.78535055582$m。结果如表1所示：

表 1 附件1先进行垂直误差校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点A

503	13.38791985	13.38791985	1
200	0.865057806	14.25297766	0
136	14.27111329	13.40605548	1
80	4.286747706	17.69280319	0
237	8.914492276	4.62774457	1
278	17.94221998	22.56996455	0
375	23.48405057	5.541830589	1
172	15.49641641	21.038247	0
340	21.64492891	6.148512498	1
277	12.00237598	18.15088848	0
612	16.35090448	28.35328046	终点 B

假设先进行水平误差校正，得出飞行器从 A 点出发到 B 点中间需要经过 10 个点，编号分别为[521, 64, 80, 237, 282, 33, 11, 403, 594, 501]。总航迹长度 $s = 108436.662991301355$m。

结果如下表 2 所示：

表 2 附件 1 先进行水平误差校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
521	9.626510211	9.626510211	0
64	21.75543959	12.12892937	1
80	11.42105429	23.54998366	0
237	16.04879886	4.62774457	1
282	13.6816771	18.30942167	0
33	16.15160156	2.469924455	1
11	10.92386909	13.39379354	0
403	23.76474756	12.84087847	1
594	11.0291151	23.86999357	0
501	22.22606141	11.19694631	1
612	8.490014015	19.68696033	终点 B

针对附件 1 中的校正点，通过比较两组数据，校正点的个数相同的情况下，先进行水平误差校正的航迹比先进行垂直误差校正的航迹长度小，因此选择先进行水平误差校正的航迹。即飞行器先进行水平误差校正，此时其航迹经过 10 个误差校正点，总航迹长度 $s = 108436.66299130135$m。

其飞行路径如图 3 所示：

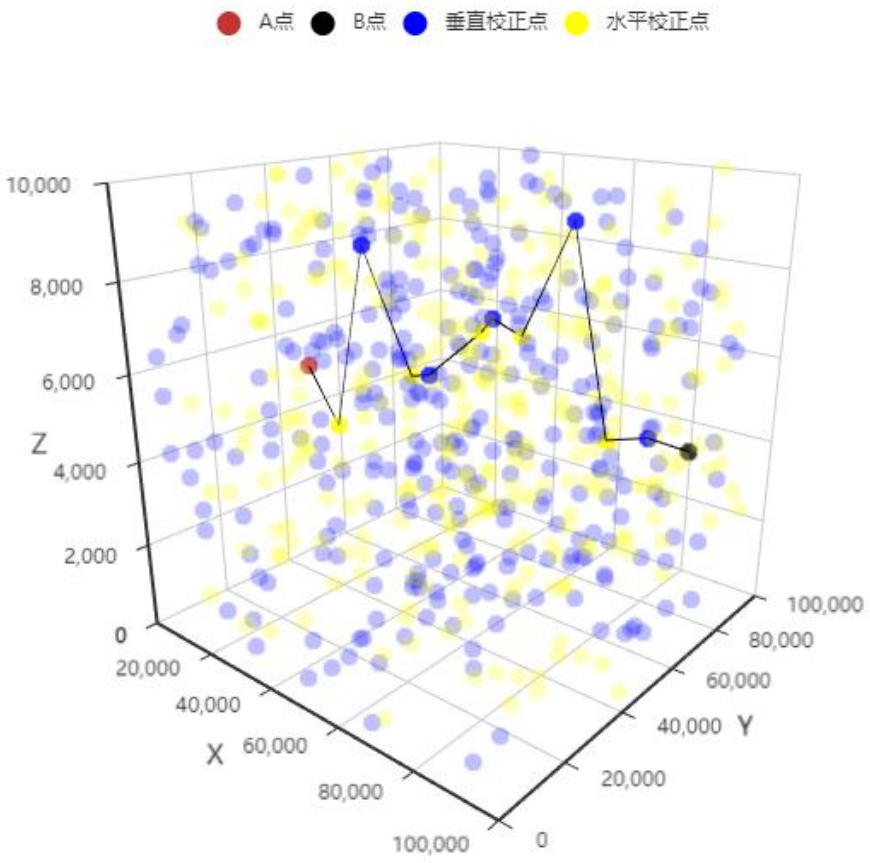


图 3 附件 1 先进行水平误差校正点的飞行路径

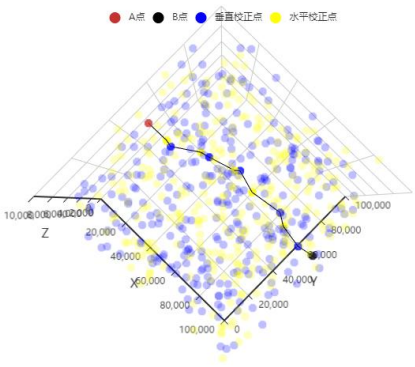


图 3.1 z-y 平面飞行路径

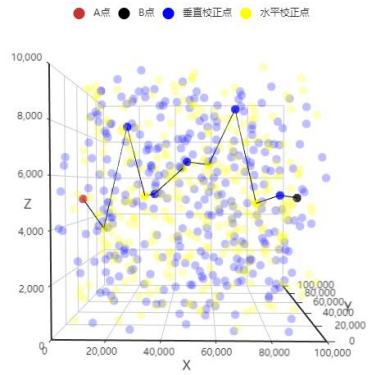


图 3.2 x-z 平面飞行路径

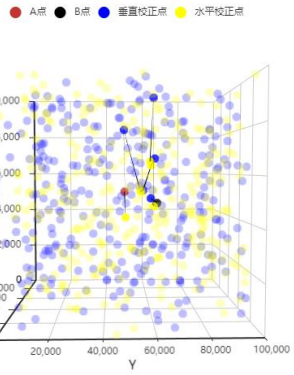


图 3.3 y-z 平面飞行路径

（2）针对附件二中的数据，采取相同的方式进行求解。得出以下结果：

先进行垂直校正时，总共经过 10 个校正点，编号分别为[105, 188, 309, 305, 123, 49, 160, 92, 93, 61]，总航迹长度 $s=109265.6035968887$m。结果如表 3 所示：

表 3 附件 2 先进行垂直校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
105	13.82090858	13.82090858	1

188	10.66821176	24.48912033	0
309	24.55095559	13.88274383	1
305	5.968714547	19.85145838	0
123	15.17310764	9.204393096	1
49	11.79019446	20.99458755	0
160	18.30400023	6.513805771	1
92	5.776163625	12.2899694	0
93	15.26088202	9.484718396	1
61	9.834209702	19.3189281	0
326	12.32153984	22.15574954	终点 B

先进行水平校正时，总共经过 11 个校正点，编号分别为[163, 238, 234, 309, 305, 123, 49, 160, 92, 93, 61]。总航迹长度 $s=107743.15210357278$m。结果如表 4 所示：

表 4 附件 2 先进行水平校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 *A*
163	13.28789761	13.28789761	0
238	21.19023956	7.902341949	1
234	2.347539566	10.24988151	0
309	15.65917311	13.31163354	1
305	5.968714547	19.28034809	0
123	15.17310764	9.204393096	1
49	11.79019446	20.99458755	0
160	18.30400023	6.513805771	1
92	5.776163625	12.2899694	0
93	15.26088202	9.484718396	1
61	9.834209702	19.3189281	0
326	12.32153984	22.15574954	终点 *B*

针对附件二中的数据，经过比较，虽然先垂直校正总航迹长度比先水平校正总航迹长度要大，但是校正点却少一个，航迹之差为 1522.4514933159，误差不到两个单位，可以忽略不计，所以应选择先进行垂直误差校正的方式，即总共经过 10 个点，总航迹长度 $s=109265.6035968887$m。其飞行路线如图 4 所示：

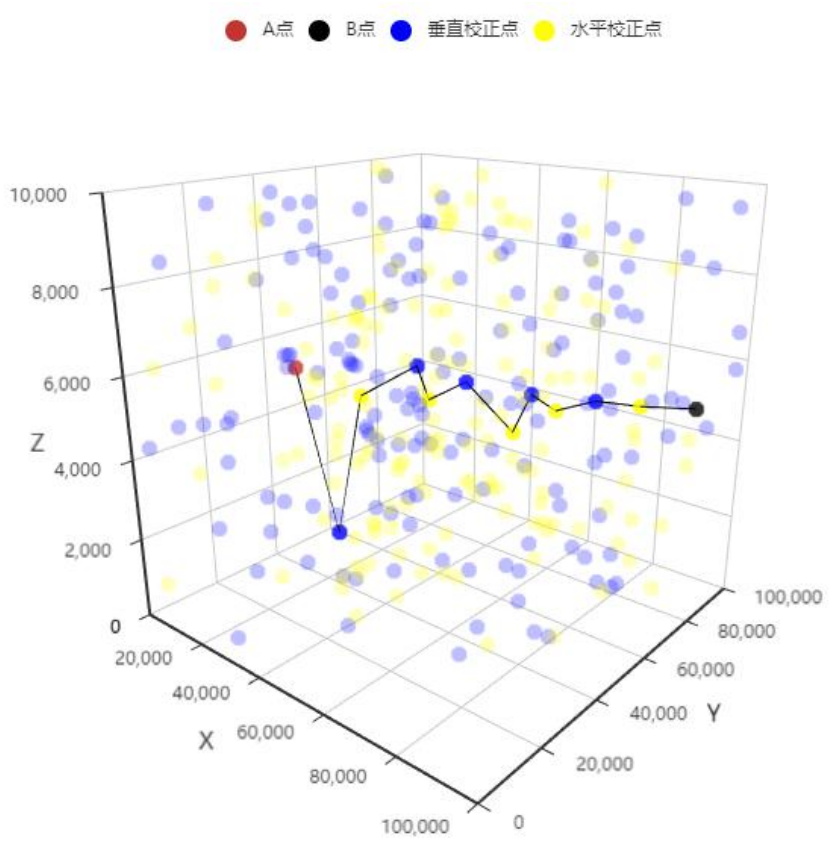


图 4 附件 2 先进行垂直校正的飞行路径

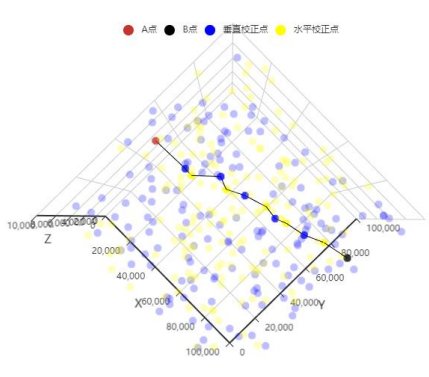


图 4.1 z-y 平面飞行路径

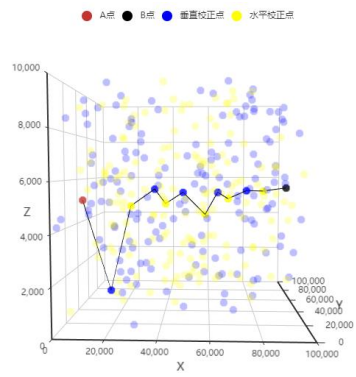


图 4.2 x-z 平面飞行路径

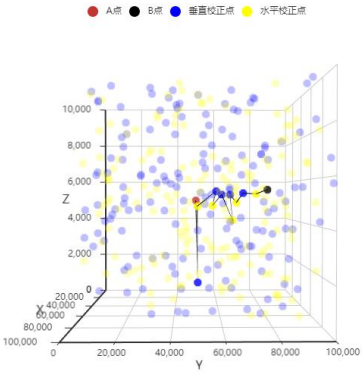


图 4.3 y-z 平面飞行路径

算法的有效性和复杂度分析：

（1）算法的有效性分析

飞行器从前一个点到后一个点的过程中，将飞行器的运动轨迹简单得转化为直线运动，即飞行器的运动轨迹为折线图，不考虑飞行器实际所进行的方向调整，由此可能会造成一定的轨迹误差。在运用软件对数据处理的过程中，本文选择了与所给数据同样的十位小数精度，由此可能会造成一定程度上的精度误差。上述两种误差在计算中数值较小，均可忽略不计，故本文认为在处理问题一时的算法是可靠有效的。

（2）算法的复杂度分析

①从算法的时间复杂度方面分析，问题一所有源程序文件中主要求解代码块对于附件 1

的平均执行时间为 0.017s，对于附件 2 的平均执行时间为 0.007s，说明本题的模型是有效的。

②从算法的空间复杂度方面分析，两个表格中的数据量是不会改变的，算法中涉及到的全局变量较多，局部变量的规模也是确定的，所以空间复杂度在该题目的基础上为一个常量。

时间复杂度和空间复杂度往往是相互影响的，我们综合考虑算法的使用频率，算法处理的现有数据量的大小，以及算法本身所具有的特性，在此基础上改进 A*算法，我们认为算法的复杂度总体还是比较优的。

5.2 问题二的模型建立与求解

5.2.1 问题二模型的建立

基于问题一，飞行器在转弯时受到结构和控制系统的限制，前进方向无法突然改变，需要有一个缓冲空间，且飞行器的最小转弯半径为 200m，本文认为飞行器从 A 点出发时到达第一个校正点不需要转弯，所以在从 A 点出发到下一个校正点 A_1 时飞行器可以沿直线飞行，此时问题一中求出的校正点 A_1 便为飞行器从 A 点出发所到达的第一个校正点。本题将通过迭代法来解决问题，具体思路是确定了第 $i-1$ 和第 i 个校正点之后，运用校正点 A_{i-1} 和 A_i 迭代来求出第 $i+1$ 个点 A_{i+1}。

在新的约束条件下，飞行器在两个校正点之间的飞行轨迹将不再是直线，为了使飞行轨迹尽可能的缩短，本文认为飞行器在经过一个校正点 A_i 之后，将以这个方向为切线做一个以 200m 为半径以 O_i 为圆心的圆弧运动（O_i 将会在建模的过程中给出具体表达式），过圆弧上的 J_i 点以切线的方向经过直线运动到达下一个校正点 A_{i+1}，在二维平面上的投影如图 5 所示：

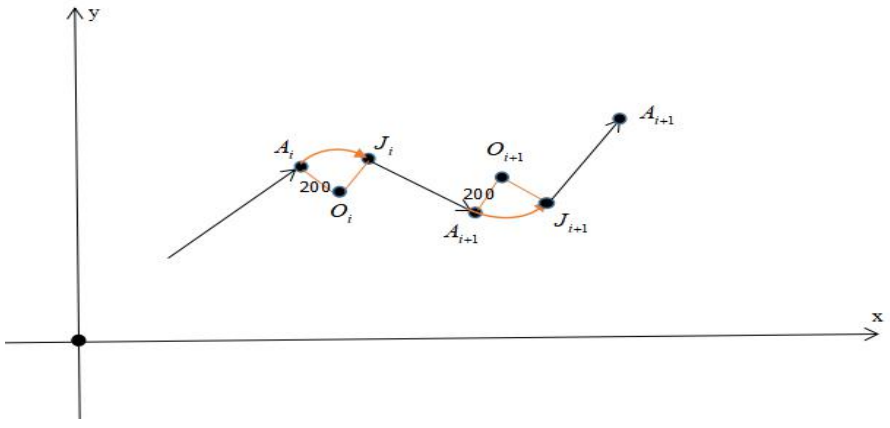


图 5 飞行器运动轨迹二维投影

在解决问题二时，本文认为问题一中的交替校正依然可以满足本题的条件。和问题一算法类似，唯一的不同便是 A_iA_{i+1} 的长度算法不同。下面以计算 A_1A_2 的距离为例，来说明 A_iA_{i+1} 距离的变化。

步骤和问题一相似，当飞行器第一次进行的是垂直校正时，由第一题已知 A_1 的坐标，将要寻找满足条件的水平校正点 A_2。先通过问题一中的约束条件利用式(7)先确定半径 r_2，A_2 点在以 A_1 为球心以 r_2 为半径的球内。

为了简化运算，本文先在二维平面上进行相应的运算如图 6。

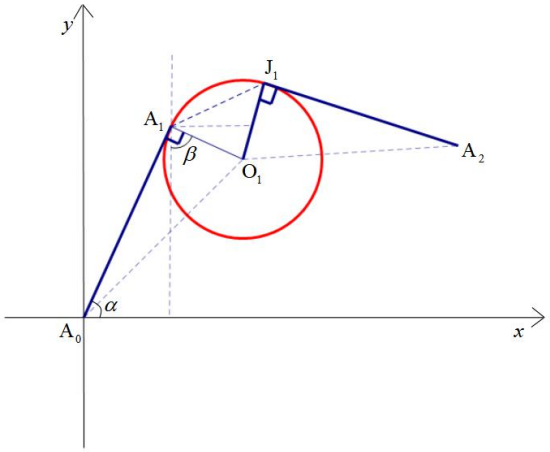


图 6 二维平面内运动详解

在已知 $A_1(x_1,y_1)$，A_0A_1 的水平距离为 $d_1=\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}$。如图 3 所示，因为 $\angle\alpha$ 与 $\angle\beta$ 和同一个角互余，所以 $\angle\alpha=\angle\beta$。且 $\tan\alpha=\left|\dfrac{y_1-y_0}{x_1-x_0}\right|$，所以

$$\alpha=\arctan\left|\frac{y_1-y_0}{x_1-x_0}\right|。\tag{10}$$

如果 A_2 的横坐标比 A_1 的横坐标大的话，O_1 的横坐标就会比 A_1 的横坐标大，纵坐标也是如此，反之也类似，所以圆心 O_1 的坐标可以表示为

$$\left(x_1+\frac{x_2-x_1}{|x_2-x_1|}\times 200\sin\alpha,\ y_1+\frac{y_2-y_1}{|y_2-y_1|}\times 200\cos\alpha\right)。\tag{11}$$

O_1 离下一校正点 A_2 的水平距离可以表示为

$$d(O_1,A_2)=\sqrt{(x_2-O_{1x})^2+(y_2-O_{1y})^2}。\tag{12}$$

（其中 O_{1x} 表示圆心 O_1 的横坐标，O_{1y} 表示圆心 O_1 的纵坐标）

因为 $O_1J_1\perp J_1A_2$，所以利用式(12)求出的距离得出圆弧上的切点 J_1 离 A_2 的距离

$$d(J_1,\mathrm{A}_2)=\sqrt{d(O_1,\mathrm{A}_2)^2-200^2}\ 。\tag{13}$$

下面对J_1的坐标(J_{1x},J_{1y})进行求解。已经求出了O_1的坐标、$J_1\mathrm{A}_2$的长度，所以把J_1看成是以O_1为圆心，200 为半径的圆与以A_2为圆心，$J_1\mathrm{A}_2$为半径的圆的一个交点。

可得方程组

$$\begin{cases}(J_{1x}-O_{1x})^2+(J_{1y}-O_{1y})^2=200^2\\(J_{1x}-x^2)^2+(J_{1y}-y_2)^2=J_1\mathrm{A}_2{}^2\end{cases}\tag{14}$$

从而由式(14)求出两个圆之间的交点，本文认为两个圆之间必有两个交点，如果两个圆的交点不是两个的话，说明所要寻找的A_2不符合条件，应该去掉。对于这两个交点，通过上图可以观察出当$y_2<y_1$时，交点应该选择y值较大的那个交点，当$y_2>y_1$时，应该选择y值较小的那个交点。通过这种判断标准，找到了J_1的坐标(J_{1x},J_{1y})。

然后再得出

$$d(\mathrm{A}_1,J_1)=\sqrt{(J_{1x}-x_1)^2+(J_{1y}-y_1)^2}\ 。\tag{15}$$

通过上述计算，利用式(15)可以利用余弦定理求出$\angle\mathrm{A}_1O_1J_1$，具体公式如下：

$$d(\mathrm{A}_1,J_1)^2=200^2+200^2-2\times200\times200\cos\angle\mathrm{A}_1O_1J_1\ 。$$

所以得到弧长$l_{\mathrm{A}_1J_1}=200\pi/180$。

现在将二维平面扩展到三维空间中，计算A_1A_2的最短距离。正如前面所描述的，这时的飞行轨迹是一段弧线加上一段直线，它在二维空间的投影如图 6 所示，而它在三维空间的轨迹为了方便表示，将弧长放在圆柱体中表示，如图 7 所示：

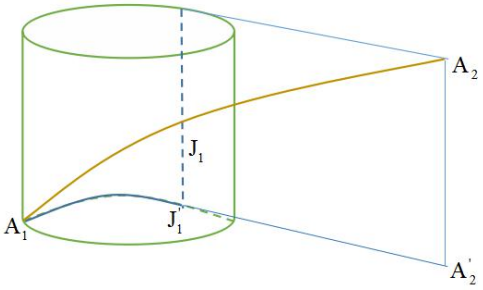


图 7　三维空间运动轨迹

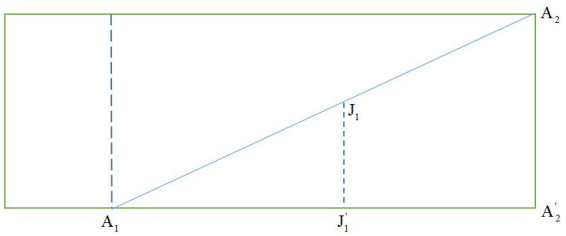


图 8 圆柱展开示意图

在图 7 中可以看到，要使$\mathrm{A}_1\mathrm{A}_2$的距离最短，要将圆柱剪开，如图 8 所示，那么刚才求出的弧长$l_{A_1J_1}$在此时变成了直线段$\mathrm{A}_1J_1{}'$，所以$\mathrm{A}_1\mathrm{A}_2$的距离在此图中可以用勾股定理轻易求

出，$d(\mathrm{A}_2,\mathrm{A}_2^{'})=|z_2-z_1|$，$J_1^{'}\mathrm{A}_2^{'}$的长度就等于上面求出的二维平面中的$J_1\mathrm{A}_2$的长度。所以就有$d(\mathrm{A}_1,\mathrm{A}_2)=\sqrt{(d(\mathrm{A}_1,J_1^{'})+d(J_1^{'},\mathrm{A}_2^{'}))^2+d(\mathrm{A}_2,\mathrm{A}_2^{'})^2}$。

利用式(5)要寻找的A_2点就要满足$d_2=d(\mathrm{A}_1,\mathrm{A}_2)+d(\mathrm{A}_2,\mathrm{B})$最小且是水平校正点。

然后再寻找下一个垂直校正点A_3。这时通过J_1、A_2来寻找A_3（J_1的作用相当于A_0）计算方法和上面相同，如图，通过迭代法，每次都是通过J_{i-1}、A_i来寻找A_{i+1}。

接下来的做法与问题一完全一致，只是每次确定A_i点时需要用到J_{i-1}，$\mathrm{A}_i\mathrm{A}_{i+1}$的距离需要上面介绍的方法来计算，据此来寻找下一个校正点。

当飞行器第一次进行的是水平误差校正时，做法与上述完全一致。

5.2.2 问题二的求解

对于附件 1 的数据，当飞行器先进行垂直误差校正时，由问题 1 的求解，A_0、A_1的坐标已知。A_1的坐标为(11392.9607416196, 56973.0182393612, 4097.85801775604)，然后通过上面介绍的$\mathrm{A_iA_{i+1}}$距离的算法以及其他与问题一一致的求解步骤，得到飞行器的飞行轨迹过 10 个校正点，编号分别为 [503, 200, 136, 80, 237, 278, 375, 172, 340, 277]，总航迹长度为 109030.77832139329m，结果如表 5 所示：

表 5 附件 1 先进行垂直校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
503	13.38791985	13.38791985	1
200	0.727629068	14.11554892	0
136	14.05570686	13.3280778	1
80	4.142323274	17.47040107	0
237	8.607890986	4.465567712	1
278	17.877042	22.34260971	0
375	23.34178359	5.464741587	1
172	15.41429542	20.879037	0
340	21.4341625	6.019867085	1
277	11.85241005	17.87227714	0
612	16.35090448	28.20331453	终点 B

当飞行器先进行水平校正时由问题 1 的求解，A_0、A_1的坐标已知，A_1的坐标为(9522.69064718244, 50387.4587105637, 3644.29448812232)。与上述步骤相同，得到飞行器的飞行轨迹过 10 个校正点，编号分别为[521, 64, 80, 237, 155, 375, 11, 403, 594, 501]
总航迹长度为 109507.23034500123m，结果如表 6 所示：

表 6 附件 1 先进行水平校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
521	9.626510211	9.626510211	0
64	21.7335827	12.10707249	1
80	11.27490181	23.3819743	0
237	15.73579263	4.460890826	1
155	11.11806686	15.57895769	0
375	22.68573018	11.56766332	1
11	5.948803142	17.51646646	0
403	18.69078008	12.74197694	1
594	11.01584207	23.75781901	0
501	22.17133073	11.15548866	1
612	8.490014015	19.64550268	终点 B

通过比较两组数据，两者通过的校正点的个数相同，但先垂直校正的总航迹长度略微小一些。所以针对附件一的数据，选择先垂直校正的方式，一共经过 10 个校正点，总航迹长度为 109030.77832139329m，航行轨迹如图 9-图 9.3 所示。为了更好地表示圆弧运动，航迹在二维平面中的投影如图 9.4

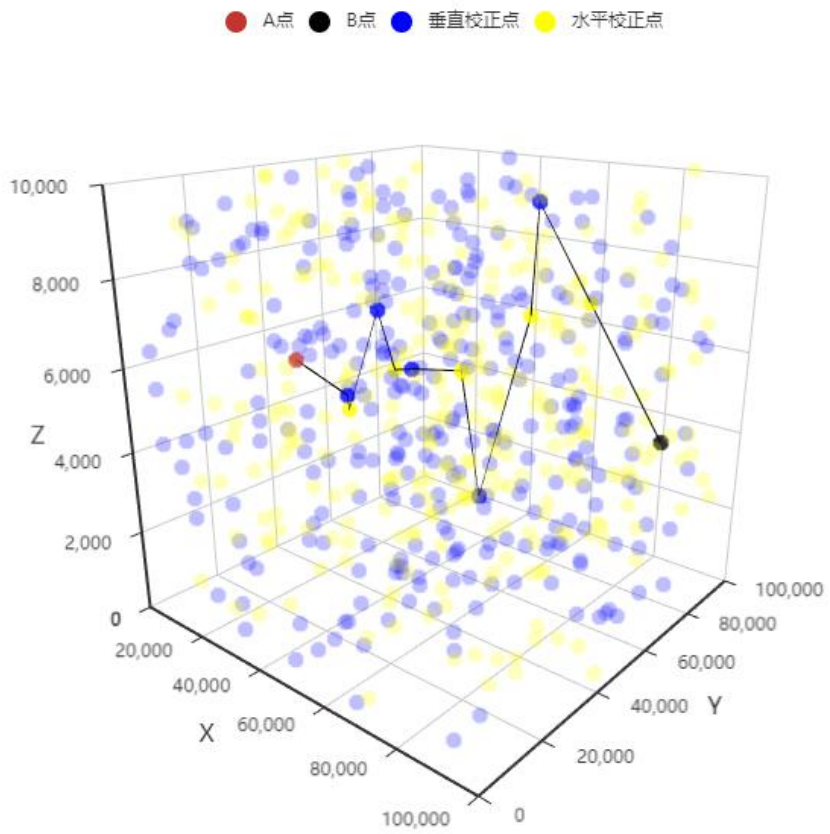


图 9 附件 1 先进行垂直校正的飞行路径

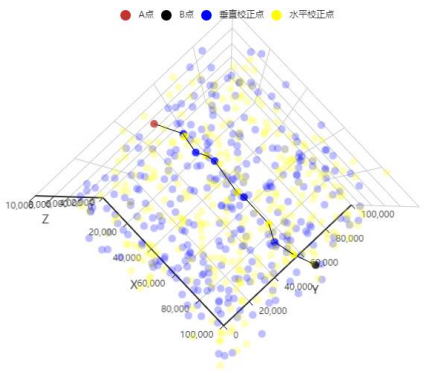


图 9.1　z-y 平面飞行路径

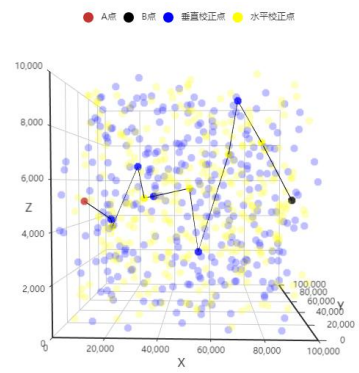


图 9.2　x-z 平面飞行路径

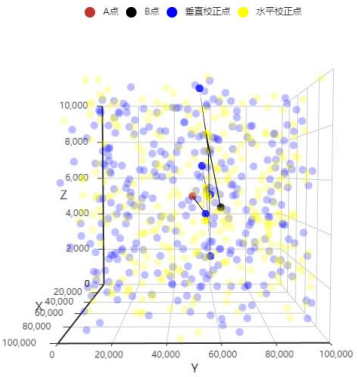


图 9.3　y-z 平面飞行路径

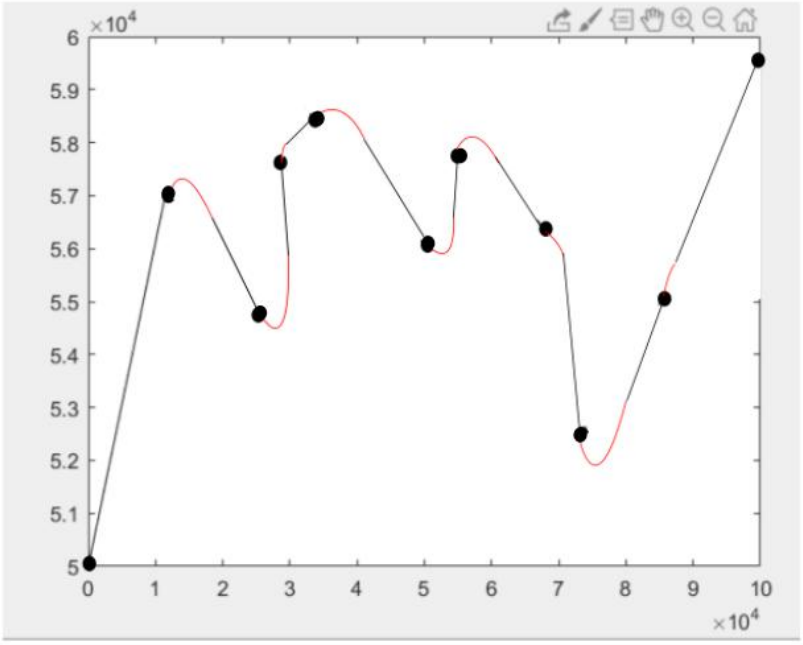


图 9.4 航迹在二维平面中的投影

对于附件 2 的数据，当飞行器先进行垂直校正时，A_1 的坐标为 (13090.4194711808, 50889.4503775067, 656.34830892419)，飞行器的飞行轨迹过 10 个点，编号分别为[105, 188, 309, 305, 123, 49, 160, 92, 93, 61]。总航迹长度为 108272.80202221114m，结果如表 7 所示：

表 7 附件 2 先进行垂直校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
105	13.82090858	13.82090858	1
188	10.65390919	24.47481777	0
309	24.41175426	13.75784507	1
305	5.805955091	19.56380016	0
123	14.87910192	9.073146824	1

49	11.65838471	20.73153153	0
160	18.0679684	6.409583694	1
92	5.662613465	12.07219716	0
93	15.04401725	9.381403781	1
61	9.727511776	19.10891556	0
326	12.32153984	22.04905162	终点 B

当飞行器先进行水平校正时，A_1 的坐标为 (12712.8372282571, 53865.1451047564, 4887.54630795409)，飞行器的飞行轨迹过 11 个点，编号分别为 [163, 128, 227, 309, 305, 123, 49, 160, 92, 93, 61]总航迹长度为 109435.3781544632m，结果如表 8 所示：

表 8 附件 2 先进行水平校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
163	13.28789761	13.28789761	0
128	23.2577178	9.969820185	1
227	10.85910775	20.82892793	0
309	16.15442751	5.295319767	1
305	5.789048757	11.08436852	0
123	14.86219558	9.073146824	1
49	11.65838471	20.73153153	0
160	18.0679684	6.409583694	1
92	5.662613465	12.07219716	0
93	15.04401725	9.381403781	1
61	9.727511776	19.10891556	0
326	12.32153984	22.04905162	终点 B

经过比较两组数据，选择先进行垂直误差校正的航迹，因为经过的校正点少，且总航迹长度更小，一共经过 10 个校正点，总航迹长度为 108272.80202221114m。
航行轨迹如图 10-图 10.3 所示，航迹在二维平面中的投影如图 10.4。

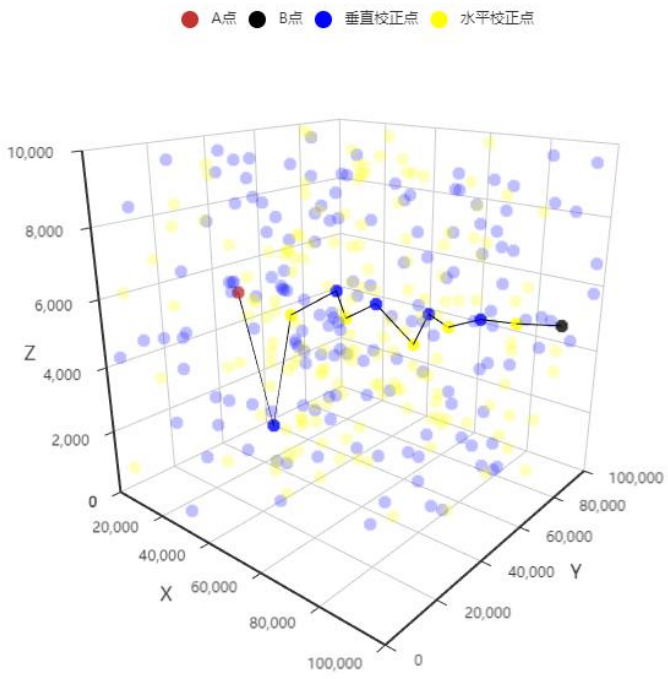


图 10 附件 2 先进行垂直校正的飞行路径

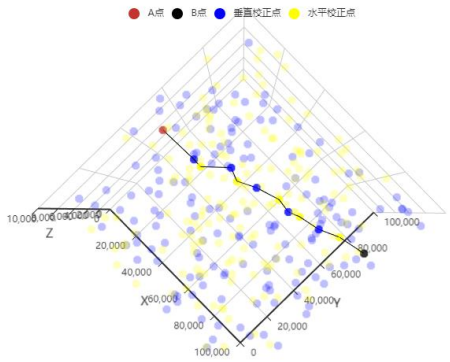


图 10 .1 z-y 平面飞行路径

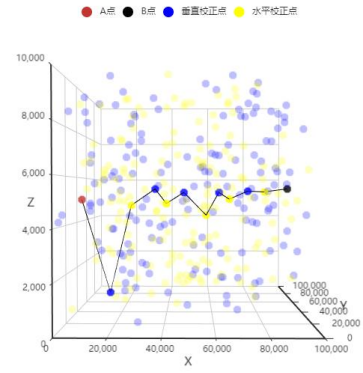


图 10.2 x-z 平面飞行路径

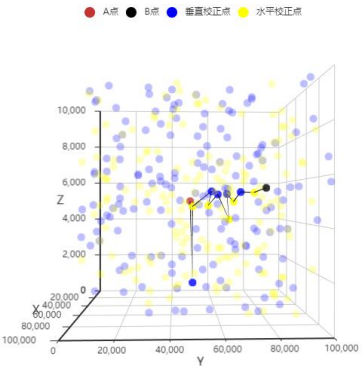


图 10.3 y-z 平面飞行路径

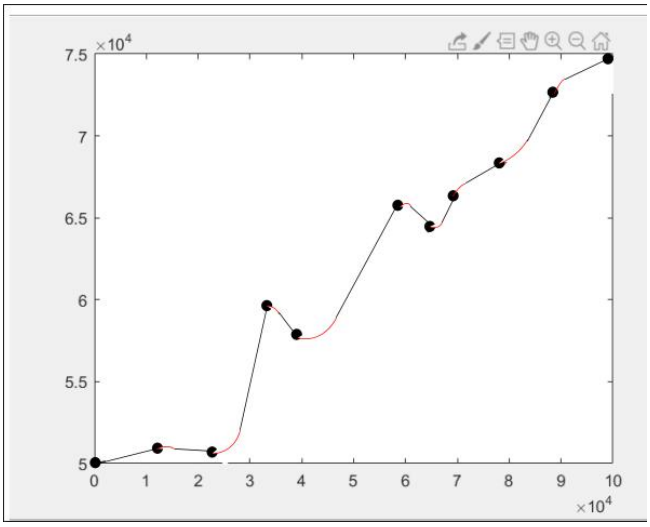


图 10. 4 航迹在二维平面中的投影

算法的有效性和复杂度分析：

（1）算法的有效性分析

由于飞行器除了从A_0到A_1是直线航迹外，其余路径均为一段弧和一段直线。而本文在对最后一个校正点到B的距离的计算仍然采取了直线的计算方式，在判断最后的误差范围和总航迹长度中均用到了这段距离，由此产生了模型误差。

（2）算法的复杂度分析

①从算法的时间复杂度方面分析，问题二所有源程序文件中主要求解代码块对于附件1的平均执行时间为0.019s，对于附件2的平均执行时间为0.008s，耗时较少。

②从算法的空间复杂度方面分析，两个表格中的数据量是不会改变的，算法中涉及到的全局变量较多，局部变量的规模也是确定的，所以空间复杂度在该题目的基础上为一个常量。

5.3 问题三的模型的建立与求解

5.3.1 模型的建立

问题一是在理想条件下飞行器经过误差校正点时进行误差校正可以完全将垂直误差或者水平误差校正为0，但是考虑实际情况，例如天气等不可控因素，飞行器部分校正点能够成功将误差校正的概率为80%，如果校正失败，校正后会出现剩余误差，这将会导致即使飞行器到达误差校正点也无法达到理想的校正结果。题目中给出了校正失败后的剩余误差为$\min(\text{error},5)$，据此重新规划飞行器的航迹问题。

基于问题一的思想，飞行器从A点出发到达第一个点时依然分为两种情况，第一种情况为飞行器先飞到垂直误差校正点，第二种情况为飞行器先飞到水平误差校正点。依然使用垂直误差与水平误差交替校正的思想。

（1）从A点出发，先经过垂直误差校正点以A点为球心，$r_1=\min\left(\frac{\alpha_1}{\delta},\frac{\alpha_2}{\delta}\right)$为球半径画一个球，只有在球内垂直误差校正点可以满足条件，即垂直误差校正点A_1的坐标(x_1,y_1,z_1)应满足式(2)。

利用式(3)求出的点A_1与B点的距离$d(A_1,B)$。

求出的点A_1距离A点与B点距离之和最小，令$G_1=d(A_0,A_1)$，$H_1=d(A_1,B)$，使得$F_1=G_1+H_1$取得最小值，即$d_1=d(A_0,A_1)+d(A_1,B)$最小。在此限制条件下求出垂直误差校正点A_1的坐标，找出该点在附件中的位置编号。此时需要判断垂直误差校正点A_1是否为可能出现问题的点。由于飞行器到达该校正点时即可知道在该点处是否能够校正成功，因此需采用随机数的思想来判断飞行器在该点处是否校正成功。

如果A_1点为可能出现问题的点，则在计算机中随机生成一个1-10的整数，由于飞行器在这些可能出现问题的点能够将误差校正为0的概率为80%，即这些可能出现问题的点不能将误差校正为0的概率为20%。因此，我们令当随机生成的整数$9\le\xi\le10$时，认为飞行

器校正失败，此时校正后的剩余误差为$e_1=\min\left(d(\mathrm{A},\mathrm{A}_1)\times\delta,5\right)$，此时飞行器的垂直误差和水平误差分别为$\omega_1=\left(e_1,d(\mathrm{A},\mathrm{A}_1)\times\delta\right)=\left[\omega_1(a),\omega_1(b)\right]$，且找出校正失败的垂直误差校正点的编号；如果随机生成的整数$1\le\xi\le8$或者垂直误差校正点A_1为正常校正点，则此时误差为

$$\omega_1=\left(0,d(A,A_1)\times\delta\right)=\left[\omega_1(a),\omega_1(b)\right]。$$

寻找下一个水平误差校正点，此时以A_1点为球心，由于垂直误差和水平误差的存在，球心半径还需要满足消除垂直误差和水平误差的条件，所以球心半径应取

$$r_2=\min\left\{\frac{\beta_1-\omega_1(a)}{\delta},\frac{\beta_2-\omega_2(b)}{\delta}\right\},$$

同问题一，寻找满足$d(\mathrm{A}_1,\mathrm{A}_2)=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}<r_2$和$F_2=G_2+H_2$最小的水平误差校正点。此时同找出$\mathrm{A}_1$点的判断条件相似，对$\mathrm{A}_2$点进行判断，先判断$\mathrm{A}_2$点是否为可能出现问题的点，如果$\mathrm{A}_2$点为可能出现问题的点且其生成的随机数$9\le\xi\le10$，则$e_2=\min\left(\omega_1(b)+d(\mathrm{A}_1,\mathrm{A}_2)\times\delta,5\right)$，误差为$\omega_2=\left(\omega_1(a)+d(\mathrm{A}_1,\mathrm{A}_2)\times\delta,e_2\right)=\left[\omega_2(a),\omega_2(b)\right]$，且找出校正失败的垂直误差校正点的编号；如果随机生成的整数$1\le\xi\le8$或者垂直误差校正点A_2为正常校正点，则此时误差$\omega_2=\left(d(\mathrm{A}_1,\mathrm{A}_2)\times\delta,0\right)=\left[\omega_2(a),\omega_2(b)\right]$。判断$\mathrm{A}_3$时运用同样的办法。当满足$\max\left\{\omega_n(a)+d(\mathrm{A}_\mathrm{n},\mathrm{B})\times\delta,\omega_n(b)+d(\mathrm{A}_\mathrm{n},\mathrm{B})\times\delta\right\}<\theta$时，即$A_n$为飞行器所经过的最后一个误差校正点，找出该点的编号。

（2）从 A 点出发，先经过水平误差校正点

以 A 点为球心，$r_1=\min\left(\frac{\beta_1}{\delta},\frac{\beta_2}{\delta}\right)$为球半径画一个球，只有在球内垂直误差校正点可以满足条件，即垂直误差校正点 A_1的坐标(x_1,y_1,z_1)应满足式(2)，利用式(3)求出的点 A_1与 B 点的距离$d(\mathrm{A}_1,\mathrm{B})$求出$d_1=d(\mathrm{A}_0,\mathrm{A}_1)+d(\mathrm{A}_1,\mathrm{B})$最小。在此限制条件下求出垂直误差校正点$A_1$的坐标，找出该点在附件中的位置编号。接下来的步骤同先经过垂直误差校正点的步骤相同，最后判断输出最后一个校正点的条件也相同。

5.3.2 问题三的求解

（1）对于附件 1 中的数据，将$\alpha_1=25,\alpha_2=15,\beta_1=20,\beta_2=25,\theta=30,\delta=0.001$代入模型，得到飞行器所经过的校正点分别为[503, 200, 136, 80, 237, 278, 375, 172, 340, 277, 370]，总航迹长度为 112367.5025846904。此时飞行器有两个误差校正失败点，分别为垂直误差校正点

172 和水平误差校正点 340，则其成功到达终点的概率为 96%。结果如下表所示：

表 9 附件 1 进行垂直校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
503	13.38791985	13.38791985	11
200	0.865057806	14.25297766	01
136	14.27111329	13.40605548	11
80	4.286747706	17.69280319	01
237	8.914492276	4.62774457	11
278	17.94221998	22.56996455	01
375	23.48405057	5.541830589	11
172	15.49641641	21.038247	02
340	26.64492891	11.1485125	12
277	17.00237598	28.15088848	01
370	23.01349984	6.011123863	11
612	12.65149785	18.66262171	终点 B

此时飞行器航迹如下图 11 所示：

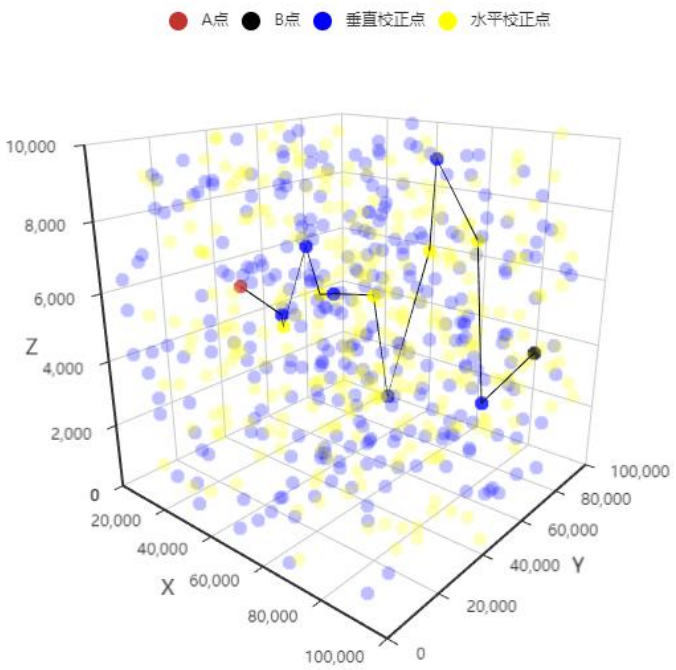
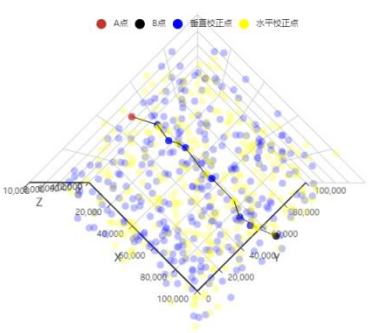
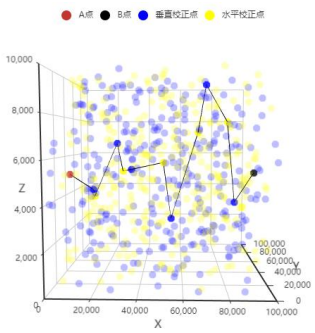


图 11 附件 1 进行垂直校正的飞行路径





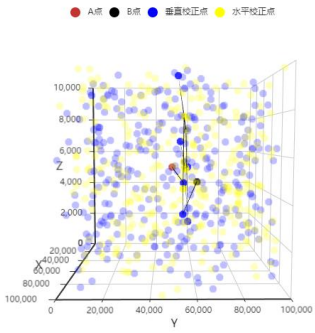


图 11.1　z-y 平面飞行路径　图 11.2　x-z 平面飞行路径　图 11.3　y-z 平面飞行路径

（1）对于附件 2 中的数据，将 $\alpha_1 = 20, \alpha_2 = 10, \beta_1 = 15, \beta_2 = 20, \theta = 20, \delta = 0.001$ 代入模型，得到飞行器所经过的校正点分别为[105, 188, 222, 230, 123, 49, 160, 92, 93, 61]，总航迹长度为 111087.66363688158m，此时飞行器有两个误差校正失败点，均为水平误差校正点，其编号分别为 188 和 92，则其成功到达终点的概率为 96%。结果如下表所示：

表 10　附件 2 进行垂直校正的所有校正点

校正点编号	校正前垂直误差	校正前水平误差	校正点类型
0	0	0	出发点 A
105	13.82090858	13.82090858	11
188	10.66821176	24.48912033	02
222	24.41795579	13.74974404	11
230	11.15990347	24.90964751	01
123	22.12816747	10.968264	11
49	11.79019446	22.75845846	01
160	18.30400023	6.513805771	11
92	5.776163625	12.2899694	02
93	20.26088202	14.4847184	11
61	9.834209702	24.3189281	01
612	12.32153984	22.15574954	终点 B

此时飞行器航迹如下图 12 所示：

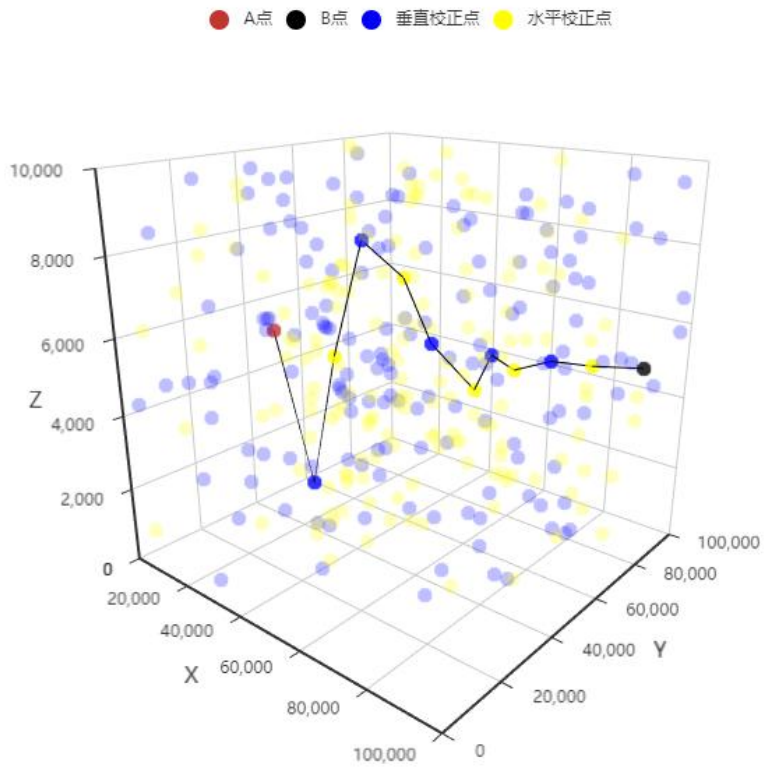


图 12 附件 2 进行垂直校正的飞行路径

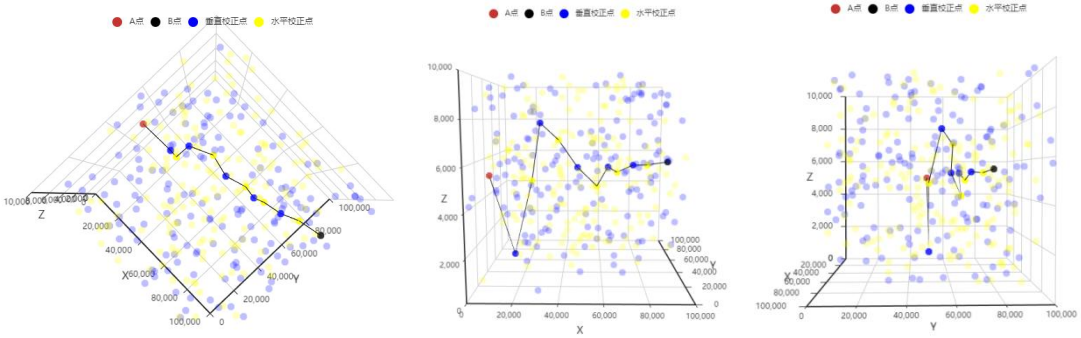


图 12.1 z-y 平面飞行路径 图 12.2 x-z 平面飞行路径 图 12.3 y-z 平面飞行路径

六、模型的评价

6.1 模型的优点

1、改进的 A*算法与贪心算法三维寻路模型在求误差校正点时采用加和求最小值的算法，避免了因为求一步最优解而忽略全局的情况。

2、采用立体几何模型，将空间中三维坐标转换为二维平面上的坐标，简化了运算量，找出了最优航迹。

3、运用计算机生成随机数的方法求飞行器成功到达终点的概率，多次实验，每次返回不同的结果，通过比较可以得出到达终点概率较大的航迹。

4、本文将飞行器假设为简单的质点，不考虑飞行器的形状、质点、大小等，大大简化了运算步骤，减少了工作量，在计算过程中从而达到理想的效果。

6.2 模型的缺点

1、在问题二的计算过程中，最后一个校正点与终点的轨迹应为一段圆弧加上一段直线，而在实际计算过程中直接采取了这两点之间的距离，所以产生了一定的数值误差。

2、本文只考虑了题目中所给出的 8 个约束条件，而在现实生活中飞行器在航迹规划的过程中要考虑的因素很多，如天气、环境、安全因素等，所以在推广此模型时，还需要考虑多方面的因素，以及这些因素的权重。

参考文献

[1] 陶骥华. 无人机航迹规划算法的研究[D].杭州电子科技大学,2017.
[2] 李文广,胡永江,孙世宇,李建增,褚丽娜.无人机任务规划约束空间建模[J].现代防御技术,2019,47(04):7-13.
[3] 聂俊岚,张庆杰,王艳芬.基于加权 Voronoi 图的无人飞行器航迹规划[J].飞行力学,2015,33(04):339-343.
[4] 王升. 基于加权 Voronoi 图的航迹规划算法及可视化研究[D].燕山大学,2015.
[5] 马云红,张恒,齐乐融,贺建良.基于改进 A*算法的三维无人机路径规划[J/OL].电光与控制:1-5[2019-09-22].http://kns.cnki.net/kcms/detail/41.1227.tn.20190624.1645.016.html.
[6] 王闯,董宏丽,谷星澍,李佳慧,陈建玲.改进粒子群算法及其在航迹规划中的应用[J].控制工程,2019,26(08):1466-1471.
[7] 齐小刚,李博,范英盛,刘立芳.多约束下多无人机的任务规划研究综述[J/OL].智能系统学报:1-14[2019-09-22].http://kns.cnki.net/kcms/detail/23.1538.tp.20190409.0932.010.html.
[8] 袁小凯,傅强,侯明利.基于遗传算法的 ADS-B 飞行航迹规划方法研究[J].市场周刊,2019(03):170-171.

附录

注:程序均为最后最优结果的运算程序，在每个问题第一个点的选取是水平校正点还是垂直校正点部分的比较代码此处没放（占用空间太大）

问题一-附件 1 程序（Python）

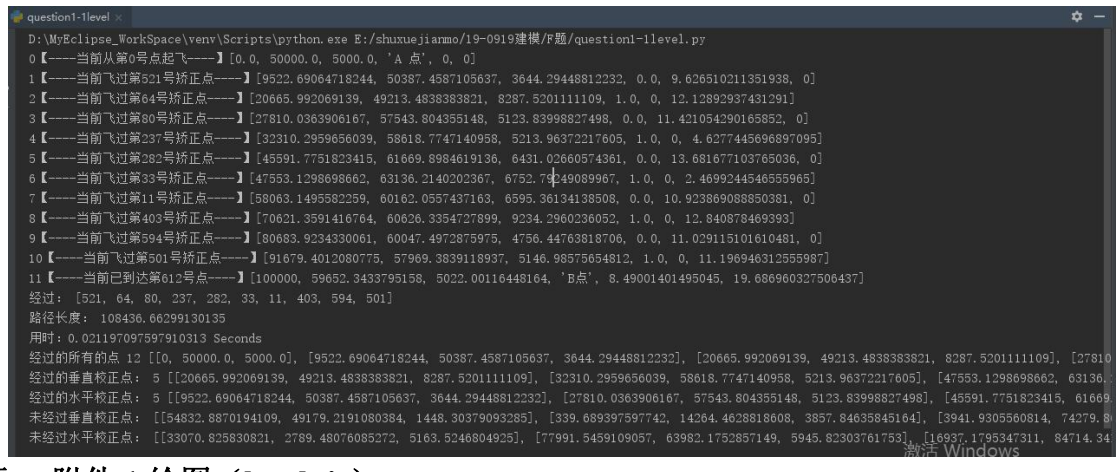
```
import xlrd
import xlwt
import math
import time
# import necessary module
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
# 绘制结果图，仅作测试用，实际为 html+js 绘图
def draw_picture(x,y,z):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # set figure information
    ax.set_title("========result=========")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    # draw the figure, the color is r = read
    ax.plot(x,y,z, c='r')
    plt.show()
#  设置权重
def weight(x,y,a = 1,b = 1):
    return a*x+b*y
def main():
    path_1 = './static_file/dataset1.xlsx'
    exl_1 = xlrd.open_workbook(path_1)
    table_1 = exl_1.sheet_by_name('Sheet1')
    num_rows_1 = table_1.nrows
    num_cols_1 = table_1.ncols
    file_1 = xlwt.Workbook(encoding = 'utf-8')
    #print(num_rows_1)#统计的总行数
    # 1.xlsx
    x = [0.0]
    y = [0.0]
    z = [0.0]
    coordinates = []
    for i in range(num_rows_1):
        coordinate = list(table_1.row_values(i))
        coordinate.append(0)
```

```
            coordinate.append(0)
            coordinates.append(coordinate)
        passed_coordinates = [coordinates[0]] #  记录走过的正确的点
        # print(coordinates)
        j = 0
        mins = []
        min_i = 0
        min_index = []
        meet_coordinate = [] #  满足在球内部的点
        B = [100000,59652.3433795158,5022.0011644816404]
        flag = 0
        min_toB = 0
        pre_toNow_dist = 0
        start_time = time.clock()
        print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
        while True:
            min_value = math.inf
            if j == 0:   #  如果是第一个数
                r = 15 / 0.001    #  球的半径条件
            elif j % 2 == 1:   #  奇数为垂直
                r = min(20 / 0.001,(25 - passed_coordinates[j][4]) / 0.001)
                # print('垂直',(25 - passed_coordinates[j][4]) / 0.001)
                flag = 1
            elif j % 2 == 0:   #  偶数为水平
                r = 15 / 0.001 if (15 / 0.001) < (25 - passed_coordinates[j][4]) / 0.001 else (25 -
passed_coordinates[j][
                    4]) / 0.001
                # print('水平',(25 - passed_coordinates[j][5]) / 0.001)
                flag = 0
            for i in range(1,len(coordinates)):
                #  计算当前点跟上一个走过的点之间的距离
                pre_dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 +
(coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] -
passed_coordinates[j][2])**2)
                #  计算可选点到 B 点的距离
                if pre_dist <= r     and coordinates[i][3] == flag and coordinates[i] not in
passed_coordinates: #  判断是否在球里面
                    meet_coordinate.append(coordinates[i]) #找到在球内部的点就把它加进去
                    #  再去判断选中的点是否是距离 B 点和之前路径和最小的点
                    temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] -
coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                    if min_value > weight(temp_toB,pre_dist):
                        pre_toNow_dist = pre_dist
                        min_value = weight(temp_toB,pre_dist)
```

```
                    min_i = i #  下表
                    min_toB = temp_toB
            else:
                pass
        min_index.append(min_i)
        mins.append(pre_toNow_dist)
        j += 1
        # print( 0 % 2)
        passed_coordinates.append (coordinates[min_i]) #  经过的点多一
        if j % 2 == 1:
            passed_coordinates[j][4] = pre_toNow_dist * 0.001
            # print(j, passed_coordinates[j][4])
        elif j % 2 == 0:
            passed_coordinates[j][5] = pre_toNow_dist * 0.001

        print("{0} 【 ---- 当 前 飞 过 第 {1} 号 矫 正 点 ---- 】 {2}".format(j, min_i,
passed_coordinates[j]))
        # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] -
passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
        if pre_toNow_dist * 0.001   + min_toB * 0.001< 30:
            break
    end_time = time.clock()
    print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 612,
[100000,59652.3433795158,5022.0011644816404,'B 点 ',min_toB * 0.001,pre_toNow_dist *
0.001   + min_toB * 0.001]))
    print('经过：',min_index)
    print('路径长度：',sum(mins) + min_toB)
    print('用时：%s Seconds' % (end_time - start_time))
    # print(passed_coordinates)
    draw_nodes_1 = []
    draw_nodes_0 = []
    draw_nodes = [[0, 50000.0, 5000.0]]
    for i in range(len(passed_coordinates)):
        draw_node_1 = []
        draw_node_0 = []
        x.append(passed_coordinates[i][0])
        y.append(passed_coordinates[i][1])
        z.append(passed_coordinates[i][2])
        if passed_coordinates[i][3] == 1:
            draw_node_1.append(passed_coordinates[i][0])
            draw_node_1.append(passed_coordinates[i][1])
            draw_node_1.append(passed_coordinates[i][2])
            draw_nodes_1.append(draw_node_1)
            draw_nodes.append(draw_node_1)
```

```
            elif passed_coordinates[i][3] == 0:
                draw_node_0.append(passed_coordinates[i][0])
                draw_node_0.append(passed_coordinates[i][1])
                draw_node_0.append(passed_coordinates[i][2])
                draw_nodes_0.append(draw_node_0)
                draw_nodes.append(draw_node_0)
        draw_nodes.append(B)
        print('经过的所有的点', len(draw_nodes), draw_nodes)
        print('经过的垂直校正点：',len(draw_nodes_1), draw_nodes_1)
        print('经过的水平校正点：', len(draw_nodes_0),draw_nodes_0)
        nodes_1 = []   # 其它垂直校正点
        nodes_0 = []   # 其它水平校正点
        for i in range(num_rows_1):
            node_1 = []
            node_0 = []
            if i not in min_index and i != 0 and i != num_rows_1 - 1:
                if table_1.row_values(i)[3] == 1:
                    node_1.append(table_1.row_values(i)[0])
                    node_1.append(table_1.row_values(i)[1])
                    node_1.append(table_1.row_values(i)[2])
                    nodes_1.append(node_1)
                elif table_1.row_values(i)[3] == 0:
                    node_0.append(table_1.row_values(i)[0])
                    node_0.append(table_1.row_values(i)[1])
                    node_0.append(table_1.row_values(i)[2])
                    nodes_0.append(node_0)
        print('未经过垂直校正点：', nodes_1)
        print('未经过水平校正点：', nodes_0)
        # draw_picture(x,y,z)
if __name__ == '__main__':
        main()
```

问题一-附件 1 程序运行结果截图



问题一-附件 1 绘图（html+js）

注：以下所有问题均采用此绘图方式，不过数据点不一样。

```
<!DOCTYPE html>
<html style="height: 100%">
    <head>
        <meta charset="utf-8">
    </head>
    <body style="height: 100%; margin: 0">
        <div id="container" style="height: 100%;margin-bottom:150px;margin-left: 350px;margin-right: 350px"></div>
        <script type="text/javascript" src="../static_file/echarts.js"></script>
        <script type="text/javascript" src="../static_file/echart-gl.min.js"></script>
        <script type="text/javascript">
          var myChart = echarts.init(document.getElementById('container'));
          var title = {
                text: '',
                textAlign:'right'
          };
          var legend = {
                top:'50'
          };
          var toolbox = {
          show: true,
          feature: {
                dataZoom: {
                      yAxisIndex: 'none'
                },
                dataView: {readOnly: false},
                magicType: {type: ['line', 'bar']},
                restore: {},
                saveAsImage: {}
          }
      };
          var xAxis3D={
                type: 'value'
          };
          var yAxis3D={
              type: 'value'
          };
          var zAxis3D = {
              type: 'value'
          };
          var grid3D={};
          var series = [
```

```
{
    type: 'line3D', //设置图表类型为折线图
    //name:'航迹线',
    data:[[数据在代码中生成]] ,
    lineStyle: {
        //symbol:'circle', //去掉折线图中的节点
       // smooth: 0.5,   //true 为平滑曲线，false 为直线
        width: 1,
        color:'black'
    }
},
{
    type: 'scatter3D', //起始点
    name:'A 点',
    data: [[0.0, 50000.0, 5000.0]],
    itemStyle: {
        color:'#c23531',
        width: 2
}
},
 {
    type: 'scatter3D', //终点
    name:'B 点',
    data: [[100000, 59652.3433795158, 5022.00116448164]],
    itemStyle: {
        color:'black',
        width: 2
}
},
{
    type: 'scatter3D', //经过的垂直点
    name:'垂直校正点',
    data:[[数据在代码中生成]] ,
    itemStyle: {
        color:'blue',
        width: 2
}
},
{
    type: 'scatter3D', //经过的水平点
    name:'水平校正点',
    data: [[数据在代码中生成]],
    itemStyle: {
        color:'yellow',
```

```
                    width: 2
                }
                },
                {
                    type: 'scatter3D', //未经过垂直校正点
                    //name:'校正点',
                    data: [[数据在代码中生成]],
                    itemStyle: {
                        // symbol:'circle', //去掉折线图中的节点
                        // smooth: true,  //true 为平滑曲线，false 为直线
                        color:'blue',
                        opacity:0.25,
                        width: 2
                    }
                },
                {
                    type: 'scatter3D', //未经过水平校正点
                    //name:'校正点',
                    data: ,
                    itemStyle: {
                        // symbol:'circle', //去掉折线图中的节点
                        // smooth: true,  //true 为平滑曲线，false 为直线
                        color:'yellow',
                        opacity:0.25,
                        width: 2
                    }
                },
            ];
            var json = {};
            json.title = title;
            json.legend = legend;
            json.toolbox = toolbox;
            json.xAxis3D = xAxis3D;
            json.yAxis3D = yAxis3D;
            json.zAxis3D = zAxis3D;
            json.grid3D = grid3D;
            json.series = series;
            // 为 echarts 对象加载数据
            myChart.setOption(json);
        </script>
    </body>
</html>
```

问题一-附件 2 程序（Python）

```
import xlrd
```

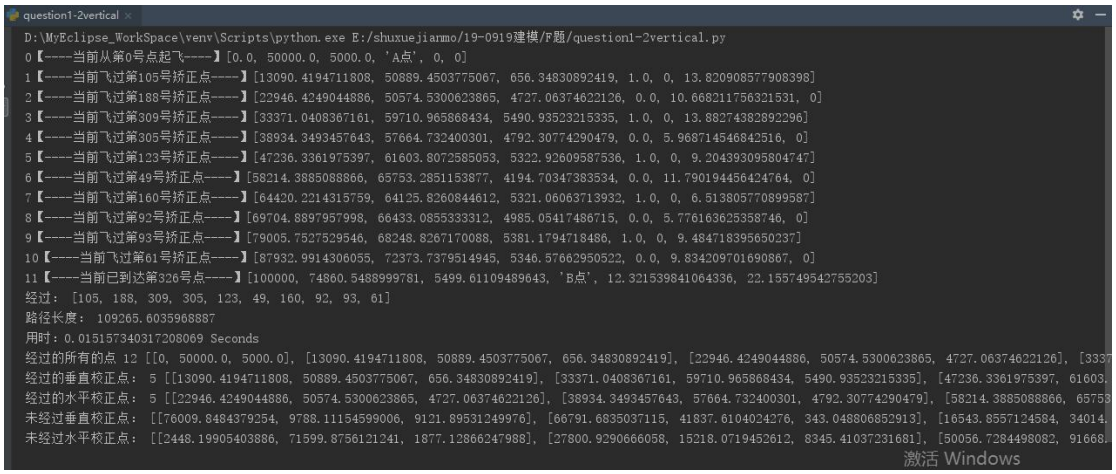
```
import xlwt
import math
import time
import matplotlib.pyplot as plt
# 绘制结果图，仅作测试用，实际为 html+js 绘图
def draw_picture(x,y,z):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # set figure information
    ax.set_title("========result=========")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    # draw the figure, the color is r = read
    ax.plot(x,y,z, c='r')
    plt.show()
#   设置权重
def weight(x,y,a = 1,b = 1):
    return a*x+b*y
def main():
    path_1 = './static_file/dataset2.xlsx'
    exl_1 = xlrd.open_workbook(path_1)
    table_1 = exl_1.sheet_by_name('Sheet1')
    num_rows_1 = table_1.nrows
    num_cols_1 = table_1.ncols
    file_1 = xlwt.Workbook(encoding = 'utf-8')
    #print(num_rows_1)#统计的总行数
    # 1.xlsx
    x = [0.0]
    y = [0.0]
    z = [0.0]
    coordinates = []
    for i in range(num_rows_1):
        coordinate = list(table_1.row_values(i))
        coordinate.append(0)
        coordinate.append(0)
        coordinates.append(coordinate)
    passed_coordinates = [coordinates[0]] #  记录走过的正确的点
    # print(coordinates)
    j = 0
    min_value = math.inf
    mins = []
    min_i = 0
    min_index = []
```

```
    meet_coordinates = [] # 满足在球内部的点
    B = [100000,74860.5488999781,5499.61109489643]
    r = 0
    flag = 1
    temp_toB = 0
    min_toB = 0
    pre_toNow_dist = 0
    count_distance = 0
    start_time = time.clock()
    print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
    while True:
        min_value = math.inf
        if j == 0:  # 如果是第一个数
            r = 15 / 0.001  # 球的半径条件
        elif j % 2 == 1:  # 奇数为水平
            r = 20 / 0.001 if (20 / 0.001) < (25 - passed_coordinates[j][5]) / 0.001 else (25 -
passed_coordinates[j][
                5] ) / 0.001
            flag = 0
        elif j % 2 == 0:  # 偶数为垂直
            r = 15 / 0.001 if (15 / 0.001) < (25 - passed_coordinates[j][4] ) / 0.001 else (25 -
passed_coordinates[j][
                4] ) / 0.001
            flag = 1
        for i in range(1,len(coordinates)):
            # 计算当前点跟上一个走过的点之间的距离
            pre_dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 +
(coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] -
passed_coordinates[j][2])**2)
            # 计算可选点到 B 点的距离
            if pre_dist <= r  and coordinates[i][3] == flag and coordinates[i] not in
passed_coordinates: # 判断是否在球里面
                meet_coordinates.append(coordinates[i]) #找到在球内部的点就把它加进去
                # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] -
coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                if min_value > weight(temp_toB,pre_dist):
                    pre_toNow_dist = pre_dist
                    min_value = weight(temp_toB,pre_dist)
                    min_i = i #  下表
                    min_toB = temp_toB
            else:
                pass
        min_index.append(min_i)
```

```
            mins.append(pre_toNow_dist)
            count_distance += pre_toNow_dist
            j += 1
            passed_coordinates.append (coordinates[min_i]) # 经过的点多一
            if j % 2 == 0:
                passed_coordinates[j][4] = pre_toNow_dist * 0.001
            elif j % 2 == 1:
                passed_coordinates[j][5] = pre_toNow_dist * 0.001
            print("{0} 【 ---- 当 前 飞 过 第 {1} 号 矫 正 点 ---- 】 {2}".format(j, min_i,
passed_coordinates[j]))
            # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] -
passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
            if pre_toNow_dist * 0.001 + min_toB * 0.001< 30:
                break
        end_time = time.clock()
        print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 326,
[100000,74860.5488999781,5499.61109489643,'B 点',min_toB * 0.001,pre_toNow_dist * 0.001
+ min_toB * 0.001]))
        print('经过：',min_index)
        print('路径长度：',sum(mins) + min_toB)
        print('用时：%s Seconds' % (end_time - start_time))
        # print(passed_coordinates)
        draw_nodes_1 = []
        draw_nodes_0 = []
        draw_nodes = [[0,50000.0,5000.0]]
        for i in range(len(passed_coordinates)):
            draw_node_1 = []
            draw_node_0 = []
            x.append(passed_coordinates[i][0] )
            y.append(passed_coordinates[i][1])
            z.append(passed_coordinates[i][2])
            if passed_coordinates[i][3] == 1:
                draw_node_1.append(passed_coordinates[i][0] )
                draw_node_1.append(passed_coordinates[i][1])
                draw_node_1.append(passed_coordinates[i][2])
                draw_nodes_1.append(draw_node_1)
                draw_nodes.append(draw_node_1)
            elif passed_coordinates[i][3] == 0:
                draw_node_0.append(passed_coordinates[i][0])
                draw_node_0.append(passed_coordinates[i][1])
                draw_node_0.append(passed_coordinates[i][2])
                draw_nodes_0.append(draw_node_0)
                draw_nodes.append(draw_node_0)
        draw_nodes.append(B)
```

```
        print('经过的所有的点',len(draw_nodes),draw_nodes)
        print('经过的垂直校正点：',len(draw_nodes_1),draw_nodes_1)
        print('经过的水平校正点：',len(draw_nodes_0),draw_nodes_0)
        nodes_1 = []   # 其它垂直校正点
        nodes_0 = []   # 其它水平校正点
        for i in range(num_rows_1):
            node_1 = []
            node_0 = []
            if i not in min_index and i != 0 and i != num_rows_1 - 1:
                if table_1.row_values(i)[3] == 1:
                    node_1.append(table_1.row_values(i)[0])
                    node_1.append(table_1.row_values(i)[1])
                    node_1.append(table_1.row_values(i)[2])
                    nodes_1.append(node_1)
                elif table_1.row_values(i)[3] == 0:
                    node_0.append(table_1.row_values(i)[0])
                    node_0.append(table_1.row_values(i)[1])
                    node_0.append(table_1.row_values(i)[2])
                    nodes_0.append(node_0)
        print('未经过垂直校正点：',nodes_1)
        print('未经过水平校正点：', nodes_0)
        # draw_picture(x,y,z)
if __name__ == '__main__':
        main()
```

问题一-附件 2 程序运行结果截图



问题二-附件 1 程序（Python）

```
import xlrd
import xlwt
import math
import time
import matplotlib.pyplot as plt
# 绘制结果图，仅作测试用，实际为 html+js 绘图
```

```
def draw_picture(x,y,z):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # set figure information
    ax.set_title("========result=========")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    # draw the figure, the color is r = read
    ax.plot(x,y,z, c='r')
    plt.show()
# 求解两个圆的交点坐标
def intersection(p1, r1, p2, r2):
    x = p1[0]
    y = p1[1]
    R = r1
    a = p2[0]
    b = p2[1]
    S = r2
    d = math.sqrt((abs(a - x)) ** 2 + (abs(b - y)) ** 2)
    if d > (R + S) or d < (abs(R - S)):
        print("Two circles have no intersection")
        return [None]
    elif d == 0 and R == S:
        print("Two circles have same center!")
        return [None]
    else:
        A = (R ** 2 - S ** 2 + d ** 2) / (2 * d)
        h = math.sqrt(R ** 2 - A ** 2)
        x2 = x + A * (a - x) / d

        y2 = y + A * (b - y) / d
        x3 = round(x2 - h * (b - y) / d, 10)
        y3 = round(y2 + h * (a - x) / d, 10)
        x4 = round(x2 + h * (b - y) / d, 10)
        y4 = round(y2 - h * (a - x) / d, 10)
        # print(x3, y3)
        # print(x4, y4)
        c1 = [x3, y3]
        c2 = [x4, y4]
        return c1, c2
# 求弧长加直飞距离
def distance(pre,rear,candidate): # 前俩是之前选过的点，最后是待选的点
    right_candidate = 1
```

```
    # 求直飞的夹角
    alpha = math.atan(abs((rear[1] - pre[1])/(rear[0] - pre[0])))
    # 求圆心坐标
    o = [rear[0]+((candidate[0] - rear[0]) / abs(candidate[0] - rear[0])) * 200 *
         math.sin(alpha),rear[1]+((candidate[1] - rear[1]) / abs(candidate[1] - rear[1])) *200 *
math.cos(alpha)]
    # 圆心到到下一个待选点 candidate 的直线距离
    o_candidate_length = math.sqrt((candidate[0] - o[0]) ** 2 + (candidate[1] - o[1]) ** 2)
    # 以新的圆弧上的点到下一个待选点 candidate 的直线距离
    stright_length = math.sqrt(abs(o_candidate_length**2 - 200**2))
    # 求新的圆弧上的点坐标
    new_points = intersection(o,200,candidate,stright_length)
    new_point = []
    # 如果有两个交点
    if len(new_points)  == 2:
        if candidate[1] > rear[1]:
            new_point = new_points[0] if new_points[0][1] < new_points[1][1] else
new_points[1]
        elif candidate[1] < rear[1]:
            new_point = new_points[1] if new_points[0][1] < new_points[1][1] else
new_points[0]
        pass
    # 如果有一个或者没有，相切
    else:
        right_candidate = 0
    # print(new_points)
    # print(new_point)
    # 相交点到上一个走过的点的直线距离
    level_length = math.sqrt((new_point[0] - rear[0]) ** 2 + (new_point[1] - rear[1]) ** 2)
    theta =  math.acos((2 * 200 ** 2 - level_length) / (2 * 200 **2))
    min_length = math.sqrt((theta * 200  + stright_length) ** 2 + (rear[2] - candidate[2]) ** 2)
    # = math.sqrt(h ** 2 + level_length ** 2) + math.sqrt(h ** 2 + stright_length ** 2)
    # print(stright_length)
    # print(alpha)
    # print(o)
    return min_length,right_candidate
# 权重求和
def weight(x,y,a = 1,b = 1):
    return a*x+b*y
# 程序运行主函数
def main():
    path_1 = './static_file/dataset1.xlsx'
    exl_1 = xlrd.open_workbook(path_1)
    table_1 = exl_1.sheet_by_name('Sheet1')
```

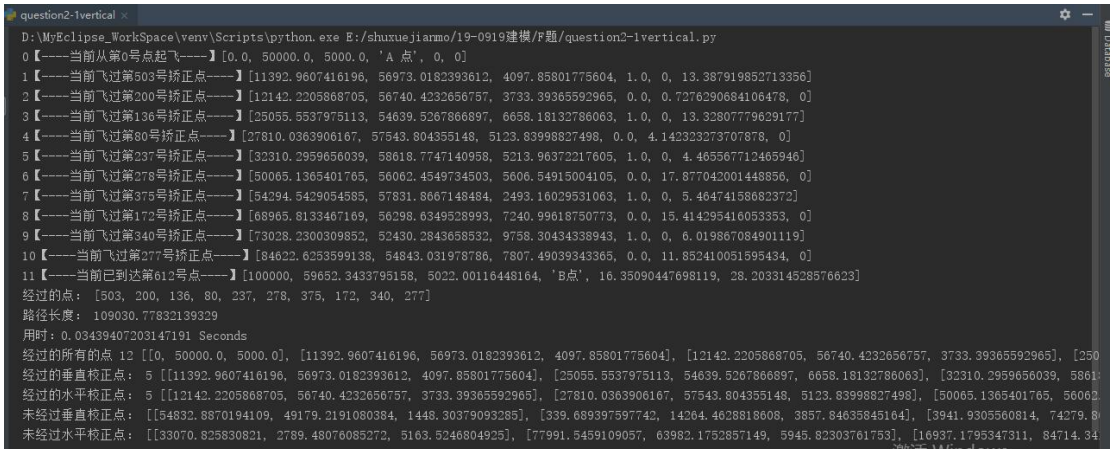
```
    num_rows_1 = table_1.nrows
    num_cols_1 = table_1.ncols
    file_1 = xlwt.Workbook(encoding = 'utf-8')
    #print(num_rows_1)#统计的总行数
    # 1.xlsx
    x = [0.0]
    y = [0.0]
    z = [0.0]
    coordinates = [] #  表格中所有的点
    for i in range(num_rows_1):
        coordinate = list(table_1.row_values(i))
        coordinate.append(0)
        coordinate.append(0)
        coordinates.append(coordinate)
    passed_coordinates = [coordinates[0]] #  记录走过的正确的点
    # print(distance(coordinates[0],coordinates[1],coordinates[2]))
    j = 0
    mins = [] #  走过的路径长度
    min_i = 0 #  记录要走点的下标
    min_index = [] #  记录所有走的下标
    meet_coordinates = [] #  满足在球内部的点
    B = [100000,59652.3433795158,5022.0011644816404]
    r = 0
    flag = 1
    count_distance = 0
    start_time = time.clock()
    print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
    while True:
        pre_toNow_dist = 0
        min_toB = 0
        min_value = math.inf
        if j == 0:   #  如果是第一个数
            r = 15 / 0.001   #  球的半径条件
        elif j % 2 == 1:   #  奇数为水平
            r = 20 / 0.001 if (20 / 0.001) < (25 - passed_coordinates[j ][5] ) / 0.001 else (25 -
passed_coordinates[j][
                    5] ) / 0.001
            flag = 0
        elif j % 2 == 0:   #  偶数为垂直
            r = 15 / 0.001 if (15 / 0.001) < (25 - passed_coordinates[j ][4] ) / 0.001 else (25 -
passed_coordinates[j ][
                    4] ) / 0.001
            flag = 1
        for i in range(1,len(coordinates)):
```

```
            # 计算当前点跟上一个走过的点之间的距离
            dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 + (coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] - passed_coordinates[j][2])**2)
            # 计算可选点到 B 点的距离
            if len(passed_coordinates) == 1: # 刚从第一个点出发
                if dist <= r  and coordinates[i][3] == flag : # 判断是否在球里面 并 满足相应矫正点
                    meet_coordinates.append(coordinates[i]) #找到在球内部的点就把它加进去
                    # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                    temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] - coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                    if min_value > temp_toB+ dist:
                        pre_toNow_dist = dist # 记录经过的点到现在点的距离
                        min_value = temp_toB + dist # 记录最小的距离和
                        min_i = i # 记录最小距离下的下标
                        min_toB = temp_toB
            else: # 走过点的个数>=1
                if dist <= r  and coordinates[i][3] == flag and coordinates[i] not in passed_coordinates:
                    meet_coordinates.append(coordinates[i])  # 找到在球内部的点就把它加进去
                    # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                    temp_toB = math.sqrt(
                        (B[0] - coordinates[i][0]) ** 2 + (B[1] - coordinates[i][1]) ** 2 + (B[2] - coordinates[i][2]) ** 2)
                    arc_distance,right_candidate = distance(passed_coordinates[j-1],passed_coordinates[j],coordinates[i])
                    # print(arc_distance,right_candidate)
                    if min_value > weight(temp_toB,arc_distance) and right_candidate == 1:# 满足距离最小 且 有两个交点
                        pre_toNow_dist = arc_distance
                        min_value = weight(temp_toB,arc_distance)
                        min_toB = temp_toB
                        min_i = i  # 下标
        min_index.append(min_i)
        # print(min_index)
        mins.append(pre_toNow_dist) # 走的总路径
        j += 1
        passed_coordinates.append (coordinates[min_i]) # 经过的点多一
        if j % 2 == 0:
            passed_coordinates[j][4] = pre_toNow_dist * 0.001
        elif j % 2 == 1:
```

```
                passed_coordinates[j][5] = pre_toNow_dist * 0.001
            print("{0} 【 ---- 当 前 飞 过 第 {1} 号 矫 正 点 ---- 】 {2}".format(j,min_i, passed_coordinates[j]))
            # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] - passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
            if pre_toNow_dist * 0.001    + min_toB * 0.001 < 30:
                # print("{0} , {1}".format(pre_toNow_dist,min_toB))
                break
        end_time = time.clock()
        print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 612, [100000,59652.3433795158,5022.0011644816404,'B 点 ',min_toB * 0.001,pre_toNow_dist * 0.001    + min_toB * 0.001]))
        print('经过的点：',min_index)
        print('路径长度：',sum(mins)+min_toB)
        print('用时：%s Seconds' % (end_time - start_time))
        # print(passed_coordinates)
        draw_nodes_1 = []
        draw_nodes_0 = []
        draw_nodes = [[0,50000.0,5000.0]]
        for i in range(len(passed_coordinates)):
            draw_node_1 = []
            draw_node_0 = []
            x.append(passed_coordinates[i][0] )
            y.append(passed_coordinates[i][1])
            z.append(passed_coordinates[i][2])
            if passed_coordinates[i][3] == 1:
                draw_node_1.append(passed_coordinates[i][0] )
                draw_node_1.append(passed_coordinates[i][1])
                draw_node_1.append(passed_coordinates[i][2])
                draw_nodes_1.append(draw_node_1)
                draw_nodes.append(draw_node_1)
            elif passed_coordinates[i][3] == 0:
                draw_node_0.append(passed_coordinates[i][0])
                draw_node_0.append(passed_coordinates[i][1])
                draw_node_0.append(passed_coordinates[i][2])
                draw_nodes_0.append(draw_node_0)
                draw_nodes.append(draw_node_0)
        draw_nodes.append(B)
        print('经过的所有的点',len(draw_nodes),draw_nodes)
        print('经过的垂直校正点：',len(draw_nodes_1),draw_nodes_1)
        print('经过的水平校正点：',len(draw_nodes_0),draw_nodes_0)
        nodes_1 = []   # 其它垂直校正点
        nodes_0 = []   # 其它水平校正点
        for i in range(num_rows_1):
```

```
            node_1 = []
            node_0 = []
            if i not in min_index and i != 0 and i != num_rows_1 - 1:
                if table_1.row_values(i)[3] == 1:
                    node_1.append(table_1.row_values(i)[0])
                    node_1.append(table_1.row_values(i)[1])
                    node_1.append(table_1.row_values(i)[2])
                    nodes_1.append(node_1)
                elif table_1.row_values(i)[3] == 0:
                    node_0.append(table_1.row_values(i)[0])
                    node_0.append(table_1.row_values(i)[1])
                    node_0.append(table_1.row_values(i)[2])
                    nodes_0.append(node_0)
        print('未经过垂直校正点：',nodes_1)
        print('未经过水平校正点：', nodes_0)
        x.append(100000)
        y.append(59652.3433795158)
        z.append(5022.0011644816404)
        # draw_picture(x,y,z)
if __name__ == '__main__':
        main()
```

问题二-附件 1 程序运行结果截图



问题二-附件 2 程序（Python）

```
import xlrd
import xlwt
import math
import time
import matplotlib.pyplot as plt
# 绘制结果图，仅作测试用，实际为 html+js 绘图
def draw_picture(x,y,z):
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        # set figure information
```

```
        ax.set_title("========result=========")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_zlabel("z")
        # draw the figure, the color is r = read
        ax.plot(x,y,z, c='r')
        plt.show()
# 求解两个圆的交点坐标
def intersection(p1, r1, p2, r2):
        x = p1[0]
        y = p1[1]
        R = r1
        a = p2[0]
        b = p2[1]
        S = r2
        d = math.sqrt((abs(a - x)) ** 2 + (abs(b - y)) ** 2)
        if d > (R + S) or d < (abs(R - S)):
                print("Two circles have no intersection")
                return [None]
        elif d == 0 and R == S:
                print("Two circles have same center!")
                return [None]
        else:
                A = (R ** 2 - S ** 2 + d ** 2) / (2 * d)
                h = math.sqrt(R ** 2 - A ** 2)
                x2 = x + A * (a - x) / d
                y2 = y + A * (b - y) / d
                x3 = round(x2 - h * (b - y) / d, 10)
                y3 = round(y2 + h * (a - x) / d, 10)
                x4 = round(x2 + h * (b - y) / d, 10)
                y4 = round(y2 - h * (a - x) / d, 10)
                # print(x3, y3)
                # print(x4, y4)
                c1 = [x3, y3]
                c2 = [x4, y4]
                return c1, c2
# 求弧长加直飞距离
def distance(pre,rear,candidate): # 前俩是之前选过的点，最后是待选的点
        right_candidate = 1
        # 求直飞的夹角
        alpha = math.atan(abs((rear[1] - pre[1])/(rear[0] - pre[0])))
        # 求圆心坐标
        o = [rear[0]+((candidate[0] - rear[0]) / abs(candidate[0] - rear[0])) * 200 *
                math.sin(alpha),rear[1]+((candidate[1] - rear[1]) / abs(candidate[1] - rear[1])) *200 *
```

```
math.cos(alpha)]
    # 圆心到到下一个待选点 candidate 的直线距离
    o_candidate_length = math.sqrt((candidate[0] - o[0]) ** 2 + (candidate[1] - o[1]) ** 2)
    # 以新的圆弧上的点到下一个待选点 candidate 的直线距离
    stright_length = math.sqrt(abs(o_candidate_length**2 - 200**2))
    # 求新的圆弧上的点坐标
    new_points = intersection(o,200,candidate,stright_length)
    new_point = []
    # 如果有两个交点
    if len(new_points)  == 2:
        if candidate[1] > rear[1]:
            new_point = new_points[0] if new_points[0][1] < new_points[1][1] else
new_points[1]
        elif candidate[1] < rear[1]:
            new_point = new_points[1] if new_points[0][1] < new_points[1][1] else
new_points[0]
        pass
    # 如果有一个或者没有，相切
    else:
        right_candidate = 0
    # print(new_points)
    # print(new_point)
    # 相交点到上一个走过的点的直线距离
    level_length = math.sqrt((new_point[0] - rear[0]) ** 2 + (new_point[1] - rear[1]) ** 2)
    theta =  math.acos((2 * 200 ** 2 - level_length) / (2 * 200 **2))
    min_length = math.sqrt((theta * 200  + stright_length) ** 2 + (rear[2] - candidate[2]) ** 2)
    # = math.sqrt(h ** 2 + level_length ** 2) + math.sqrt(h ** 2 + stright_length ** 2)
    # print(stright_length)
    # print(alpha)
    # print(o)
    return min_length,right_candidate

def weight(x,y,a = 1,b = 1):
    return a*x+b*y

def main():
    path_1 = './static_file/dataset2.xlsx'
    exl_1 = xlrd.open_workbook(path_1)
    table_1 = exl_1.sheet_by_name('Sheet1')
    num_rows_1 = table_1.nrows
    num_cols_1 = table_1.ncols
    file_1 = xlwt.Workbook(encoding = 'utf-8')
    #print(num_rows_1)#统计的总行数
    # 1.xlsx
```

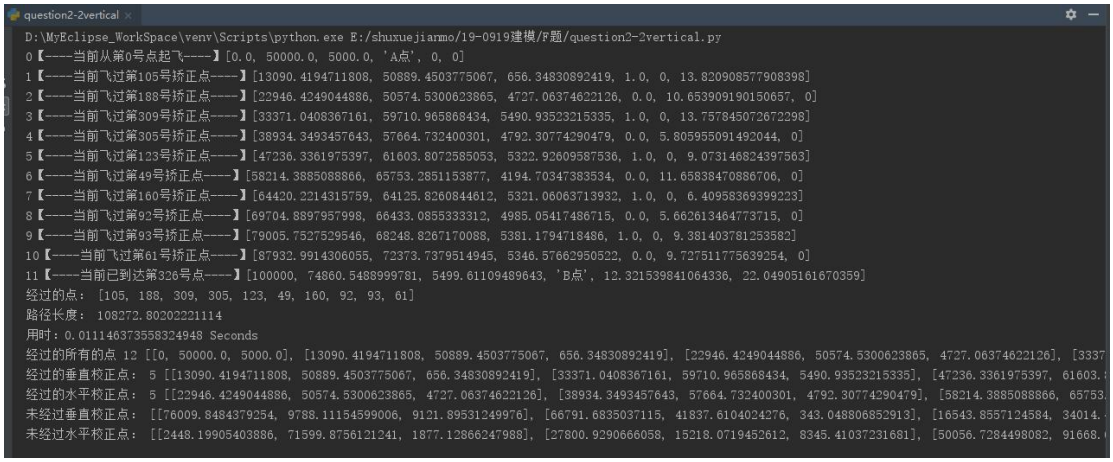
```
    x = [0.0]
    y = [0.0]
    z = [0.0]

    coordinates = [] #  表格中所有的点
    for i in range(num_rows_1):
        coordinate = list(table_1.row_values(i))
        coordinate.append(0)
        coordinate.append(0)
        coordinates.append(coordinate)
    passed_coordinates = [coordinates[0]] #  记录走过的正确的点
    # print(distance(coordinates[0],coordinates[1],coordinates[2]))
    j = 0
    mins = [] #  走过的路径长度
    min_i = 0 #  记录要走点的下标
    min_index = [] #  记录所有走的下标
    meet_coordinates = [] #  满足在球内部的点
    B = [100000, 74860.5488999781, 5499.61109489643]
    r = 0
    flag = 1
    count_distance = 0
    start_time = time.clock()
    print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
    while True:
        pre_toNow_dist = 0
        min_toB = 0
        min_value = math.inf
        if j == 0:   #  如果是第一个数
            r = 15 / 0.001   #  球的半径条件
        elif j % 2 == 1:   #  奇数为水平
            r = 20 / 0.001 if (20 / 0.001) < (25 - passed_coordinates[j][5] ) / 0.001 else (25 -
passed_coordinates[j][
                5] ) / 0.001
            flag = 0
        elif j % 2 == 0:   #  偶数为垂直
            r = 15 / 0.001 if (15 / 0.001) < (25 - passed_coordinates[j][4] ) / 0.001 else (25 -
passed_coordinates[j][
                4] ) / 0.001
            flag = 1
        for i in range(1,len(coordinates)):
            #  计算当前点跟上一个走过的点之间的距离
            dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 +
(coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] -
passed_coordinates[j][2])**2)
```

```
            # 计算可选点到 B 点的距离
            if len(passed_coordinates) == 1: # 刚从第一个点出发
                if dist <= r    and coordinates[i][3] == flag : # 判断是否在球里面 并 满足相应矫正点
                    meet_coordinates.append(coordinates[i]) #找到在球内部的点就把它加进去
                    # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                    temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] - coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                    if min_value > temp_toB+ dist:
                        pre_toNow_dist = dist # 记录经过的点到现在点的距离
                        min_value = temp_toB + dist # 记录最小的距离和
                        min_i = i # 记录最小距离下的下标
                        min_toB = temp_toB
            else: # 走过点的个数>=1
                if dist <= r    and coordinates[i][3] == flag and coordinates[i] not in passed_coordinates:
                    meet_coordinates.append(coordinates[i])  # 找到在球内部的点就把它加进去
                    # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                    temp_toB = math.sqrt(
                        (B[0] - coordinates[i][0]) ** 2 + (B[1] - coordinates[i][1]) ** 2 + (B[2] - coordinates[i][2]) ** 2)
                    arc_distance,right_candidate = distance(passed_coordinates[j-1],passed_coordinates[j],coordinates[i])
                    # print(arc_distance,right_candidate)
                    if min_value > weight(temp_toB,arc_distance) and right_candidate == 1:# 满足距离最小 且 有两个交点
                        pre_toNow_dist = arc_distance
                        min_value = weight(temp_toB,arc_distance)
                        min_toB = temp_toB
                        min_i = i  # 下标
        min_index.append(min_i)
        # print(min_index)
        mins.append(pre_toNow_dist) # 走的总路径
        j += 1
        passed_coordinates.append (coordinates[min_i]) # 经过的点多一
        if j % 2 == 0:
            passed_coordinates[j][4] = pre_toNow_dist * 0.001
        elif j % 2 == 1:
            passed_coordinates[j][5] = pre_toNow_dist * 0.001
        print("{0} 【 ---- 当 前 飞 过 第 {1} 号 矫 正 点 ---- 】 {2}".format(j,min_i, passed_coordinates[j]))
        # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] -
```

```
passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
            if pre_toNow_dist * 0.001    + min_toB * 0.001 < 30:
                # print("{0} , {1}".format(pre_toNow_dist,min_toB))
                break
        end_time = time.clock()
        print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 326, [100000,
74860.5488999781, 5499.61109489643,'B 点 ',min_toB * 0.001,pre_toNow_dist * 0.001    +
min_toB * 0.001]))
        print('经过的点：',min_index)
        print('路径长度：',sum(mins)+min_toB)
        print('用时：%s Seconds' % (end_time - start_time))
        # print(passed_coordinates)
        draw_nodes_1 = []
        draw_nodes_0 = []
        draw_nodes = [[0,50000.0,5000.0]]
        for i in range(len(passed_coordinates)):
            draw_node_1 = []
            draw_node_0 = []
            x.append(passed_coordinates[i][0] )
            y.append(passed_coordinates[i][1])
            z.append(passed_coordinates[i][2])
            if passed_coordinates[i][3] == 1:
                draw_node_1.append(passed_coordinates[i][0] )
                draw_node_1.append(passed_coordinates[i][1])
                draw_node_1.append(passed_coordinates[i][2])
                draw_nodes_1.append(draw_node_1)
                draw_nodes.append(draw_node_1)
            elif passed_coordinates[i][3] == 0:
                draw_node_0.append(passed_coordinates[i][0])
                draw_node_0.append(passed_coordinates[i][1])
                draw_node_0.append(passed_coordinates[i][2])
                draw_nodes_0.append(draw_node_0)
                draw_nodes.append(draw_node_0)
        draw_nodes.append(B)
        print('经过的所有的点',len(draw_nodes),draw_nodes)
        print('经过的垂直校正点：',len(draw_nodes_1),draw_nodes_1)
        print('经过的水平校正点：',len(draw_nodes_0),draw_nodes_0)
        nodes_1 = []   # 其它垂直校正点
        nodes_0 = []   # 其它水平校正点
        for i in range(num_rows_1):
            node_1 = []
            node_0 = []
            if i not in min_index and i != 0 and i != num_rows_1 - 1:
                if table_1.row_values(i)[3] == 1:
```

```
                node_1.append(table_1.row_values(i)[0])
                node_1.append(table_1.row_values(i)[1])
                node_1.append(table_1.row_values(i)[2])
                nodes_1.append(node_1)
            elif table_1.row_values(i)[3] == 0:
                node_0.append(table_1.row_values(i)[0])
                node_0.append(table_1.row_values(i)[1])
                node_0.append(table_1.row_values(i)[2])
                nodes_0.append(node_0)
    print('未经过垂直校正点：',nodes_1)
    print('未经过水平校正点：', nodes_0)
    x.append(100000)
    y.append(59652.3433795158)
    z.append(5022.0011644816404)
    # draw_picture(x,y,z)
if __name__ == '__main__':
    main()
```

问题二-附件 2 程序运行结果截图



问题三-附件 1 程序（Python）

```
import xlrd
import xlwt
import math
import time
import random
import matplotlib.pyplot as plt
#  问题点的判断
def judge_candidate():
    num = random.randint(1, 10)
    if num == 9 or num == 10:
        return 1
    else:
        return 0
judge_candidate()
```

```
# 绘制结果图，仅作测试用，实际为 html+js 绘图
def draw_picture(x,y,z):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # set figure information
    ax.set_title("========result=========")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    # draw the figure, the color is r = read
    ax.plot(x,y,z, c='r')
    plt.show()
#   设置权重
def weight(x,y,a = 1,b = 1):
    return a*x+b*y
# 程序主函数
def main():
    path_1 = './static_file/dataset4.xlsx'
    exl_1 = xlrd.open_workbook(path_1)
    table_1 = exl_1.sheet_by_name('Sheet1')
    num_rows_1 = table_1.nrows
    num_cols_1 = table_1.ncols
    file_1 = xlwt.Workbook(encoding = 'utf-8')
    #print(num_rows_1)#统计的总行数
    # 1.xlsx
    x = [0.0]
    y = [0.0]
    z = [0.0]
    coordinates = []
    for i in range(num_rows_1):
        coordinate = table_1.row_values(i)[:-1]
        # print(type(table_1.row_values(i)[3]))
        coordinate.append(0)
        coordinate.append(0)
        coordinate.append(table_1.row_values(i)[-1])
        coordinates.append(coordinate)
    passed_coordinates = [coordinates[0]] #  记录走过的正确的点
      # print(coordinates)
    j = 0
    min_value = math.inf
    mins = []
    min_i = 0
    min_index = []
    meet_coordinate = [] #  满足在球内部的点
```

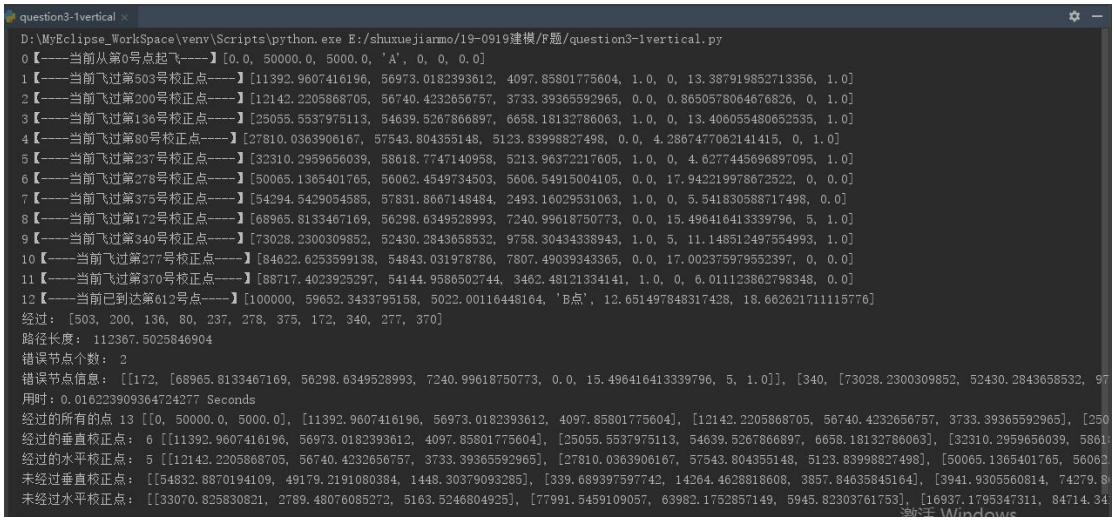
```
    wrong_candidates = []
    B = [100000,59652.3433795158,5022.0011644816404]
    r = 0
    flag = 1
    pre_toNow_dist = 0
    min_toB = 0
    start_time = time.clock()
    print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
    while True:
        min_value = math.inf
        # min_i    = 0
        if j == 0:   # 如果是第一个数
            r = 15 / 0.001    # 球的半径条件
        elif j % 2 == 1:   # 奇数为水平
            r = (20 - passed_coordinates[j][4])/ 0.001 if (20 - passed_coordinates[j][4]) / 0.001
< (25 - passed_coordinates[j][5]) / 0.001 else (25 - passed_coordinates[j][
                5]) / 0.001
            flag = 0
        elif j % 2 == 0:   # 偶数为垂直
            r = (15 - passed_coordinates[j][5])/ 0.001 if ((15 - passed_coordinates[j][5])/ 0.001)
< (25 - passed_coordinates[j][4]) / 0.001 else (25 - passed_coordinates[j][
                4]) / 0.001
            flag = 1
        for i in range(1,len(coordinates)):
            # 计算当前点跟上一个走过的点之间的距离
            pre_dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 +
(coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] -
passed_coordinates[j][2])**2)
            # 计算可选点到 B 点的距离
            if pre_dist <= r    and coordinates[i][3] == flag   : # 判断是否在球里面
                # print(i)
                meet_coordinate.append(coordinates[i]) #找到在球内部的点就把它加进去
                # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] -
coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                if min_value > weight(temp_toB,pre_dist):
                    # 加一个判断条件
                    pre_toNow_dist = pre_dist
                    min_value = weight(temp_toB,pre_dist)
                    min_i = i #  下表
                    min_toB = temp_toB
            else:
                pass
        min_index.append(min_i)
```

```
            mins.append(pre_toNow_dist)
            j += 1
            passed_coordinates.append (coordinates[min_i]) # 经过的点多一
            if j == 1 :
                if judge_candidate() and coordinates[min_i][6]:
                    wrong_candidates.append([min_i,coordinates[min_i]])
                    passed_coordinates[j][4] = min(pre_toNow_dist * 0.001,5)
                    passed_coordinates[j][5] = pre_toNow_dist * 0.001
                else:
                    passed_coordinates[j][5] = pre_toNow_dist * 0.001
                    passed_coordinates[j][4] = 0
            elif j % 2 == 0:
                if judge_candidate() and coordinates[min_i][6]:
                    wrong_candidates.append([min_i, coordinates[min_i]])
                    passed_coordinates[j][5] = min(passed_coordinates[j - 1][5] +
pre_toNow_dist * 0.001,5)
                    passed_coordinates[j][4] = passed_coordinates[j - 1][4] + pre_toNow_dist *
0.001
                else:
                    passed_coordinates[j][4] = passed_coordinates[j - 1][4] + pre_toNow_dist *
0.001
                    passed_coordinates[j][5] = 0
            elif j % 2 == 1 :
                if judge_candidate() and coordinates[min_i][6]:
                    wrong_candidates.append([min_i, coordinates[min_i]])
                    passed_coordinates[j][4] = min(passed_coordinates[j - 1][4] +
pre_toNow_dist * 0.001, 5)
                    passed_coordinates[j][5] = passed_coordinates[j - 1][5] + pre_toNow_dist *
0.001
                else:
                    passed_coordinates[j][5] = passed_coordinates[j - 1][5] + pre_toNow_dist *
0.001
                    passed_coordinates[j][4] = 0

            print("{0} 【 ---- 当 前 飞 过 第 {1} 号 校 正 点 ---- 】 {2}".format(j, min_i,
passed_coordinates[j]))
            # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] -
passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
            if max(passed_coordinates[j][5],passed_coordinates[j][4])  + min_toB * 0.001< 30:
                break
        end_time = time.clock()
        print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 612,
[100000,59652.3433795158,5022.0011644816404,'B 点 ',min_toB * 0.001,pre_toNow_dist *
0.001  + min_toB * 0.001]))
```

```
print('经过：',min_index)
print('路径长度：',sum(mins) + min_toB)
print('错误节点个数：', len(wrong_candidates))
print('错误节点信息：', wrong_candidates)
print('用时：%s Seconds' % (end_time - start_time))
# print(passed_coordinates)
draw_nodes_1 = []
draw_nodes_0 = []
draw_nodes = [[0, 50000.0, 5000.0]]
for i in range(len(passed_coordinates)):
    draw_node_1 = []
    draw_node_0 = []
    x.append(passed_coordinates[i][0])
    y.append(passed_coordinates[i][1])
    z.append(passed_coordinates[i][2])
    if passed_coordinates[i][3] == 1:
        draw_node_1.append(passed_coordinates[i][0])
        draw_node_1.append(passed_coordinates[i][1])
        draw_node_1.append(passed_coordinates[i][2])
        draw_nodes_1.append(draw_node_1)
        draw_nodes.append(draw_node_1)
    elif passed_coordinates[i][3] == 0:
        draw_node_0.append(passed_coordinates[i][0])
        draw_node_0.append(passed_coordinates[i][1])
        draw_node_0.append(passed_coordinates[i][2])
        draw_nodes_0.append(draw_node_0)
        draw_nodes.append(draw_node_0)
draw_nodes.append([100000, 59652.3433795158, 5022.0011644816404])
print('经过的所有的点', len(draw_nodes), draw_nodes)
print('经过的垂直校正点：',len(draw_nodes_1), draw_nodes_1)
print('经过的水平校正点：', len(draw_nodes_0),draw_nodes_0)
nodes_1 = []  # 其它垂直校正点
nodes_0 = []  # 其它水平校正点
for i in range(num_rows_1):
    node_1 = []
    node_0 = []
    if i not in min_index and i != 0 and i != num_rows_1 - 1:
        if table_1.row_values(i)[3] == 1:
            node_1.append(table_1.row_values(i)[0])
            node_1.append(table_1.row_values(i)[1])
            node_1.append(table_1.row_values(i)[2])
            nodes_1.append(node_1)
        elif table_1.row_values(i)[3] == 0:
            node_0.append(table_1.row_values(i)[0])
```

```
                node_0.append(table_1.row_values(i)[1])
                node_0.append(table_1.row_values(i)[2])
                nodes_0.append(node_0)
        print('未经过垂直校正点：', nodes_1)
        print('未经过水平校正点：', nodes_0)
        # draw_picture(x,y,z)
if __name__ == '__main__':
        main()
```

问题三-附件 1 程序运行结果截图



问题三-附件 2 程序（Python）

```
Xxx123
import xlrd
import xlwt
import math
import time
import random
import matplotlib.pyplot as plt
#  问题点的判断
def judge_candidate():
        num = random.randint(1, 10)
        if num == 9 or num == 10:
                return 1
        else:
                return 0
judge_candidate()
#  绘制结果图，仅作测试用，实际为 html+js 绘图
def draw_picture(x,y,z):
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        # set figure information
```

```
        ax.set_title("========result=========")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_zlabel("z")
        # draw the figure, the color is r = read
        ax.plot(x,y,z, c='r')
        plt.show()
#   设置权重
def weight(x,y,a = 1,b = 1):
        return a*x+b*y
#  程序主函数
def main():
        path_1 = './static_file/dataset5.xlsx'
        exl_1 = xlrd.open_workbook(path_1)
        table_1 = exl_1.sheet_by_name('Sheet1')
        num_rows_1 = table_1.nrows
        num_cols_1 = table_1.ncols
        file_1 = xlwt.Workbook(encoding = 'utf-8')
        #print(num_rows_1)#统计的总行数
        # 1.xlsx
        x = [0.0]
        y = [0.0]
        z = [0.0]
        coordinates = []
        for i in range(num_rows_1):
                coordinate = table_1.row_values(i)[:-1]
                # print(type(table_1.row_values(i)[3]))
                coordinate.append(0)
                coordinate.append(0)
                coordinate.append(table_1.row_values(i)[-1])
                coordinates.append(coordinate)
        passed_coordinates = [coordinates[0]] #  记录走过的正确的点
          # print(coordinates)
        j = 0
        min_value = math.inf
        mins = []
        min_i = 0
        min_index = []
        meet_coordinate = [] #  满足在球内部的点
        wrong_candidates = []
        B = [100000, 74860.5488999781, 5499.61109489643]
        r = 0
        flag = 1
        pre_toNow_dist = 0
```

```
    min_toB = 0
    start_time = time.clock()
    print("{0}【----当前从第{1}号点起飞----】{2}".format(0, 0, passed_coordinates[0]))
    while True:
        min_value = math.inf
        # min_i    = 0
        if j == 0:   # 如果是第一个数
            r = 15 / 0.001    # 球的半径条件
        elif j % 2 == 1:   # 奇数为水平
            r = (20 - passed_coordinates[j][4])/ 0.001 if (20 - passed_coordinates[j][4]) / 0.001
< (25 - passed_coordinates[j][5]) / 0.001 else (25 - passed_coordinates[j][
                5]) / 0.001
            flag = 0
        elif j % 2 == 0:   # 偶数为垂直
            r = (15 - passed_coordinates[j][5])/ 0.001 if ((15 - passed_coordinates[j][5])/ 0.001)
< (25 - passed_coordinates[j][4]) / 0.001 else (25 - passed_coordinates[j][
                4]) / 0.001
            flag = 1
        for i in range(1,len(coordinates)):
            # 计算当前点跟上一个走过的点之间的距离
            pre_dist = math.sqrt((coordinates[i][0] - passed_coordinates[j][0])**2 +
(coordinates[i][1] - passed_coordinates[j][1])**2 + (coordinates[i][2] -
passed_coordinates[j][2])**2)
            # 计算可选点到 B 点的距离
            if pre_dist <= r    and coordinates[i][3] == flag   : # 判断是否在球里面
                # print(i)
                meet_coordinate.append(coordinates[i]) #找到在球内部的点就把它加进去
                # 再去判断选中的点是否是距离 B 点和之前路径和最小的点
                temp_toB = math.sqrt((B[0]- coordinates[i][0])**2 + (B[1] -
coordinates[i][1])**2 + (B[2] - coordinates[i][2])**2)
                if min_value > weight(temp_toB,pre_dist):
                    # 加一个判断条件
                    pre_toNow_dist = pre_dist
                    min_value = weight(temp_toB,pre_dist)
                    min_i = i #  下表
                    min_toB = temp_toB
            else:
                pass
        min_index.append(min_i)
        mins.append(pre_toNow_dist)
        j += 1
        passed_coordinates.append (coordinates[min_i]) # 经过的点多一
        if j == 1 :
            if judge_candidate() and coordinates[min_i][6]:
```

```
                    wrong_candidates.append([min_i,coordinates[min_i]])
                    passed_coordinates[j][4] = min(pre_toNow_dist * 0.001,5)
                    passed_coordinates[j][5] = pre_toNow_dist * 0.001
                else:
                    passed_coordinates[j][5] = pre_toNow_dist * 0.001
                    passed_coordinates[j][4] = 0
            elif j % 2 == 0:
                if judge_candidate() and coordinates[min_i][6]:
                    wrong_candidates.append([min_i, coordinates[min_i]])
                    passed_coordinates[j][5] = min(passed_coordinates[j - 1][5] +
pre_toNow_dist * 0.001,5)
                    passed_coordinates[j][4] = passed_coordinates[j - 1][4] + pre_toNow_dist *
0.001
                else:
                    passed_coordinates[j][4] = passed_coordinates[j - 1][4] + pre_toNow_dist *
0.001
                    passed_coordinates[j][5] = 0
            elif j % 2 == 1 :
                if judge_candidate() and coordinates[min_i][6]:
                    wrong_candidates.append([min_i, coordinates[min_i]])
                    passed_coordinates[j][4] = min(passed_coordinates[j - 1][4] +
pre_toNow_dist * 0.001, 5)
                    passed_coordinates[j][5] = passed_coordinates[j - 1][5] + pre_toNow_dist *
0.001
                else:
                    passed_coordinates[j][5] = passed_coordinates[j - 1][5] + pre_toNow_dist *
0.001
                    passed_coordinates[j][4] = 0

            print("{0} 【 ---- 当 前 飞 过 第 {1} 号 校 正 点 ---- 】 {2}".format(j, min_i,
passed_coordinates[j]))
            # dist_b = math.sqrt((B[0] - passed_coordinates[j][0]) ** 2 + (B[1] -
passed_coordinates[j][1]) ** 2 + (B[2] - passed_coordinates[j][2]) ** 2)
            if max(passed_coordinates[j][5],passed_coordinates[j][4])  + min_toB * 0.001< 30:
                break
        end_time = time.clock()
        print("{0} 【 ---- 当 前 已 到 达 第 {1} 号 点 ---- 】 {2}".format(j+1, 612,
[100000,59652.3433795158,5022.0011644816404,'B 点 ',min_toB * 0.001,pre_toNow_dist *
0.001  + min_toB * 0.001]))
        print('经过：',min_index)
        print('路径长度：',sum(mins) + min_toB)
        print('错误节点个数：',len(wrong_candidates))
        print('错误节点信息：',wrong_candidates)
        print('用时：%s Seconds' % (end_time - start_time))
```

```
# print(passed_coordinates)
draw_nodes_1 = []
draw_nodes_0 = []
draw_nodes = [[0, 50000.0, 5000.0]]
for i in range(len(passed_coordinates)):
    draw_node_1 = []
    draw_node_0 = []
    x.append(passed_coordinates[i][0])
    y.append(passed_coordinates[i][1])
    z.append(passed_coordinates[i][2])
    if passed_coordinates[i][3] == 1:
        draw_node_1.append(passed_coordinates[i][0])
        draw_node_1.append(passed_coordinates[i][1])
        draw_node_1.append(passed_coordinates[i][2])
        draw_nodes_1.append(draw_node_1)
        draw_nodes.append(draw_node_1)
    elif passed_coordinates[i][3] == 0:
        draw_node_0.append(passed_coordinates[i][0])
        draw_node_0.append(passed_coordinates[i][1])
        draw_node_0.append(passed_coordinates[i][2])
        draw_nodes_0.append(draw_node_0)
        draw_nodes.append(draw_node_0)
draw_nodes.append(B)
print('经过的所有的点', len(draw_nodes), draw_nodes)
print('经过的垂直校正点：',len(draw_nodes_1), draw_nodes_1)
print('经过的水平校正点：', len(draw_nodes_0),draw_nodes_0)
nodes_1 = []  # 其它垂直校正点
nodes_0 = []  # 其它水平校正点
for i in range(num_rows_1):
    node_1 = []
    node_0 = []
    if i not in min_index and i != 0 and i != num_rows_1 - 1:
        if table_1.row_values(i)[3] == 1:
            node_1.append(table_1.row_values(i)[0])
            node_1.append(table_1.row_values(i)[1])
            node_1.append(table_1.row_values(i)[2])
            nodes_1.append(node_1)
        elif table_1.row_values(i)[3] == 0:
            node_0.append(table_1.row_values(i)[0])
            node_0.append(table_1.row_values(i)[1])
            node_0.append(table_1.row_values(i)[2])
            nodes_0.append(node_0)
print('未经过垂直校正点：', nodes_1)
print('未经过水平校正点：', nodes_0)
```

```
    # draw_picture(x,y,z)
if __name__ == '__main__':
    main()
```

问题三-附件 2 程序运行结果截图