“FAST”主动反射面形状调节的研究

摘要

500 米口径球面射电望远镜（Five-hundred-meter Aperture Spheric α radio Telescope，简称 FAST）是我国具有自主知识产权、用于探索宇宙的单口径球面射电望远镜。本文主要研究 FSAT 主动反射面形状的调节，以满足观测不同位置天体时的要求。在反射面板调节约束下，通过调节促动器的径向伸缩量，调整反射面的形状，将其作为工作抛物面，并使得该工作抛物面尽量贴近理想抛物面，以获得天体电磁波经反射面反射后的最佳接收效果。

对于问题一，待观测天体 S 位于基准球面正上方，位置比较特殊，可以直接在空间直角坐标系中，在确定焦点的情况下得到理想抛物面的方程。设出理想抛物面的方程，与促动器下端点（地锚点）和基准态时上端点（顶端）确定的直线相交，通过交点可以计算出理想抛物面相较于基准球面上下移动的距离，距离必须在促动器的约束条件（-0.6m~+0.6m）允许的范围内。同时要求照明区域的口径（即直径）应大于 300m，即在允许的主索节点的集合里，距离 z 轴最远的主索节点与 z 轴的距离应大于 150m，通过程序迭代，选择出符合要求的主索节点的编号。最终得出理想抛物面的方程。

对于问题二：待观测天体 S 的位置不再特殊，考虑坐标系旋转的方法，通过旋转矩阵，使待观测天体 S 的位置在新的坐标系中具有与问题一相同的位置属性，后续的处理方法与问题一类似，得到理想抛物面的方程，并计算主索节点编号、位置坐标、各促动器的伸缩量等结果。

问题三：本题要是对问题二的方案进行评估。假设所有入射电磁波为平行光，考察每一条电磁波与每组相邻主索点构成的三角形平面的交点，如果交点刚好在理想抛物面上，就认为此线刚好射到该三角形平面。入射电磁波关于此三角形平面的反射电磁波与焦面的交点如果和理想抛物面的焦点距离在一米内，则认为成功接收。成功接收的电磁波量与入射到三角面的电磁波量的比值就是接收比。同理，也可以得到基准球面的接收比。

关键词：FAST，旋转抛物面，迭代，坐标系变换，旋转矩阵

1 问题重述

2021 年 3 月 31 日，“中国天眼”（FAST）正式向全球天文学家开放。有了它，可以推动对宇宙深空的了解与探测，为天文学的发展提供新的可能。

在观测不同的天体时，如何调整 FAST 主动反射面的形状，以及信号接收系统（馈源舱）的位置，成为了能够更好待观测天体的关键。FAST 的反射面单元主要由面板单元、背架、调整装置、连接机构等组成[1]，并且有两种工作状态：基准态和工作态。基准态时，反射面是由 4300 个反射单元组成的球冠型索膜结构，半径约 300 米、口径为 500 米，此时的状态称为基准球面；工作态时，根据天文轨迹规划和测量数据，通过调整促动器的伸长量来控制反射面节点位置，在观测方向形成 300 米口径瞬时抛物面以汇聚电磁波[2]，此时的状态工作抛物面。与此同时，馈源舱接收平面的中心被移动到焦面上，主动反射面将来自目标天体的平行电磁波反射汇聚到馈源舱的有效区域，从而实现了天文观测。

因此，本次研究主要针对 FAST 的主动反射面以及信号接收系统（馈源舱）。当观测不同位置的天体时，确定理想抛物面的位置，并且确定如何调节促动器的径向伸缩量，将反射面调节为工作抛物面，使得该工作抛物面尽量贴近理想抛物面，是主动反射面技术的关键。如何建立和简化数学模型，并利用数学模型解决实际问题，是本次研究的重点。

所用数据集介绍：附件 1 给出了所有主索节点的坐标和编号，附件 2 给出了促动器下端点（地锚点）坐标、基准态时上端点（顶端）的坐标，以及促动器对应的主索节点编号，附件 3 给出了 4300 块反射面板对应的主索节点编号。

综上所述，本次研究需要解决的问题如下：

(1) 已知某一天体的方位角和仰角，结合考虑反射面板调节因素，确定理想抛物面,计算出理想抛物面的顶点坐标等相关数据。

(2) 建立反射面板调节模型，调节相关促动器的伸缩量，使反射面尽量贴近该理想抛物面，计算出调节后反射面 300 米口径内的主索节点编号、位置坐标、各促动器的伸缩量等结果。

(3) 基于反射面调节方案，计算调节后馈源舱的接收比，即馈源舱有效区域接收到的反射信号与 300 米口径内反射面的反射信号之比，并与基准反射球面的接收比作比较。

综合上述问题，查阅相关资料及数据，简化反射面的数学模型，分析得到理想抛物面、反射面以及馈源舱的各项状态量，探究理想抛物面以及反射面的实际情况，结合基准数据进行分析和比较，解决实际问题，并撰写研究论文。

2 问题分析

问题一：在此问题中，待观测天体 S 的位置比较特殊，位于基准球面正上方，可以直接在空间直角坐标系中设出理想抛物面的方程，在确定焦点的情况下得到一个抛物面的方程。该抛物面必须使得可以通过调节促动器的伸缩在约束条件（伸缩范围有限）下让主索点尽可能在所求的理想抛物面上，且满足相邻点距离的偏移在可控范围内（0.07%）。通过调节参数，监控基准面和理想抛物面在基准球面法向量上所需的位置移动确立抛物面方程参数的可选范围，最后选取满足条件的参数来确定理想抛物面。

问题二：本题要求我们在问题一的结论下，由题目要求得到一个新的理想抛

物面，并给出我们的调节方案（各促动器的移动值），调节后的结果（主索点的位置）。但是在此问题中，待观测天体 S 的位置不再特殊，这也是本题的难点，我们考虑通过坐标系旋转的方法，使待观测天体 S 的位置在新的坐标系中具有与问题一相同的位置属性，与问题一的处理方法类似，得到理想抛物面的方程，并计算主索节点编号、位置坐标、各促动器的伸缩量等结果。

问题三：本题是对问题二的方案进行评估，在问题二确定的反射面调节方案中，考察馈源舱有效区域接收到的反射信号与 300 米口径内反射面的反射信号之比，即入射电磁波反射到焦点（馈源舱）上的比例，并与基准反射球面的接收比进行比较。假设所有入射电磁波为平行光，每一条电磁波与每组相邻主索点构成的三角形平面的交点，如果交点刚好在理想抛物面上，就认为此线刚好射到该三角形平面。入射电磁波关于此三角形平面的反射电磁波与焦面的交点如果和理想抛物面的焦点距离在一米内，则认为成功接收。成功接收的电磁波量与入射到三角面的电磁波量的比值就是接收比。同理，也可以得到基准球面的接收比。

3 模型假设

(1) 基准态下，所有主索节点均位于基准球面上。
(2) 每一块反射面板均为基准球面的一部分。反射面板上开有许多直径小于 5 毫米的小圆孔，用于透漏雨水。由于小孔的直径小于所观察的天体电磁波的波长，不影响对天体电磁波的反射，所以可以认为面板是无孔的。
(3) 电磁波信号及反射信号均视为直线传播。
(4) 主索节点调节后，相邻节点之间的距离可能会发生微小变化，变化幅度不超过 0.07%。
(5) 将主索节点坐标作为对应的反射面板顶点坐标。
(6) 通过促动器顶端的伸缩，可控制主索节点的移动变位，但连接主索节点与促动器顶端的下拉索的长度保持不变。

4 符号说明

符号	含义	单位
S	被观测的天体	/
C	基准球面的球心	/
R	基准球面的半径	米 (m)
F	馈源舱接收平面中心所在平面与基准球面（两个同心球面）的半径差	米 (m)
α	待观测天体方位的方位角	度 (°)
β	待观测天体方位的仰角	度 (°)

表 1 符号说明

注：α、β 的示意图如图 1 所示。

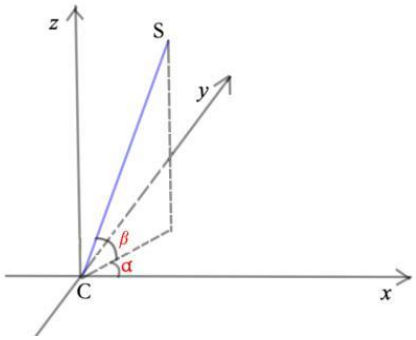

图 1 α、β 的示意图

5 模型的建立与求解

5.1 建模前的准备

5.1.1 实际问题的数学化处理

在基准状态时，反射面为半径约 300 米、口径为 500 米的球面（基准球面），在工作状态时，通过调整促动器的伸长量来控制反射面节点位置，反射面的形状被调节为一个 300 米口径的近似旋转抛物面（工作抛物面），二者可以表示在同一个坐标系下，如图 2 所示。

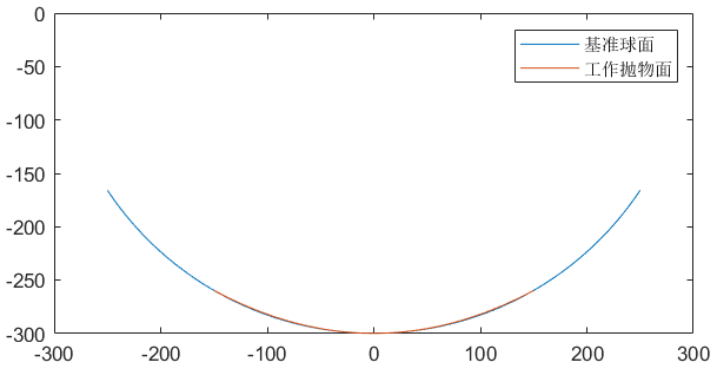

图 2 基准球面与工作抛物面示意图

反射面节点位置是通过调整促动器的伸长量来控制的，如图 3 所示，工作抛物面和基准球面之间可能会存在一定的距离（可能在基准球面上方，也可能在基准球面的下方）。但是，由于促动器顶端径向伸缩范围为-0.6～+0.6 米，工作抛物面和基准球面之间的距离不能超过 0.6 米。

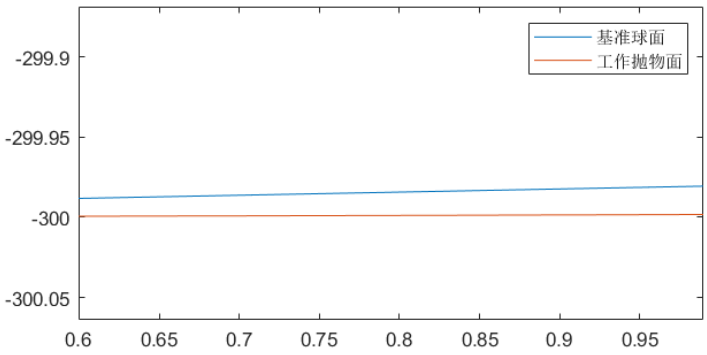

图 3 基准球面与工作抛物面放大示意图

5.1.2 主索节点、促动器及反射面板对应主索节点的示意图

附件 1 给出了所有主索节点的坐标和编号，我们可以画出主索节点的位置示意图，如图 4 所示。

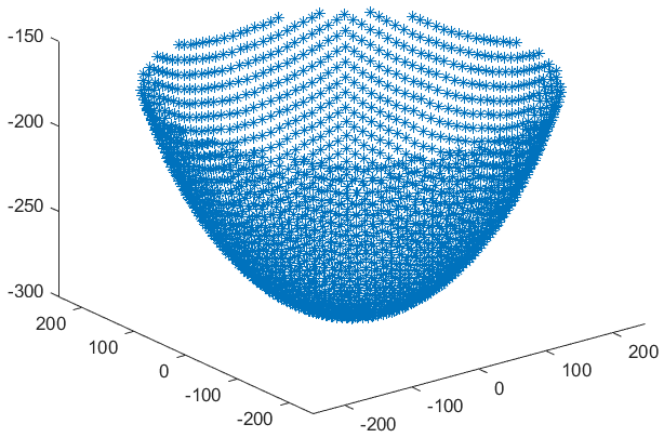

图 4 主索节点的位置示意图

附件 2 给出了促动器下端点（地锚点）坐标、基准态时上端点（顶端）的坐标，以及促动器对应的主索节点编号。分别画出促动器上下端点的示意图，可以表示促动器的位置，如图 5 所示。

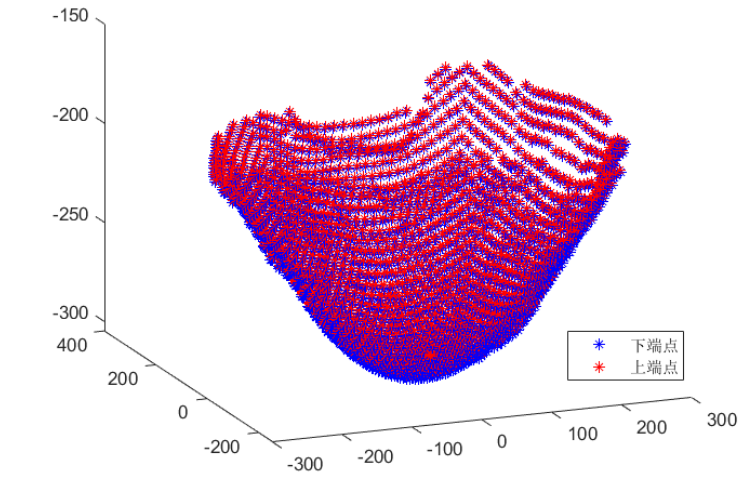

图 5 促动器上下端点的位置示意图

附件 3 给出了反射面板对应的主索节点编号，结合附件 1 作图，画出反射面板位置的示意图，如图 6 所示。

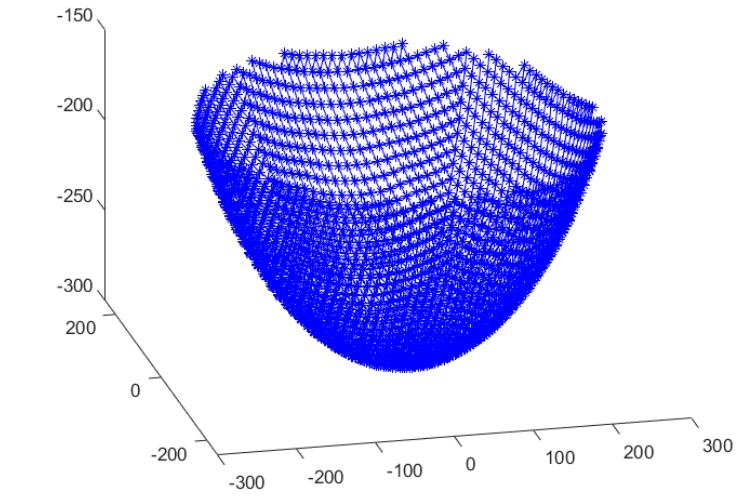

图 6 反射面板位置示意图

5.1.3 球面的焦点和焦距

如图 7 所示的球面，球面的焦点即为球面的球心，坐标为（0,0,0），焦距即为球面的半径 r 。

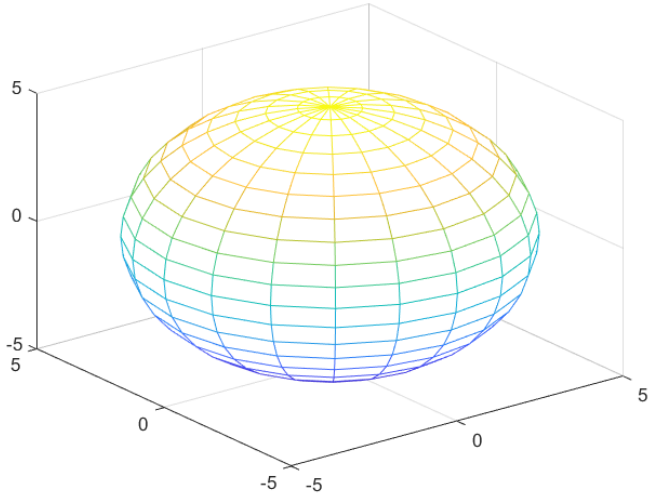

图 7 球面

5.1.3 旋转抛物面的焦点和焦距

如图 8 所示的旋转抛物面，在直角坐标下，其方程为：

$$x^2 + y^2 = 2pz \tag{1}$$

焦点坐标为 $(0, \frac{p}{2})$，焦距为 p 。

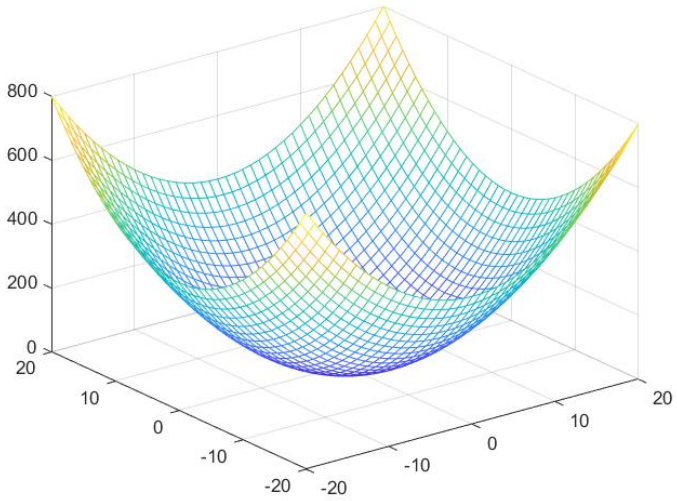

图 8 旋转抛物面

5.2 问题一的建模与求解

5.2.1 促动器上下端点与主索节点三点共线的证明

如何将反射面调节为工作抛物面，是主动反射面技术的关键，该过程的实现需要下拉索与促动器的配合。下拉索长度固定。促动器沿基准球面径向安装，其底端固定在地面，顶端可沿基准球面径向伸缩来完成下拉索的调节，从而调节反射面板的位置，最终形成工作抛物面。简单来说，就是后通过调节促动器的径向

伸缩量，将反射面调节为工作抛物面，使得该工作抛物面尽量贴近理想抛物面。这里就引出了一个基本的问题，那就是促动器的径向伸缩量与反射面板的移动量之间是什么关系呢？通过分析题目，我们可以知道，反射面板安装在三角网格上，每个主索节点连接一根下拉索，下拉索下端与固定在地表的促动器连接，实现对主索网的形态控制。所以我们想到，如果促动器对应的主索节点，促动器基准态时上端点（顶端）以及促动器下端点（地锚点）这三点之间的关系是共线的话，此时促动器顶端的伸缩量就与反射面板位置的上下移动量是相对应的，二者在数值上相等，在方向上也是共线的。反之，如果这三个点不是共线的，那么促动器顶端的伸缩量就不等于反射面板位置的上下移动量，如果想从促动器顶端的伸缩量求得反射面板位置的上下移动量，就需要找出二者之间存在的角度关系，这无疑会大大增加问题的难度，通过下面的过程，可以证明出：促动器对应的主索节点，促动器基准态时上端点（顶端）以及促动器下端点（地锚点）这三点之间的关系是共线的。

采用向量的方法证明。首先，设促动器对应的主索节点为点 A，促动器基准态时上端点（顶端）为点 B，促动器下端点（地锚点）为点 C，在同一个直角坐标系下，三个点的坐标分别为 $(x_1, y_1, z_1)^T$、$(x_2, y_2, z_2)^T$、$(x_3, y_3, z_3)^T$，设向量

$$\overrightarrow{OA} = \overrightarrow{\lambda_1} = (x_1, y_1, z_1)^T \tag{2}$$

$$\overrightarrow{OB} = \overrightarrow{\lambda_2} = (x_2, y_2, z_2)^T \tag{3}$$

$$\overrightarrow{OC} = \overrightarrow{\lambda_3} = (x_3, y_3, z_3)^T \tag{4}$$

如果向量 $\overrightarrow{\mathrm{AC}}$ 与向量 $\overrightarrow{\mathrm{AB}}$ 的外积为 $\vec{0}$，即

$$\left(\overrightarrow{\lambda_3} - \overrightarrow{\lambda_1}\right) \times \left(\overrightarrow{\lambda_2} - \overrightarrow{\lambda_1}\right) = 0 \tag{5}$$

则说明向量 $\overrightarrow{\lambda_3} - \overrightarrow{\lambda_1}$ 与向量 $\overrightarrow{\lambda_2} - \overrightarrow{\lambda_1}$ 是共线的，即点 A、B、C 三点共线。

附件 1 给出了所有主索节点的坐标和编号，附件 2 给出了促动器下端点（地锚点）坐标、基准态时上端点（顶端）的坐标，以及促动器对应的主索节点编号，将附件中的数据带入，即可判断三点是否共线。下面计算一组数据。

主索节点 A1 的坐标为 $(-6.1078, 8.407, -300.22)^{\mathrm{T}}$，与之对应的促动器下端点坐标为 $(-6.1944, 8.526, -304.472)^{\mathrm{T}}$，基准态时上端点坐标为 $(-6.1541, 8.4706, -302.494)^{\mathrm{T}}$，即

$$\overrightarrow{\lambda_1} = (-6.1078, 8.407, -300.22)^T \tag{6}$$

$$\overrightarrow{\lambda_2} = (-6.1944, 8.526, -304.472)^T \tag{7}$$

$$\overrightarrow{\lambda_3} = (-6.1541, 8.4706, -302.494)^T \tag{8}$$

向量的差

$$\overrightarrow{\lambda_3} - \overrightarrow{\lambda_1} = (-0.0463, 0.0636, -2.274)^T \tag{9}$$

$$\overrightarrow{\lambda_2} - \overrightarrow{\lambda_1} = (-0.0866, 0.119, -4.252)^T \tag{10}$$

计算向量外积

$$\begin{aligned}&\left(\overrightarrow{\lambda_3} - \overrightarrow{\lambda_1}\right) \times \left(\overrightarrow{\lambda_2} - \overrightarrow{\lambda_1}\right)\\ &= (-0.0463, 0.0636, -2.274)^T \times (-0.0866, 0.119, -4.252)^T\end{aligned} \tag{11}$$

最终的结果在数量级上为 10^{-5}，与附件中的数据比较可以近似认为是 0。

附件 1 和附件 2 共给出了 2226 组数据，每一组数据都用上面的方式计算一遍显然是不现实的，我们采用了 C++ 程序进行计算，相应的代码在本文的最后给出。最终的结果表示，促动器对应的主索节点，促动器基准态时上端点（顶端）以及促动器下端点（地锚点）这三点之间的关系是共线的。

5.2.2 计算问题一所求的理想抛物面

通过上面的证明，可以简化问题一的求解步骤。对于问题一，待观测天体 S 在基准球面的正上方，即 $\alpha = 0°$，$\beta = 90°$。提出问题的数学化模型，即设理想抛物面 F_1 是旋转抛物面，F_1 的方程：

$$F_1: x^2 + y^2 = 2p(z + \Delta z) \tag{12}$$

为了方便后续的研究和计算，我们想要使旋转抛物面 F_1 的顶点在原点附近，而在附件 1 中，所有主索节点的 z 坐标均为负值，为了与其匹配，我们引入了 Δz，Δz 表示为抛物面在 z 方向上的上下增量，即对 z 方向上的坐标进行上下平移，考虑初始坐标，则

$$\Delta z \in 300.4 + [-0.6, +0.6] \tag{13}$$

以主索节点 A0 为例，当 Δz 取 300.4 时，z 取-300.4，此时

$$z + \Delta z = 0 \tag{14}$$

在 F_1 中刚好就是旋转抛物面的顶点。

F_1 方程中的 p 在数值上等于焦距的 2 倍，即

$$p = 2[\Delta z - (R - F)] \tag{15}$$

其中

$$F = 0.466R \tag{16}$$

$$R = 300m \tag{17}$$

促动器可以看作空间中的一条直线 F_2，设其参数方程为

$$F_2: \begin{cases} x = x_0 + Nt \\ y = y_0 + Mt \\ z = z_0 + Qt \end{cases} \tag{18}$$

其中，(x_0, y_0, z_0) 为基准点（主索节点）的坐标，t 是参数，M、N、Q 为 x、y、z 三个方向的方向参数，满足

$$\begin{cases} N = x_l - x_h \\ M = y_l - y_h \\ Q = z_l - z_h \end{cases} \tag{19}$$

其中，(x_l, y_l, z_l) 和 (x_h, y_h, z_h) 分别是促动器下端点（地锚点）和基准态时上端点（顶端）的坐标。

联立方程 F_1 和 F_2

$$(x_0 + Nt)^2 + (y_0 + Mt)^2 = 2p[(z_0 + Qt) + \Delta z] \tag{20}$$

化简得

$$(N^2+M^2)t^2+2(Nx_0+My_0-PQ)t+[{x_0}^2+{y_0}^2-2p(z_0+\Delta z)]=0 \quad (21)$$

简记为

$$At^2+Bt+C=0 \quad (22)$$

其中

$$A=N^2+M^2 \quad (23)$$

$$B=2(Nx_0+My_0-PQ) \quad (24)$$

$$C={x_0}^2+{y_0}^2-2p(z_0+\Delta z) \quad (25)$$

对于每一个主索节点，附件 1 都给出了其坐标，式 (20) 是一元二次方程，将主索节点的坐标代入，都可以解出一组 t_1、t_2 满足式 (20)。t_1、t_2 分别对应了一个在该参数下理想抛物面相较于基准球面上下移动的距离 d_1、d_2，距离 d_1、d_2 分别满足

$$d_1=\sqrt{M^2+N^2+Q^2}\times t_1 \quad (26)$$

$$d_2=\sqrt{M^2+N^2+Q^2}\times t_2 \quad (27)$$

取

$$d=min\{d_1,d_2\} \quad (28)$$

且

$$d<0.6 \quad (29)$$

称满足式 (24) 和式 (25) 两个条件的主索节点为允许的主索节点的集合。

同时，要求照明区域的口径（即直径）应大于 300m，即在允许的主索节点的集合里，距离 z 轴最远的主索节点与 z 轴的距离应大于 150m，这个距离定义为 R′，利用 C++程序，计算出 Δz 取不同值时对应的 R′ 的值，如表 2 所示。

Δz 的值/m	R′ 的值/m
299.00	0
299.01	0
299.02	0
……	
299.80	0
299.81	0
299.82	10.3915
299.83	10.3915
299.84	16.818
299.85	16.818
……	
300.46	102.397

300.47	158.85
300.48	158.85
300.49	158.85
300.50	158.85
……	
300.86	151.272
300.87	151.272
300.88	148.996
300.89	148.996
300.90	148.996
……	

表 2 Δz 取不同值时对应的 R′ 的值

通过描点、连线，建立 Δz 与 R′ 的关系图像，如图 9 所示。

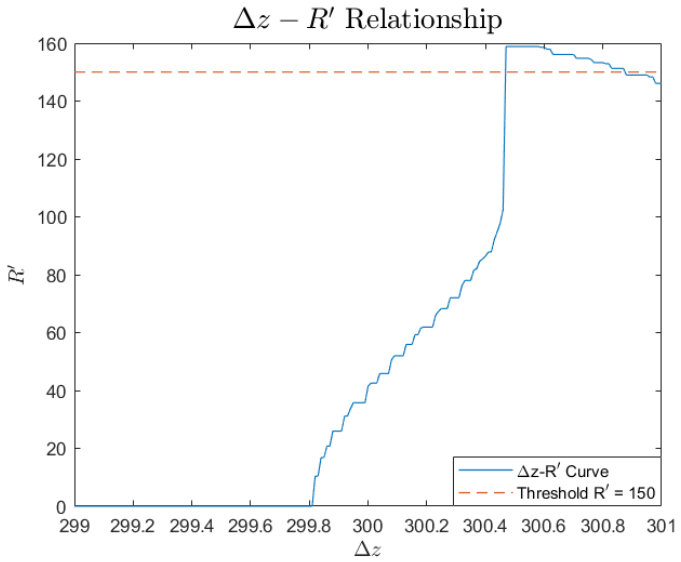

图 9 Δz 与 R′ 的关系

从表 2 和图 9 可以看出，满足口径大于 300m 的 Δz 范围结果（单位：m）是

$$\Delta z \in [300.47,300.87] \tag{30}$$

把 Δz 和式 (16) 式 (17) 代入式 (15)

$$p = 2[\Delta z - (R - F)] = 2(\Delta z - 0.534 \times 300) \tag{31}$$

得到 p 的范围（单位：m）

$$p \in [280.54,281.34] \tag{32}$$

同时，题目要求主索节点调节后，相邻节点之间的距离可能会发生微小变化，变化幅度不超过 0.07%，变化幅度如图 10 所示，可见，求得的 Δz 满足要求。

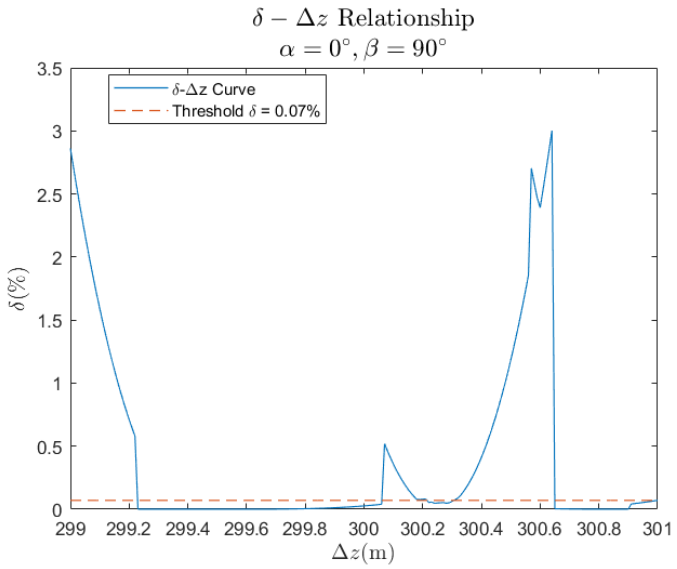

图 10 相邻节点距离的变化幅度

所以，理想抛物面的方程（方程中各字母代表的量的单位：m）是

$$F_1: x^2 + y^2 = 2p(z + \Delta z) \tag{33}$$

其中

$$\Delta z \in [300.47, 300.87] \tag{34}$$

$$p \in [280.54, 281.34] \tag{35}$$

5.3 问题二的建模与求解

5.3.1 三维直角坐标系旋转矩阵的推导及其应用

在问题二中，待观测天体 S 的位置不再特殊，方位角 α 和仰角 β 都是一般的常数，于是我们考虑通过旋转坐标系的方法来变换待观测天体 S 的坐标，将问题简化。

旋转矩阵是在乘以一个向量的时候有改变向量的方向但不改变大小的效果并保持了手性的矩阵。旋转矩阵不包括点反演，点反演可以改变手性，也就是把右手坐标系改变成左手坐标系或反之。所有旋转加上反演形成了正交矩阵的集合。旋转可分为主动旋转与被动旋转。主动旋转是指将向量逆时针围绕旋转轴所做出的旋转。被动旋转是对坐标轴本身进行的逆时针旋转，它相当于主动旋转的逆操作。[3]

当进行三维的旋转时，首先要规定正负方向，在本文中，如无特殊说明，所有的三维直角坐标系均为右手系。

坐标系旋转前后，坐标的变换关系如下：[4]

$$(\text{变换后的坐标}) = [\text{旋转矩阵}] \cdot (\text{变换前的坐标}) \tag{36}$$

(1) 绕 x 轴旋转角度为 α（在 yz 平面顺时针旋转）

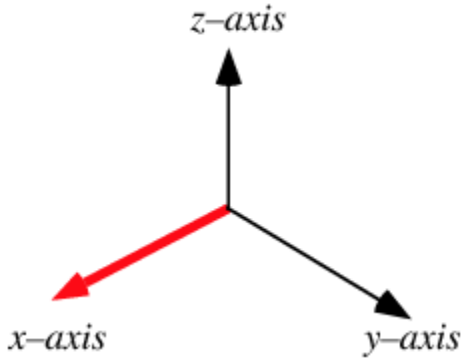

图 11 绕 x 轴旋转

进行如图 11 所示的旋转时（旋转的角度为 α ），旋转矩阵为

$$\mathcal{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \tag{37}$$

(2) 绕 y 轴旋转角度为 β（在 zx 平面顺时针旋转）

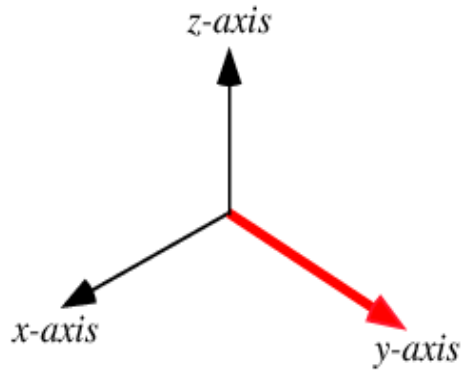

图 12 绕 y 轴旋转

进行如图 12 所示的旋转时（旋转的角度为 β ），旋转矩阵为

$$\mathcal{R}_y = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \tag{38}$$

(3) 绕 z 轴旋转角度为 γ（在 xy 平面顺时针旋转）

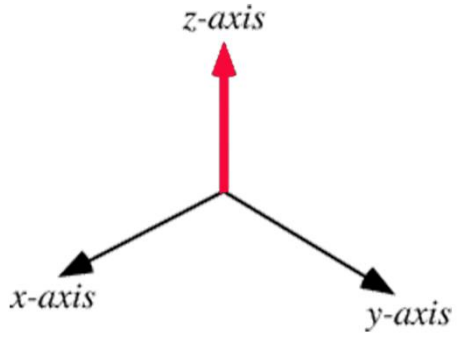

图 13 绕 z 轴旋转

进行如图 13 所示的旋转时（旋转的角度为 β ），旋转矩阵为

$$\mathcal{R}_z = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \tag{39}$$

5.3.2 问题二的建模与求解

上面讨论了空间直角坐标系绕不同轴旋转时对应的旋转矩阵，如图 14 所示，考虑到待观测天体 S 的方位角 $\alpha = 36.795°$ 和仰角 $\beta = 78.169°$ 。

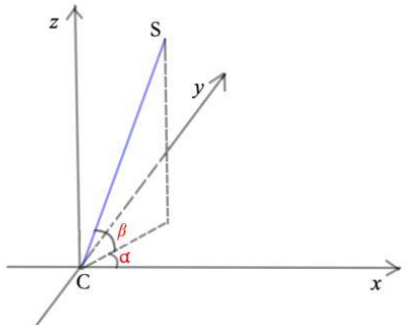

图 14 α、β 的示意图

对应到坐标系旋转中，等价于先绕 z 轴旋转 α 角，再绕 y 轴旋转 $\frac{\pi}{2} - \beta$ 角，记

$$\theta = \frac{\pi}{2} - \beta \tag{40}$$

θ 角如图 15 所示

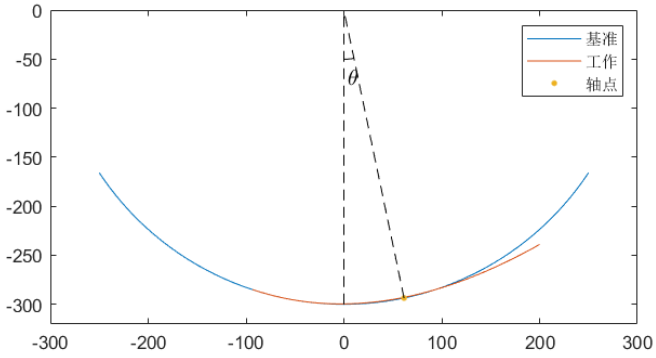

图 15 θ 角的示意图

对应的旋转矩阵

$$\mathcal{R}_z(\alpha) = \begin{bmatrix} cos\alpha & 0 & -sin\alpha \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix} \tag{41}$$

$$\mathcal{R}_y(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{42}$$

设空间中任一点的坐标为

$$\lambda = (x, y, z)^T \tag{43}$$

那么在经过坐标系变换后，在原坐标系中，λ 的坐标变为

$$\lambda_1 = (x_1, y_1, z_1)^T = \mathcal{R}_y(\theta)\mathcal{R}_z(\alpha)\lambda \tag{44}$$

与问题一中的方法类似，在坐标系变换之前，设理想抛物面 F_0 的方程：

$$F_0: x^2 + y^2 = 2p(z + \Delta z) \tag{45}$$

经过坐标系变换，抛物面 F_1 的方程：

$$F_1: x_1^2 + y_1^2 = 2p(z_1 + \Delta z_1) \tag{46}$$

把坐标变换关系式(49)代入，得

$$F_1: (ycos\alpha - xsin\alpha)^2 + (xcos\alpha cos\theta + ysin\alpha cos\theta - zsin\theta)^2$$

$$= 2p(xcos\alpha sin\theta + ysin\alpha sin\theta + zcos\theta - \Delta z) \tag{47}$$

方程中的 p 在数值上等于焦距的 2 倍，即

$$p = 2[\Delta z - (R - F)] \tag{48}$$

其中

$$F = 0.466R \tag{49}$$

$$R = 300m \tag{50}$$

同样地，把促动器看作空间中的一条直线 F_2，设其参数方程为

$$F_2: \begin{cases} x_1 = x_0 + Nt \\ y_1 = y_0 + Mt \\ z_1 = z_0 + Qt \end{cases} \tag{51}$$

其中，(x_0, y_0, z_0) 是基准点（主索节点）的坐标，t 是参数，M、N、Q 为 x、y、z 三个方向的方向参数，满足

$$\begin{cases} N = x_l - x_h \\ M = y_l - y_h \\ Q = z_l - z_h \end{cases} \tag{52}$$

其中，(x_l, y_l, z_l) 和 (x_h, y_h, z_h) 分别是促动器下端点（地锚点）和基准态时上端点（顶端）的坐标。

联立方程 F_1 和 F_2，化简得

$$D: At^2 + Bt + C = 0 \tag{53}$$

即

$$\begin{aligned}
&(M^2 cos^2\alpha + N^2 sin^2\alpha + Q^2 sin^2\theta + N^2 cos^2\alpha cos^2\theta + M^2 sin^2\alpha cos^2\theta \\
&\quad - 2NQcos\alpha cos\theta sin\theta - 2MQsin\alpha cos\theta sin\theta + 2MNcos\alpha sin\alpha cos^2\theta \\
&\quad - 2MNcos\alpha sin\alpha)t^2 \\
&+(2qz_0 sin^2\theta - 2PQcos\theta + 2My_0 cos^2\alpha + 2Nx_0 sin^2\alpha + 2Nx_0 cos^2\alpha cos^2\theta \\
&\quad + 2My_0 sin^2\alpha cos^2\theta - 2Mx_0 cos\alpha sin\alpha - 2Ny_0 cos\alpha sin\alpha \\
&\quad - 2Npcos\alpha sin\theta - 2Mpsin\alpha sin\theta - 2Nz_0 cos\alpha cos\theta sin\theta \\
&\quad - 2Qx_0 cos\alpha cos\theta sin\theta - 2Mz_0 sin\alpha cos\theta sin\theta - 2Qy_0 sin\alpha cos\theta sin\theta \\
&\quad + 2Mx_0 cos\alpha sin\alpha cos^2\theta + 2Ny_0 cos\alpha sin\alpha cos^2\theta)t \\
&+(-2p\Delta z + x_0 sin^2\alpha + y_0 cos^2\alpha + z_0 sin^2\theta + x_0^2 cos^2\alpha cos^2\theta \\
&\quad + y_0^2 sin^2\alpha cos^2\theta - 2py_0 sin\alpha sin\theta + -2x_0 y_0 cos\alpha sin\alpha \\
&\quad - 2px_0 cos\alpha sin\theta + 2x_0 y_0 cos\alpha sin\alpha cos^2\theta - 2x_0 z_0 cos\alpha cos\theta sin\theta \\
&\quad - 2y_0 z_0 sin\alpha cos\theta sin\theta - 2pz_0 cos\theta) \\
&\quad = 0
\end{aligned} \tag{54}$$

解法与问题一相同，利用C++程序，计算出 Δz 取不同值时对应的 R′ 的值，

二者关系图如图 16 所示。

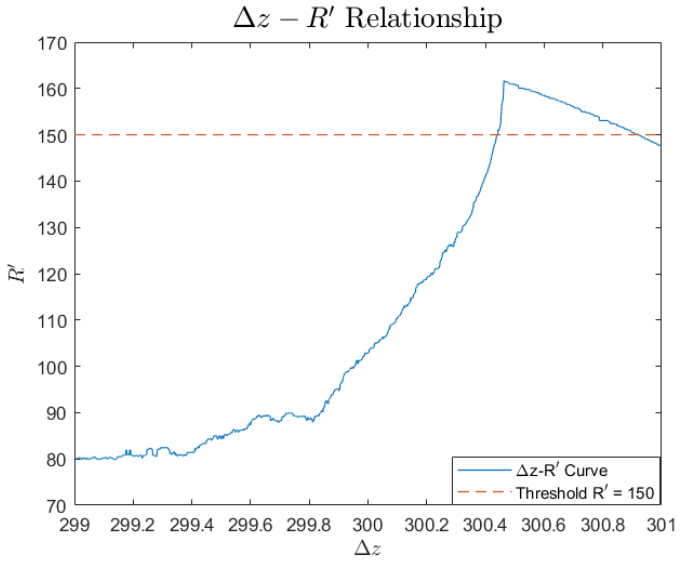

图 16 Δz 与 R′ 的关系

满足口径大于 300m 的 Δz 范围结果（单位：m）是

$$\Delta z \in [300.442, 300.922] \tag{55}$$

把 Δz 和式 (54) 式 (55) 代入式 (53)

$$R = 300m \tag{56}$$

$$p = 2[\Delta z - (R - F)] = 2(\Delta z - 0.534 \times 300) \tag{57}$$

得到 p 的范围（单位：m）

$$p \in [280.484, 281.444] \tag{58}$$

同时，题目要求主索节点调节后，相邻节点之间的距离可能会发生微小变化，变化幅度不超过 0.07%，变化幅度如图 17 所示，可见，求得的 Δz 满足要求。

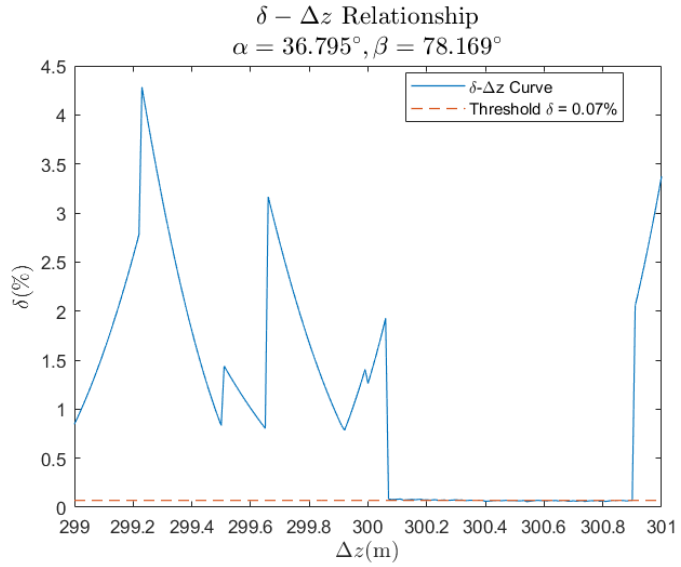

图 17 相邻节点距离的变化幅度

所以，理想抛物面的方程（方程中各字母代表的量的单位：m）是

$$F_1: x^2 + y^2 = 2p(z + \Delta z) \tag{59}$$

其中

$$\Delta z \in [300.442, 300.922] \tag{60}$$

$$p \in [280.484, 281.444] \tag{61}$$

图 18、图 19、图 20 对理想抛物面进行了大致的说明。

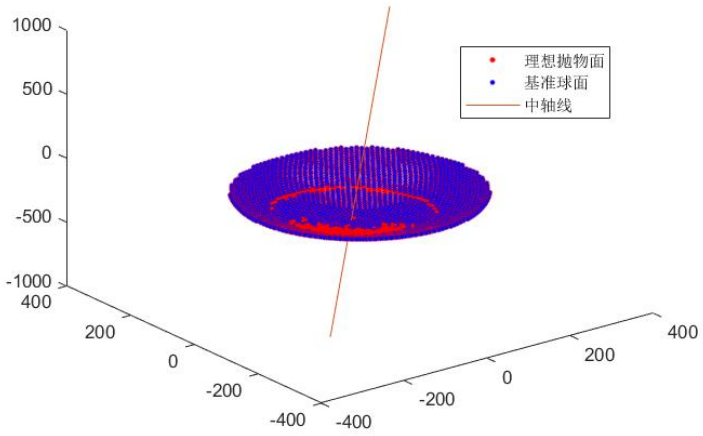

图 18 理想抛物面的示意图

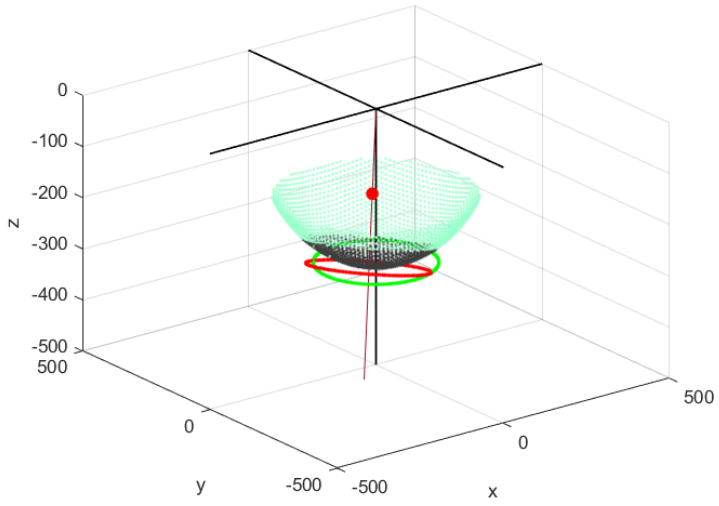

图 19 理想抛物面与基准球面的对比，绿色点组成基准球面，绿色区域为到水平面的投影。黑色点代表需要调整的三角形反射面，在水平面上投影为红色区域

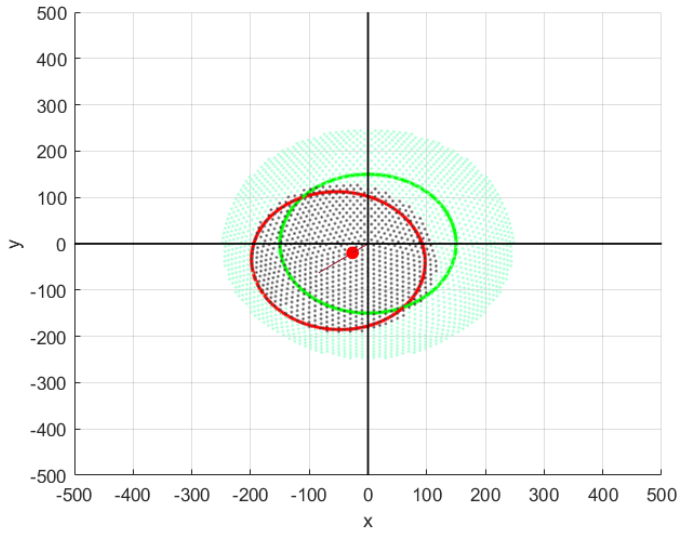

图 20 俯视图

题目中要求的理想抛物面的顶点坐标，以及调节后反射面 300 米口径内的主索节点编号、位置坐标、各促动器的伸缩量等结果按照规定的格式（见附件 4）保存在了“result.xlsx”文件中。

5.4 问题三的建模与求解

基于问题二的反射面调节方案，假设有若干束入射电磁波设想反射面，能够照射到反射面上的入射电磁波是一个椭圆形区域，如图 21 所示。

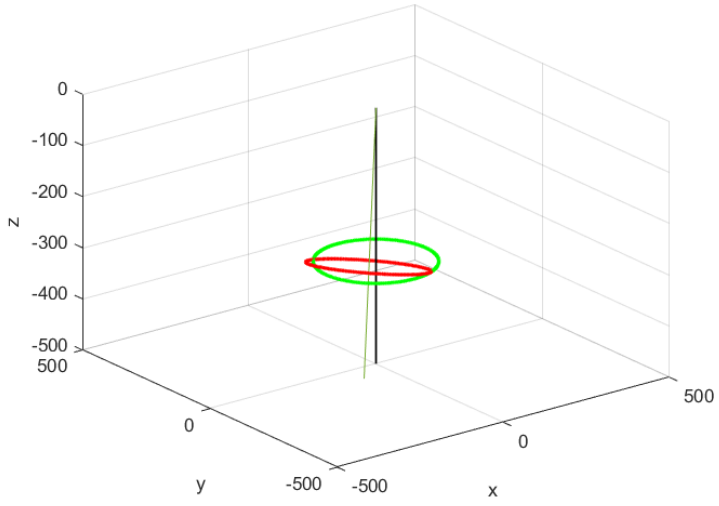

图 21 反射面能接收到的入射电磁波的区域（红色）

也就是说，在某个椭圆形区域内入射的电磁波才会被反射面接收，这一区域如图 22 所示。在这个区域内，所有的入射电磁波都是平行的，即方向相同，用程序模拟足够多的入射电磁波，对问题三进行处理。

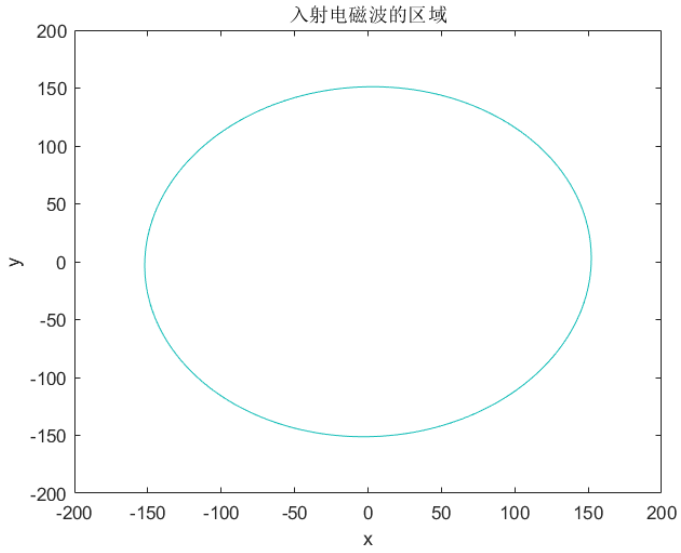

图 22 入射电磁波的区域

(1) 计算入射电磁波与三角形反射面所在平面的交点

待观测天体 S 的方位角 $\alpha = 36.795°$ 和仰角 $\beta = 78.169°$，取向量 $\overrightarrow{OS}$ 的单位方向向量，直角坐标为

$$\vec{n} = (cos\beta cos\alpha, cos\beta sin\alpha, sin\beta) = (0.16418, 0.12280, 0.97876) \tag{62}$$

取该方向上的任意一点 (x_4, y_4, z_4)，那么入射电磁波的方程可以确定

$$\frac{x - x_4}{x_0} = \frac{y - y_4}{y_0} = \frac{z - z_4}{z_0} \tag{63}$$

每一块三角形反射面的三个顶点各有一个主索节点，设这三个主索节点的坐标分别为 (x_1, y_1, z_1)、(x_2, y_2, z_2)、(x_3, y_3, z_3)，三角形反射面所在平面的方程为：

$$M(x - x_1) + N(y - y_1) + Q(z - z_1) = 0 \tag{64}$$

其中

$$M = (y_2 - y_1)(z_3 - z_1) - (y_3 - y_1)(z_2 - z_1) \tag{65}$$

$$N = (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1) \tag{66}$$

$$Q = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1) \tag{67}$$

联立式 (63) 和式 (64)，即可解得入射电磁波与三角形反射面交点的坐标如下：

若 $x_0 \neq 0$ 且 $y_0 \neq 0$

$$x = \frac{\left[N\left(\frac{y_0}{x_0}\right) + Q\left(\frac{z_0}{x_0}\right)\right] x_4 + Mx_1 + Ny_1 + Qz_1 - Ny_4 - Qz_4}{M + N\left(\frac{y_0}{x_0}\right) + Q\left(\frac{z_0}{x_0}\right)} \tag{68}$$

$$y = \left(\frac{y_0}{x_0}\right)(x - x_4) + y_4 \tag{69}$$

$$z = \left(\frac{z_0}{x_0}\right)(x - x_4) + z_4 \tag{70}$$

若 $x_0 = 0$ 且 $y_0 \neq 0$

$$x = x_4 \tag{71}$$

$$y = \frac{Mx_1 + Ny_1 + Qz_1 - Mx_4 - Qz_4 + Q\frac{z_0}{y_0}y_4}{N + Q\frac{z_0}{y_0}} \tag{72}$$

$$z = \frac{z_0}{y_0}(y - y_4) + z_4 \tag{73}$$

若 $x_0 = 0$ 且 $y_0 = 0$

$$x = x_4 \tag{74}$$

$$y = y_4 \tag{75}$$

$$z = \frac{Mx_1 + Ny_1 + Qz_1 - Mx_4 - Ny_4}{Q} \tag{76}$$

至此，求出了入射电磁波与三角形反射面所在平面的交点。

(2) 确定交点是否在三角形反射面内部

上面的过程求出了入射电磁波与三角形反射面所在平面的交点，下面判断这个交点是否在三角形反射面内部，如果每条电磁波和每个相邻主索点组成的三角面的交点在三角形反射面内，那么这一点也在三角形反射面组成的理想抛物面上。用向量的方法证明一点在三角形内部。

对于任一三角形 ABC 和一点 P，可以有如下的向量表示：

$$\overrightarrow{AP} = u\overrightarrow{AC} + v\overrightarrow{AB} \tag{77}$$

则 P 点在 ΔABC 内部的充分必要条件是：

$$0 \le u \le 1 \tag{78}$$

$$0 \le v \le 1 \tag{79}$$

$$u + v \le 1 \tag{80}$$

在直角坐标系中，如果已知 A、B、C、P 四个点的坐标，可以求出 u、v，方法如下：

$$\overrightarrow{AP} \cdot \overrightarrow{AC} = \left(u\overrightarrow{AC} + v\overrightarrow{AB}\right) \cdot \overrightarrow{AC} \tag{81}$$

$$\overrightarrow{AP} \cdot \overrightarrow{AB} = \left(u\overrightarrow{AC} + v\overrightarrow{AB}\right) \cdot \overrightarrow{AB} \tag{82}$$

解方程得到：

$$u = \frac{\left(\overrightarrow{AB} \cdot \overrightarrow{AB}\right) \times \left(\overrightarrow{AP} \cdot \overrightarrow{AC}\right) - \left(\overrightarrow{AC} \cdot \overrightarrow{AB}\right) \times \left(\overrightarrow{AP} \cdot \overrightarrow{AB}\right)}{\left(\overrightarrow{AC} \cdot \overrightarrow{AC}\right) \times \left(\overrightarrow{AB} \cdot \overrightarrow{AB}\right) - \left(\overrightarrow{AC} \cdot \overrightarrow{AB}\right) \times \left(\overrightarrow{AB} \cdot \overrightarrow{AC}\right)} \tag{83}$$

$$v = \frac{\left(\overrightarrow{AC} \cdot \overrightarrow{AC}\right) \times \left(\overrightarrow{AP} \cdot \overrightarrow{AB}\right) - \left(\overrightarrow{AC} \cdot \overrightarrow{AB}\right) \times \left(\overrightarrow{AP} \cdot \overrightarrow{AC}\right)}{\left(\overrightarrow{AC} \cdot \overrightarrow{AC}\right) \times \left(\overrightarrow{AB} \cdot \overrightarrow{AB}\right) - \left(\overrightarrow{AC} \cdot \overrightarrow{AB}\right) \times \left(\overrightarrow{AB} \cdot \overrightarrow{AC}\right)} \tag{84}$$

如果 u、v 分别满足式(67)、式(68)、式(69)，那么 P 点在 ΔABC 内部。

经过计算，对于模拟的每一条入射电磁波，交点均位于三角形反射平面内部。

(3) 确定反射电磁波

由光的反射定律，反射电磁波与入射电磁波与法线在同一平面上；反射电磁波和入射电磁波分居在法线的两侧；反射角等于入射角。

如图 23 所示，入射点为 O，该点的单位法向量为 $\vec{\mathrm{N}}$，入射电磁波为 $\overrightarrow{\mathrm{AO}}$，反射电磁波为 $\overrightarrow{\mathrm{OB}}$，把传播路径看成一个等腰三角形 AOB，则

$$\overrightarrow{OB} = \overrightarrow{AB} - \overrightarrow{AO} \tag{85}$$

$$\overrightarrow{AB} = 2\overrightarrow{AP} = 2\left(\overrightarrow{AO} + \overrightarrow{OP}\right) \tag{86}$$

$$\overrightarrow{OB} = \overrightarrow{AO} + 2\overrightarrow{OP} \tag{87}$$

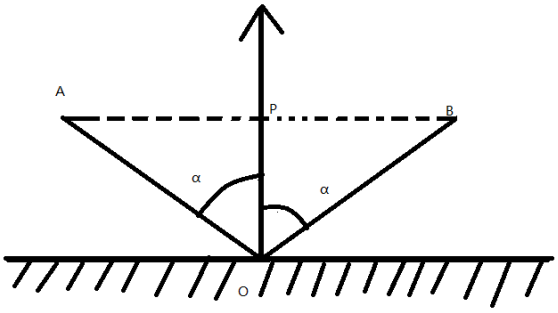

图 23 电磁波反射示意图

于是问题转化为求 $\overrightarrow{\mathrm{OP}}$ 的问题。

$$\overrightarrow{OP} = -\left(\overrightarrow{AO} \cdot \vec{N}\right) \cdot \vec{N} \tag{88}$$

把式 (88) 代入式 (87)，得

$$\begin{aligned} \overrightarrow{OB} &= \overrightarrow{AO} - 2\left(\overrightarrow{AO} \cdot \vec{N}\right) \cdot \vec{N} \\ &= (x_0, y_0, z_0) \end{aligned} \tag{89}$$

入射点即为（1）中求得的交点 $(\mathrm{x}_5, \mathrm{y}_5, \mathrm{z}_5)$，由此可以求出反射电磁波的方程:

$$\frac{x - x_5}{x_0} = \frac{y - y_5}{y_0} = \frac{z - z_5}{z_0} \tag{90}$$

(4) 计算反射电磁波与焦面的交点

上面求出了反射电磁波的方程，焦面是一个球面，方程为

$$x^2 + y^2 + z^2 = r^2 \tag{91}$$

联立式 (90) 和式 (91)，化简得

$$ax^2 + bx + c = 0 \tag{92}$$

分以下三种情况讨论：

若 $x_0 \neq 0$，则

$$k_1 = \frac{y_0}{x_0} \tag{93}$$

$$k_2 = \frac{z_0}{x_0} \tag{94}$$

$$l_1 = y_5 - \left(\frac{y_0}{x_0}\right)x_5 \tag{95}$$

$$l_2 = z_5 - \left(\frac{z_0}{x_0}\right)x_5 \tag{96}$$

$$a = 1 + k_1^2 + k_2^2 \tag{97}$$

$$b = 2k_1l_1 + 2k_2l_2 \tag{98}$$

$$c = l_1^2 + l_2^2 - r^2 \tag{99}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{100}$$

$$y = k_1x + l_1 \tag{101}$$

$$z = k_2x + l_2 \tag{102}$$

交点为 $(\frac{-b\pm\sqrt{b^2-4ac}}{2a}, k_1x + l_1, k_2x + l_2)$。

若 $x_0 = 0$ 且 $y_0 \neq 0$，则

$$k_3 = \frac{z_0}{y_0} \tag{103}$$

$$l_3 = z_5 - \frac{z_0}{y_0}y_5 \tag{104}$$

$$a = 1 + k_3^2 \tag{105}$$

$$b = 2k_3l_3 \tag{106}$$

$$c = l_3^2 + x_5^2 - r^2 \tag{107}$$

$$x = x_5 \tag{108}$$

$$z = k_3y + l_3 \tag{109}$$

$$y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{110}$$

交点为 $(\mathrm{x}_5, \frac{-b\pm\sqrt{b^2-4ac}}{2a}, k_3x + l_3)$。

若 $x_0 = 0$ 且 $y_0 = 0$，则

$$x = x_5 \tag{111}$$

$$y = y_5 \tag{112}$$

$$z = \pm\sqrt{r^2 - x_5^2 - y_5^2} \tag{113}$$

交点为 $(\mathrm{x}_5, y_5, \pm\sqrt{r^2 - x_5^2 - y_5^2})$。

(5) 计算交点与馈源舱的距离

反射电磁波与焦面的交点表示来自目标天体的平行电磁波反射汇聚到焦面的位置，馈源舱接收信号的有效区域为直径 1 米的中心圆盘，即如果交点与馈源舱的距离小于等于 1 米，就认为馈源舱有效接收。

由式 (62)，

$$\vec{n} = (cos\beta cos\alpha, cos\beta sin\alpha, sin\beta) = (0.16418, 0.12280, 0.97876) \tag{114}$$

由此计算馈源舱的坐标

$$P = -\vec{n}(R - F) = (26.30164, 19.67256, 156.79735) \tag{115}$$

求与第四步求得的交点的距离，如果小于等于 1 米，表示馈源舱有效接收。

基准反射球面和问题二反射面调节后的接收情况分别如图 24 和图 25 所示，在图中，蓝色是入射电磁波的区域，绿色是反射到焦面的电磁波位置，黄色是基准反射球面反射的电磁波位置，黑色是主索节点的位置，红色直线是理想抛物面顶点的法线，直线上的红点是馈源舱有效接收的位置。

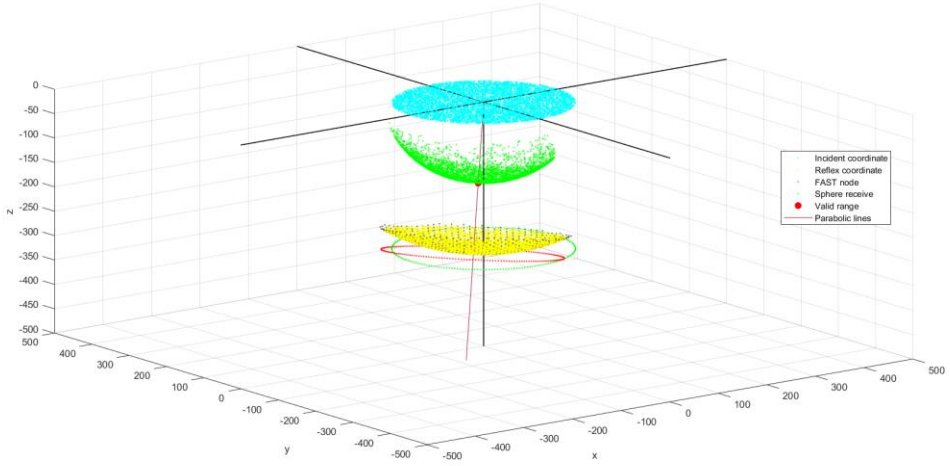

图 24 基准反射球面的接收情况

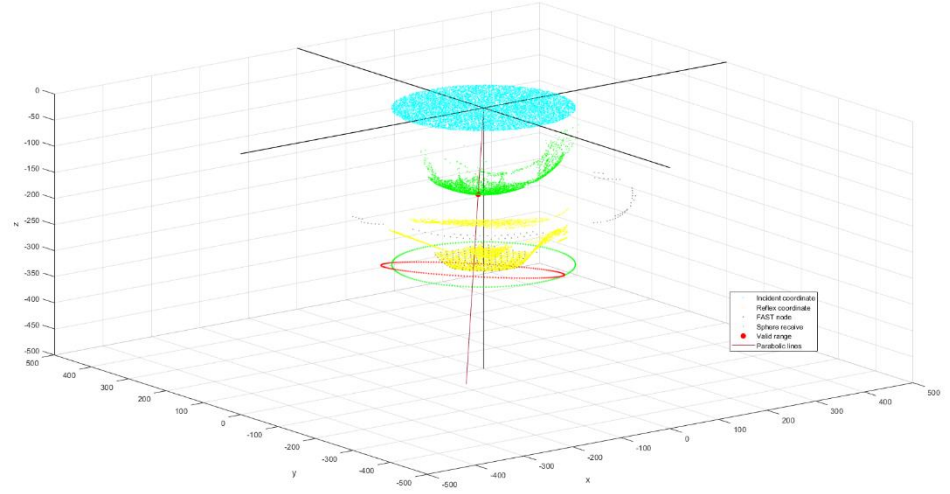

图 25 问题二反射面调节后的接受情况

对比两张接收情况图可以看出，问题二反射面调节后的接收情况优于基准反射球面的接收情况。

使用程序模拟大量平行入射的电磁波，经过工作抛物面后反射在焦面上，求出其中满足要求的电磁波的数量，与总的电磁波的数量相比，得到所求的接收比，最后的结果如表 3 所示。

	馈源舱有效接收电磁波的数量	反射在焦面上电磁波的数量	接收比
问题二反射面调节方案	51	121148	0.042097
基准反射球面	105	628124	0.016716

表 3 接收比

6 模型评价

模型优点：

(1) 伸缩量的求解：由于直接在基准球面径向移动使得促动器的伸缩量能较简易的得到结果。

(2) 接收比的计算：直接用大量数据模拟，结果更接近真实情况，能更好的反映模型的好坏。

模型缺点：

(1) 伸缩量的求解：将每个点分开考虑难以直接得到相邻点距离的变化，需要进行进一步的优化才能较好的满足要求

(2) 接收比的计算：计算公式繁多，计算量大，对计算机算力要求很高。

7 参考文献

[1] FAST 工程办公室.500 米口径球面射电望远镜工程[J].科学,2016,068(004):6-9.

[2] 中国科学院国家天文台,中国科学院基础科学局.中国天文科学大型装置的研制与应用——500 米口径球面射电望远镜(FAST)[J].中国科学院院刊,2009,024(006):670-673,插 2,封 3.

[3] 许永平.旋转矩阵的概念与一些结论[J].江苏广播电视大学学报,1997,2:81-84.

[4] 杨凡,李广云,王力.三维坐标转换方法研究[J].测绘通报,2010(6):5-7.

8 附录

支撑材料

支撑材料\C++程序

支撑材料\C++程序\Auxiliary_program_main.cpp

支撑材料\C++程序\Auxiliary_program_sub.cpp

支撑材料\C++程序\基准面模拟 1.out

支撑材料\C++程序\基准面模拟 2.out

支撑材料\C++程序\基准面模拟详表.txt

支撑材料\matlab

支撑材料\matlab\C.txt

支撑材料\matlab\cloud.m

支撑材料\matlab\D.txt

支撑材料\matlab\data.txt

支撑材料\matlab\drawsphere.m

支撑材料\matlab\err.txt

支撑材料\matlab\err0.txt

支撑材料\matlab\fujian1.m

支撑材料\matlab\fujian2.m

支撑材料\matlab\fujian3.m

支撑材料\matlab\in1.txt

支撑材料\matlab\in2.txt

支撑材料\matlab\in3.txt
支撑材料\matlab\lixiangpaowumian.m
支撑材料\matlab\receive.m
支撑材料\matlab\sandian.m
支撑材料\matlab\sta1.txt
支撑材料\matlab\sta3.txt
支撑材料\matlab\standard.m
支撑材料\matlab\yinhanshu.m
支撑材料\matlab\z_R.m
支撑材料\承诺书.pdf
支撑材料\附件 4.xlsx
#
文件数
31

9 代码

辅助程序 1（Auxiliary_program_main.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

/*
 * 主程序处理：用于问题 1、问题 2
 */
struct data{
    string s;//节点名
    double x[3], y[3], z[3];//节点坐标（0 主索点 1 促动轴底 2 促动轴顶）
    bool check;//是否共线
    int seq;//排序编号（按与主轴水平距离升序）
};
struct res{
    int num[2], id[2];//num 连续数目 id 最远序号
    double dis[2], err;//最远距离 误差
    //方法 1 直接按距离顺序取样 方法 2 按照相邻点扩展取样
};
vector <pair<double, int> > seq;//排序器 存距离、节点编号
vector <data> v;//节点数据存放器
vector <double> p[3];
map <string, int> name2id;//映射器 节点名->节点编号
map <vector<int>, int> boardVerticle2id;//映射器 反射板节点编号->反射板编号
map <vector<int>, int>::iterator iter;//映射器迭代器
vector <int> connect[2226];//每个节点与之连接的其他节点编号存放器
vector <vector<int> > board;//反射板节点存放器
vector <double> boardSize;//反射板面积
vector <int> visList;//遍历寄存器
```

```
28  int allow[2227], allow1[2227], vis[2227];//allow、allow1：节点是否可行（方法 1/方法 2）vis：遍历缓存器
29  double al = 0*M_PI/180, th = (0)*M_PI/180;//alpha-α theta-θ=π/2-β 这里去除注释，用于问题 1
30  //double al = 36.795*M_PI/180, th = (90-78.169)*M_PI/180; //这里去除注释，用于问题 2
31
32  bool check(data a)//检查是否满足三点共线
33  {
34      p[0].push_back(a.x[0]-a.x[1]);
35      p[0].push_back(a.y[0]-a.y[1]);
36      p[0].push_back(a.z[0]-a.z[1]);
37      p[1].push_back(a.x[2]-a.x[1]);
38      p[1].push_back(a.y[2]-a.y[1]);
39      p[1].push_back(a.z[2]-a.z[1]);
40      p[2].push_back(p[1][1]*p[0][2]-p[1][2]*p[0][1]);
41      p[2].push_back(p[1][0]*p[0][2]-p[1][2]*p[0][0]);
42      p[2].push_back(p[1][1]*p[0][0]-p[1][0]*p[0][1]);
43      return (abs(p[2][0]) <= 1e-8 && abs(p[2][1]) <= 1e-8 && abs(p[2][2]) <= 1e-8);
44  }
45  double distance(vector<double> aa, vector<double> bb)//两点距离
46  {
47      return sqrt((aa[0]-bb[0])*(aa[0]-bb[0]) + (aa[1]-bb[1])*(aa[1]-bb[1])+(aa[2]-bb[2])*(aa[2]-bb[2]));
48  }
49  vector<double> edge_triangle(vector<double> aa, vector<double> bb, vector<double> cc)//由三角形三点坐标求三边长
50  {
51      vector<double> dis_triangle;
52      dis_triangle.push_back(distance(aa, bb));
53      dis_triangle.push_back(distance(aa, cc));
54      dis_triangle.push_back(distance(cc, bb));
55      return dis_triangle;
56  }
57  double p_triangle(vector<double> xx)//海伦公式-p
58  {
59      return (xx[0]+xx[1]+xx[2])/2.0;
60  }
61  double S_triangle(double p, vector<double> xx)//海伦公式-s
62  {
63      return sqrt(p*(p-xx[0])*(p-xx[1])*(p-xx[2]));
64  }
65  double size_triangle(int c1, int c2, int c3)//由节点编号求三角形面积
66  {
67      vector<double> nn;
68      nn = edge_triangle({v[c1].x[0], v[c1].y[0], v[c1].z[0]},{v[c2].x[0], v[c2].y[0], v[c2].z[0]},{v[c3].x[0],
    v[c3].y[0], v[c3].z[0]});
69      double p = p_triangle(nn);
70      double res = S_triangle(p, nn);
```

```
71          return res;
72      }
73      double size_triangle_point(vector<double> c1, vector<double> c2,vector<double> c3)//由节点坐标求三角形面积
74      {
75          vector<double> nn;
76          nn = edge_triangle(c1, c2, c3);
77          double p = p_triangle(nn);
78          double res = S_triangle(p, nn);
79          return res;
80      }
81      vector<double> edge_triangle_id(int c1, int c2, int c3)//由节点编号求三边长
82      {
83          return edge_triangle({v[c1].x[0], v[c1].y[0], v[c1].z[0]},{v[c2].x[0], v[c2].y[0], v[c2].z[0]},{v[c3].x[0]
        , v[c3].y[0], v[c3].z[0]});
84      }
85      void dfs(int id)//深度优先搜索遍历
86      {
87          visList.push_back(id);
88          vis[id] = visList.size();
89          for (int i = 0; i < connect[id].size(); ++i) {
90              if (vis[connect[id][i]] == 0 && allow[id] != 0) {
91                  dfs(connect[id][i]);
92              }
93          }
94      }
95      res adjust_set(vector<int> E)//调整坐标以便于求解主轴近点（并按与主轴水平距离升序）
96      {
97          vector <vector<double> > V;
98          vector <pair<double, int> > Seq;
99          double Dis = 1e10;
100         int Flg = -1;
101         res Res = {{0,0},{0,0},{0,0}, 0};
102         //坐标变换 旋转逆矩阵（见论文）
103         for (int i = 0; i<v.size(); ++i) {
104             double x = v[i].x[0];
105             double y = v[i].y[0];
106             double z = v[i].z[0];
107             double xx = x*cos(al)*cos(th) - y*sin(al) + z*cos(al)*sin(th);
108             double yy = y*cos(al) + x*sin(al)*cos(th) + z*sin(al)*sin(th);
109             double zz = z*cos(th) - x*sin(th);
110             V.push_back({{xx,yy,zz}});
111         }
112         for (auto i:E) {
113             double dd = distance(V[i], {0, 0, -300});
```

```
114             if (dd < Dis) {
115                 Dis = dd;
116                 Flg = i;
117             }
118         }
119         //Flg 新中轴原点序号
120         for (int j = 0; j < 2226; ++j) {
121             vis[j] = 0;
122         }
123         visList.clear();
124         dfs(Flg);
125         Res.num[1] = visList.size();
126         for (int i = 1; i < visList.size(); ++i) {
127             double xx = V[visList[i]][0] - V[visList[0]][0];
128             double yy = V[visList[i]][1] - V[visList[0]][1];
129             double dis_dfs = sqrt(abs(xx*xx+yy*yy));
130             if (Res.dis[1] <= dis_dfs) {
131                 Res.id[1] = visList[i], Res.dis[1] = dis_dfs;
132             }
133         }
134         //新坐标顺序处理
135         for (int i = 0; i<V.size(); ++i) {
136             double xx = V[i][0] - V[Flg][0];
137             double yy = V[i][1] - V[Flg][1];
138             Seq.push_back({sqrt(abs(xx*xx+yy*yy)), i});
139         }
140         sort(Seq.begin(), Seq.end());
141         int rr=0;
142         for (int k = 0; k < E.size(); ++k) {
143             if (Seq[k].second != E[k]) {
144                 rr = k-1;
145                 break;
146             }
147             else allow1[E[k]] = k+1;
148         }
149         if (rr >= E.size()) rr = E.size()-1;
150         if (rr < 0) rr = 0;
151         Res.num[0] = rr, Res.id[0] = Seq[rr].second, Res.dis[0] = Seq[rr].first;
152         return Res;
153     }
154     int main() {
155         freopen("test1.in", "r", stdin);
156         freopen("answer.out","w",stdout);//将结果输出到 answer.out 文件
157         data a;
```

```
158      string s;
159      //读入数据 附件 1
160      for (int i = 0; i < 2226; ++i) {
161          cin >> a.s >> a.x[0] >> a.y[0] >> a.z[0];
162          v.push_back(a);
163          name2id[a.s] = i;
164      }
165  //    cout << v.size() << endl;
166      //读入数据 附件 2
167      for (int i=0; i<2226; ++i){
168          cin >> s;
169          for (int i = 0; i < v.size(); ++i) {
170              if (s == v[i].s) {
171                  cin >> v[i].x[1] >> v[i].y[1] >> v[i].z[1];
172                  cin >> v[i].x[2] >> v[i].y[2] >> v[i].z[2];
173                  v[i].check =  (check(v[i]));
174                  if (v[i].check == false) {
175                      printf("%d\n", i);
176                  }
177                  seq.push_back({sqrt(abs(v[i].x[0]*v[i].x[0]+v[i].y[0]*v[i].y[0])), i});
178  //                else {
179  //                    printf("yes %d\n", i);
180  //                }
181                  break;
182              }
183          }
184      }
185      //初始排序
186      sort(seq.begin(), seq.end());
187      for (int l = 0; l < seq.size(); ++l) {
188          v[seq[l].second].seq = l;
189      }
190      //读入数据 附件 3
191      freopen("test2.in", "r", stdin);
192      for (int i = 0; i < 4300; ++i) {
193          string s1, s2, s3;
194          int flg1 = 0, flg2 = 0;
195          vector<int> G;
196          G.clear();
197          cin >> s1 >> s2 >> s3;
198          G.push_back(name2id[s1]);
199          G.push_back(name2id[s2]);
200          G.push_back(name2id[s3]);
201          sort(G.begin(), G.end());
```

```
202             board.push_back({G[0], G[1], G[2]});
203             boardVerticle2id[G] = i;
204             double triSize = size_triangle(G[0], G[1], G[2]);
205             boardSize.push_back(triSize);
206             for (int k = 0; k < 3; ++k) {
207                 for (int j = 0; j < connect[G[k]].size(); ++j) {
208                     if (connect[G[k]][j] == G[(k+1)%3]) flg1 = 1;
209                     if (connect[G[k]][j] == G[(k+2)%3]) flg2 = 1;
210                     if (flg1 && flg2) break;
211                 }
212                 if (flg1 == 0) connect[G[k]].push_back(G[(k+1)%3]);
213                 if (flg2 == 0) connect[G[k]].push_back(G[(k+2)%3]);
214                 flg1 = flg2 = 0;
215             }
216         }
217         for (int i = 0; i < 2226; ++i) {
218             sort(connect[i].begin(), connect[i].end());
219         }
220         cout << board[4299][0] << " OK" << endl;
221         //开始求解
222         double P = 279.6;
223         for (double dz=299; dz<301; dz+=0.01) {//dz 坐标偏移量按步长枚举，可调节步长和范围
224     //        printf("%lf \n", dz);
225             //初始化
226             int flg = 0;
227             vector <int> e;
228             vector <int> sol;
229             vector <vector<double> > newLocation;
230             vector <double> disChange;
231             e.clear(), sol.clear();
232             P = abs(dz-160.2)*2;
233             for (int i = 0; i < 2226; ++i) allow[i] = allow1[i] = 0;
234             //求解变化的新点坐标，在期望的抛物面方程上
235             for (int i = 0; i < 2226; ++i) {
236                 double x0 = v[i].x[0];
237                 double y0 = v[i].y[0];
238                 double z0 = v[i].z[0];
239                 double N = (v[i].x[1]-v[i].x[0]);
240                 double M = (v[i].y[1]-v[i].y[0]);
241                 double Q = (v[i].z[1]-v[i].z[0]);
242                 double A = - 2*M*N*cos(al)*sin(al)- 2*N*Q*cos(al)*cos(th)*sin(th)- 2*M*Q*sin(al)*cos(th)*sin(th)+
    2*M*N*cos(al)*sin(al)*cos(th)*cos(th)+ M*M*cos(al)*cos(al)+ N*N*sin(al)*sin(al)+ Q*Q*sin(th)*sin(th)+ N*N*cos(
    al)*cos(al)*cos(th)*cos(th)+ M*M*sin(al)*sin(al)*cos(th)*cos(th);
```

```
243             double B = 2*M*y0*sin(al)*sin(al)*cos(th)*cos(th)- 2*M*x0*cos(al)*sin(al)- 2*N*y0*cos(al)*sin(al)-
      2*N*P*cos(al)*sin(th)- 2*M*P*sin(al)*sin(th)- 2*N*z0*cos(al)*cos(th)*sin(th)- 2*Q*x0*cos(al)*cos(th)*sin(th)-
      2*M*z0*sin(al)*cos(th)*sin(th)- 2*Q*y0*sin(al)*cos(th)*sin(th)+ 2*M*x0*cos(al)*sin(al)*cos(th)*cos(th)+ 2*N*y
      0*cos(al)*sin(al)*cos(th)*cos(th)+ 2*N*x0*cos(al)*cos(al)*cos(th)*cos(th)+ 2*N*x0*sin(al)*sin(al)- 2*P*Q*cos(t
      h)+ 2*M*y0*cos(al)*cos(al)+ 2*Q*z0*sin(th)*sin(th);
244             double C = -
      2*dz*P+ y0*y0*cos(al)*cos(al)+ x0*x0*sin(al)*sin(al)+ z0*z0*sin(th)*sin(th)- 2*P*z0*cos(th)+ x0*x0*cos(al)*cos
      (al)*cos(th)*cos(th)+ y0*y0*sin(al)*sin(al)*cos(th)*cos(th)- 2*P*y0*sin(al)*sin(th)- 2*x0*y0*cos(al)*sin(al)-
      2*P*x0*cos(al)*sin(th)+ 2*x0*y0*cos(al)*sin(al)*cos(th)*cos(th)- 2*x0*z0*cos(al)*cos(th)*sin(th)- 2*y0*z0*sin(
      al)*cos(th)*sin(th);
245 //            cout << N << " " << M << " " << Q << endl;
246             double Delta = B*B-4*A*C;
247 //            cout <<"A B C " <<  A << " " << B << " " << C << " "<<Delta<<  endl;
248             double x1, x2;
249             vector <double> ans[2], ori;
250             vector <double> dis;
251             ans[0].clear(), ans[1].clear(), ori.clear();
252             dis.clear();
253             if (abs(A) < 1e-9) {
254 //                cout << "A0->"<<abs(v[i].z[0]+dz) << " " << v[i].seq << endl;
255                 if (abs(v[i].z[0]+dz) <= 0.6) {
256                     e.push_back(i);
257                     allow[i] = e.size();
258                     newLocation.push_back({0,0,-dz});
259                     disChange.push_back(-(v[i].z[0]+dz));
260                     sol.push_back(v[i].seq);
261                     flg++;
262                 }
263                 continue;
264             }
265             //求解得到新的坐标点
266             if (Delta > 0) {
267                 x1 = (-B+sqrt(Delta))/(2*A);
268                 x2 = (-B-sqrt(Delta))/(2*A);
269                 ori.push_back(v[i].x[0]);
270                 ori.push_back(v[i].y[0]);
271                 ori.push_back(v[i].z[0]);
272                 ans[0].push_back(v[i].x[0]+N*x1);
273                 ans[0].push_back(v[i].y[0]+M*x1);
274                 ans[0].push_back(v[i].z[0]+Q*x1);
275                 ans[1].push_back(v[i].x[0]+N*x2);
276                 ans[1].push_back(v[i].y[0]+M*x2);
277                 ans[1].push_back(v[i].z[0]+Q*x2);
278                 dis.push_back(distance(ans[0], ori));
```

```
279                 dis.push_back(distance(ans[1], ori));
280                 //满足要求则记录，此处允许精度误差 1e-5
281                 if (dis[0] <= 0.60+1e-5 || dis[1] <= 0.6+1e-5) {
282                     flg++;
283                     e.push_back(i);
284                     allow[i] = e.size();
285                     if (dis[0] <= 0.60+1e-5) {
286                         newLocation.push_back(ans[0]);
287                         double neg = 1;
288                         if (distance(ori,{0,0,0}) >= distance(ans[0], {0,0,0})) neg = 1;
289                         else neg = -1;
290                         disChange.push_back(neg*dis[0]);
291                     }
292                     else {
293                         newLocation.push_back(ans[1]);
294                         double neg = 1;
295                         if (distance(ori,{0,0,0}) >= distance(ans[1], {0,0,0})) neg = 1;
296                         else neg = -1;
297                         disChange.push_back(neg*dis[1]);
298                     }
299                     sol.push_back(v[i].seq);
300                 }
301             }
302         }
303         if (flg)
304         {
305             /* 计算许可节点两两最大距离并排序 */
306             vector<double> dis_set;
307             for (int k = 0; k < e.size(); ++k) {
308                 for (int j = k+1; j < e.size(); ++j) {
309                     int t1 = e[k], t2 = e[j];
310                     double dd = (v[t1].x[0]-v[t2].x[0])*(v[t1].x[0]-v[t2].x[0])+(v[t1].y[0]-
    v[t2].y[0])*(v[t1].y[0]-v[t2].y[0]);
311                     dis_set.push_back(sqrt(dd));
312                 }
313             }
314             sort(dis_set.begin(), dis_set.end());
315             /* 排序后找到连续最远节点距离 */
316             res B;
317             B = adjust_set(e);
318             //标化输出 输出关键数据参数
319             cout << "" << dz << " " ;
320 //            cout << dis_set.back() <<  endl << v[B.id[0]].s << " "<< B.num[0] << " " << B.dis[0] << " " <<en
    dl;
```

```
321 //            cout << v[B.id[1]].s << " "<< B.num[1] << " " << B.dis[1] << " " <<endl;
322 //            cout << "Err = " << B.err << endl;
323             for (int j = 0; j <= B.num[0]; ++j) {
324                 allow1[e[j]] = j+1;
325             }
326             //测试输出 按照允许的排序
327 //            cout << sol.size() << endl;
328 //            for (int j = 0; j < sol.size(); ++j) {
329 //                cout << ""<<v[seq[sol[j]].second].s << " " << sol[j] << endl;
330 //            }
331 //            putchar(10);
332             //专用输出 用于导出数据（节点名称 原坐标 新坐标 伸缩量变化）
333 //            for (int k = 0; k < 2226; ++k) {
334 //                if (allow[k] == 0) {
335 //                    printf("%s %.3lf %.3lf %.3lf %.3lf %.3lf %.3lf 0.000\n", v[k].s.c_str(), v[k].x[0] , v[k
    ].y[0] ,v[k].z[0], v[k].x[0] , v[k].y[0] ,v[k].z[0]);
336 //                }
337 //                else {
338 //                    int id = allow[k]-1;
339 //                    printf("%s %.3lf %.3lf %.3lf %.3lf %.3lf %.3lf %.3lf\n", v[k].s.c_str(),  v[k].x[0] , v[
    k].y[0] ,v[k].z[0], newLocation[id][0] , newLocation[id][1] ,newLocation[id][2], disChange[id]);
340 //                }
341 //            }
342             //误差计算与判断
343             double ERR = 0;
344             for (int k = 0; k < 4300; ++k) {
345                 int id[3] = {board[k][0], board[k][1], board[k][2]};
346                 vector<double> p_id[3];
347                 for (int l = 0; l < 3; ++l) {
348                     if (allow1[id[l]] == 0) {
349                         p_id[l].push_back(v[id[l]].x[0]);
350                         p_id[l].push_back(v[id[l]].y[0]);
351                         p_id[l].push_back(v[id[l]].z[0]);
352                     }
353                     else {
354                         int tt = allow1[id[l]]-1;
355                         p_id[l].push_back(newLocation[tt][0]);
356                         p_id[l].push_back(newLocation[tt][1]);
357                         p_id[l].push_back(newLocation[tt][2]);
358                     }
359                 }
360                 vector<double> oldDis = edge_triangle_id(id[0], id[1], id[2]);
361                 vector<double> newDis = edge_triangle(p_id[0], p_id[1], p_id[2]);
362                 vector<double> newErr;
```

```
                newErr.push_back((newDis[0]-oldDis[0])/oldDis[0]*100.0);
                newErr.push_back((newDis[1]-oldDis[1])/oldDis[1]*100.0);
                newErr.push_back((newDis[2]-oldDis[2])/oldDis[2]*100.0);
                double errRes = max(newErr[0]*newErr[0], max(newErr[1]*newErr[1], newErr[2]*newErr[2]));
                if (errRes >= ERR) ERR = errRes;
            }
            printf(" %lf\n", ERR);
            //测试输出 遍历结果
//            printf("%d\n", visList.size());
//            for (int k = 0; k < visList.size(); ++k) {
//                printf("%d\n", visList[k]);
//            }

        }
    }
    return 0;
}
```

辅助程序 2（Auxiliary_program_sub.cpp）

```
#include <bits/stdc++.h>
using namespace std;

/*
 * 副程序 用于问题 3
 * 数学方法-大量模拟与计算
 * 部分注释内容略，详见主程序源文件
 * 与主程序源文件定义一致相同
 */
struct data{
    string s;
    double x[3], y[3], z[3];
    bool check;
    int seq;
};
struct res{
    int num[2], id[2];//num 连续数目 id 最远序号
    double dis[2], err;//最远距离 误差
    //方法 1 直接按距离顺序取样 方法 2 按照相邻点扩展取样
};
// Modify as needed
//调整随机范围 x，y 的阈值
constexpr int MIN = -155;
constexpr int MAX = 155;
vector <pair<double, int> > seq;
```

```
vector <data> v;
vector <double> p[3];
map <string, int> name2id;
map <vector<int>, int> boardVerticle2id;
map <vector<int>, int>::iterator iter;
vector <int> connect[2226];
vector <vector<int> > board;
vector <double> boardSize;
vector <int> visList;
vector<double> flat_dir[5000];
vector<double> tar;
int allow[2227], vis[2227];
double R = 300.00, F = 0.466*300;//R，F 见题目要求
double injectHeight = 0;//入射高度即 z 轴
double al = 36.795*M_PI/180, th = (90-78.169)*M_PI/180, be = 78.169*M_PI/180;//入射角参数
double Lim = 1e-3;//许可误差范围
vector<int> allowList;
double X0=0.16418, Y0=0.12280;//直线参数，常量
//double al = 0.0001, th = 0.0001, be = M_PI/2;

double distance(vector<double> aa, vector<double> bb)
{
    return sqrt((aa[0]-bb[0])*(aa[0]-bb[0]) + (aa[1]-bb[1])*(aa[1]-bb[1])+(aa[2]-bb[2])*(aa[2]-bb[2]));
}
vector<double> edge_triangle(vector<double> aa, vector<double> bb, vector<double> cc)
{
    vector<double> dis_triangle;
    dis_triangle.push_back(distance(aa, bb));
    dis_triangle.push_back(distance(aa, cc));
    dis_triangle.push_back(distance(cc, bb));
    return dis_triangle;
}
double p_triangle(vector<double> xx)
{
    return (xx[0]+xx[1]+xx[2])/2.0;
}
double S_triangle(double p, vector<double> xx)
{
    return sqrt(p*(p-xx[0])*(p-xx[1])*(p-xx[2]));
}
double size_triangle(int c1, int c2, int c3)
{
    vector<double> nn;
```

```
69        nn = edge_triangle({v[c1].x[0], v[c1].y[0], v[c1].z[0]},{v[c2].x[0], v[c2].y[0], v[c2].z[0]},{v[c3].x[0],
      v[c3].y[0], v[c3].z[0]});
70        double p = p_triangle(nn);
71        double res = S_triangle(p, nn);
72        return res;
73    }
74    double randDouble()//随机给出一个双精度数据在要求范围内
75    {
76        setprecision(10);
77        return MIN + (double)(rand()) / ((double)(RAND_MAX/(MAX - MIN)));
78    }
79    double point_mul(vector<double> a, vector<double> b)//矢量点乘
80    {
81        return a[0]*b[0]+a[1]*b[1]+a[2]*b[2];
82    }
83    vector<double> normalize(vector<double> I)//矢量归一化
84    {
85        double s = sqrt(I[0]*I[0]+I[1]*I[1]+I[2]*I[2]);
86        return {I[0]/s, I[1]/s, I[2]/s};
87    }
88    vector<double> getFlatDir(int p1, int p2, int p3)//得到平面的法向量
89    {
90        double x1=v[p1].x[0], y1 = v[p1].y[0],z1 = v[p1].z[0];
91        double x2=v[p2].x[0], y2 = v[p2].y[0],z2 = v[p2].z[0];
92        double x3=v[p3].x[0], y3 = v[p3].y[0],z3 = v[p3].z[0];
93        double aa=(y2-y1)*(z3-z1)-(y3-y1)*(z2-z1);
94        double bb=(z2-z1)*(x3-x1)-(z3-z1)*(x2-x1);
95        double cc=(x2-x1)*(y3-y1)-(x3-x1)*(y2-y1);
96        double s = sqrt(aa*aa+bb*bb+cc*cc);
97        return {aa/s, bb/s, cc/s};
98    }
99    //求直线与平面交点，即反射点（原来的设计）
100   //vector<double> getReflexPoint(vector<double> I, vector<double> idir, int p1, int p2, int p3)
101   //{
102   //    double x0=idir[0], y0 = idir[1], z0 = idir[2];
103   //    double x1=v[p1].x[0], y1 = v[p1].y[0], z1 = v[p1].z[0];
104   //    double x2=v[p2].x[0], y2 = v[p2].y[0], z2 = v[p2].z[0];
105   //    double x3=v[p3].x[0], y3 = v[p3].y[0], z3 = v[p3].z[0];
106   //    double x4=I[0], y4 = I[1], z4 = I[2];
107   //    double A=y1*z2-y2*z1-y1*z3+y3*z1+y2*z3-y3*z2;
108   //    double B=-x1*z2+x2*z1+x1*z3-x3*z1-x2*z3+x3*z2;
109   //    double C=x1*y2-x2*y1-x1*y3+x3*y1+x2*y3-x3*y2;
110   //    double D=-x1*y2*z3+x1*y3*z2+x2*y1*z3-x2*y3*z1-x3*y1*z2+x3*y2*z1;
111   //    double t = (A*x4+B*y4+C*z4+D)/(A*x0+B*y0+C*z0);
```

```
//    return {x4+x0*t, y4+y0*t, z4+z0*t};
//}
vector<double> getReflexPoint(vector<double> I, vector<double> idir, int p1, int p2, int p3)//求直线与平面交点，即反射点
{
/// 求一条直线与平面的交点
/// planeVector 平面的法线向量
/// planePoint  平面经过的一点坐标
/// lineVector  直线的方向向量
/// linePoint   直线经过的一点坐标
    vector<double> linePoint = I;
    vector<double> lineVector = idir;
    vector<double> planeVector = getFlatDir(p1, p2, p3);
    vector<double> planePoint;
    planePoint.push_back(v[p1].x[0]);planePoint.push_back(v[p1].y[0]);planePoint.push_back(v[p1].z[0]);
    vector<double> returnResult(3);
    double vp1, vp2, vp3, n1, n2, n3, v1, v2, v3, m1, m2, m3, t,vpt;
    vp1 = planeVector[0];
    vp2 = planeVector[1];
    vp3 = planeVector[2];
    n1 = planePoint[0];
    n2 = planePoint[1];
    n3 = planePoint[2];
    v1 = lineVector[0];
    v2 = lineVector[1];
    v3 = lineVector[2];
    m1 = linePoint[0];
    m2 = linePoint[1];
    m3 = linePoint[2];
    vpt = v1 * vp1 + v2 * vp2 + v3 * vp3;
//首先判断直线是否与平面平行
    if (abs(vpt) > 1e-5)
    {
        t = ((n1 - m1) * vp1 + (n2 - m2) * vp2 + (n3 - m3) * vp3) / vpt;
        returnResult[0] = m1 + v1 * t;
        returnResult[1] = m2 + v2 * t;
        returnResult[2] = m3 + v3 * t;
    }
    return returnResult;
}
bool pointOnTriangle(vector<double> P, int p1, int p2, int p3) //求解点是否在三角形内部
{
    double x0 = P[0], y0 = P[1], z0 = P[2];
    double x1 = v[p1].x[0], y1 = v[p1].y[0], z1 = v[p1].z[0];
```

```
155       double x2 = v[p2].x[0], y2 = v[p2].y[0], z2 = v[p2].z[0];
156       double x3 = v[p3].x[0], y3 = v[p3].y[0], z3 = v[p3].z[0];
157       vector<double> AB, AP, AC;
158       AB.push_back(x2 - x1);
159       AB.push_back(y2 - y1);
160       AB.push_back(z2 - z1);
161       AC.push_back(x3 - x1);
162       AC.push_back(y3 - y1);
163       AC.push_back(z3 - z1);
164       AP.push_back(x0 - x1);
165       AP.push_back(y0 - y1);
166       AP.push_back(z0 - z1);
167       double A, B, C, D, E, F, U, V;
168       A = point_mul(AB, AB);
169       B = point_mul(AC, AC);
170       C = point_mul(AB, AC);
171       D = point_mul(AB, AP);
172       E = point_mul(AC, AP);
173       F = A * B - C * C;
174       U = (A * E - C * D) / F;
175       V = (B * D - C * E) / F;
176       return (U >= 0 && U <= 1 && V >= 0 && V <= 1 && U+V<=1);
177   }
178   vector<double> getReflexDir(vector<double> Idir, vector<double> Ndir)//求反射法向量
179   {
180       double x=Idir[0], y = Idir[1], z = Idir[2];
181       double x1=Ndir[0], y1 = Ndir[1], z1 = Ndir[2];
182       return normalize({x-2*x*x1*x1, y-2*y*y1*y1, z-2*z*z1*z1});
183   }
184   vector<double> getPointOnBall(vector<double> I, vector<double> Idir)//求与球的交点
185   {
186       double x0=Idir[0], y0 = Idir[1], z0 = Idir[2];
187       double x4=I[0], y4 = I[1], z4 = I[2];
188       double A = x0*x0+y0*y0+z0*z0;
189       double B = 2*x0*x4+2*y0*y4+2*z0*z4;
190       double C = x4*x4+y4*y4+z4*z4-(R-F)*(R-F);
191       double delta = B*B-4*A*C;
192       if (delta < 0) return {0, 0, 0};
193       if (abs(delta) <= 1e-6) {
194           double t = -B/(2*A);
195           return {x4+x0*t, y4+y0*t, z4+z0*t};
196       }
197       delta = sqrt(delta);
198       double t[2];
```

```
199       vector<double> p[2];
200       t[0] = (-B+delta)/(2*A); t[1] = (-B-delta)/(2*A);
201       p[0].push_back(x4+x0*t[0]); p[0].push_back(y4+y0*t[0]); p[0].push_back(z4+z0*t[0]);
202       p[1].push_back(x4+x0*t[1]); p[1].push_back(y4+y0*t[1]); p[1].push_back(z4+z0*t[1]);
203       if (distance(p[0], tar) <= distance(p[1], tar)) return p[0];
204       else return p[1];
205   }
206   int simulate(double x, double y, double z)//模拟
207   {
208       vector<double> ori;
209       vector<double> ori_dir;
210       vector<double> ref;
211       vector<double> ref_dir;
212       vector<double> des;
213       vector<double> rev;
214       double minDis = 10000000;
215       int rev_i;
216       ori.clear(); ori_dir.clear(); ref.clear(); ref_dir.clear(); des.clear();
217       ori.push_back(x), ori.push_back(y), ori.push_back(z);
218       ori_dir.push_back(-cos(be)*cos(al)), ori_dir.push_back(-cos(be)*sin(al)), ori_dir.push_back(-sin(be));
219   //    printf("(%lf %lf %lf) ", x, y, z);
220   //    printf("(%lf %lf %lf)\n", ori_dir[0], ori_dir[1], ori_dir[2]);
221       for (int i = 0; i < 4300; ++i) {
222           ref = getReflexPoint(ori, ori_dir, board[i][0], board[i][1], board[i][2]);
223           double tmp = distance(ref, {0, 0, 0});
224           if (tmp <= minDis) {
225               minDis = tmp;
226               rev = ref;
227               rev_i = i;
228           }
229   //        printf("%d %lf %lf %lf\n", i, ref[0], ref[1], ref[2]);
230           if (pointOnTriangle(ref, board[i][0], board[i][1], board[i][2])) {
231               ref_dir = getReflexDir(ori_dir, flat_dir[i]);
232               des = getPointOnBall(ref, ref_dir);
233               if (distance(des, tar) <= 2+1e-4)
234               {
235   //                printf(" %lf %lf %lf ", x, y, z);printf("%lf %lf %lf ", ref[0], ref[1], ref[2]);
236   //                printf("%lf %lf %lf ", v[board[i][0]].x[0], v[board[i][0]].y[0],v[board[i][0]].z[0]);
237   //                printf("%lf %lf %lf ", v[board[i][1]].x[0], v[board[i][1]].y[0],v[board[i][1]].z[0]);
238   //                printf("%lf %lf %lf ", v[board[i][2]].x[0], v[board[i][2]].y[0],v[board[i][2]].z[0]);
239   //                printf("%lf %lf %lf %d 1\n", des[0], des[1], des[2], i);
240                   return 1;
241               }
242               else
```

```
243             {
244 //                printf(" %lf %lf %lf ", x, y, z);printf("%lf %lf %lf ", ref[0], ref[1], ref[2]);
245 //                printf("%lf %lf %lf ", v[board[i][0]].x[0], v[board[i][0]].y[0],v[board[i][0]].z[0]);
246 //                printf("%lf %lf %lf ", v[board[i][1]].x[0], v[board[i][1]].y[0],v[board[i][1]].z[0]);
247 //                printf("%lf %lf %lf ", v[board[i][2]].x[0], v[board[i][2]].y[0],v[board[i][2]].z[0]);
248 //                printf("%lf %lf %lf %d 2\n", des[0], des[1], des[2], i);
249                 return 2;
250             }
251         }
252     }
253 //    printf(" %lf %lf %lf ", x, y, z);printf("%lf %lf %lf ", rev[0], rev[1], rev[2]);
254 //    int i = rev_i;
255 //    printf("%lf %lf %lf ", v[board[i][0]].x[0], v[board[i][0]].y[0],v[board[i][0]].z[0]);
256 //    printf("%lf %lf %lf ", v[board[i][1]].x[0], v[board[i][1]].y[0],v[board[i][1]].z[0]);
257 //    printf("%lf %lf %lf ", v[board[i][2]].x[0], v[board[i][2]].y[0],v[board[i][2]].z[0]);
258 //    printf("%lf %lf %lf %d 0\n", ori_dir[0], ori_dir[1], ori_dir[2], rev_i);
259 //    putchar(10);
260     return 0;
261 }
262 int main() {
263     freopen("surface0.in", "r", stdin);
264     freopen("res.out", "w", stdout);
265     data a;
266     string s;
267     for (int i = 0; i < 2226; ++i) {
268         cin >> a.s >> a.x[0] >> a.y[0] >> a.z[0];
269         v.push_back(a);
270         name2id[a.s] = i;
271     }
272     freopen("connect.in", "r", stdin);
273     for (int i = 0; i < 4300; ++i) {
274         string s1, s2, s3;
275         int flg1 = 0, flg2 = 0;
276         vector<int> G;
277         G.clear();
278         cin >> s1 >> s2 >> s3;
279         G.push_back(name2id[s1]);
280         G.push_back(name2id[s2]);
281         G.push_back(name2id[s3]);
282         sort(G.begin(), G.end());
283         board.push_back({G[0], G[1], G[2]});
284         flat_dir[i] = getFlatDir(G[0], G[1], G[2]);
285         boardVerticle2id[G] = i;
286         double triSize = size_triangle(G[0], G[1], G[2]);
```

```
287             boardSize.push_back(triSize);
288             for (int k = 0; k < 3; ++k) {
289                 for (int j = 0; j < connect[G[k]].size(); ++j) {
290                     if (connect[G[k]][j] == G[(k+1)%3]) flg1 = 1;
291                     if (connect[G[k]][j] == G[(k+2)%3]) flg2 = 1;
292                     if (flg1 && flg2) break;
293                 }
294                 if (flg1 == 0) connect[G[k]].push_back(G[(k+1)%3]);
295                 if (flg2 == 0) connect[G[k]].push_back(G[(k+2)%3]);
296                 flg1 = flg2 = 0;
297             }
298         }
299     //    freopen("allow.in", "r", stdin);
300     //    int T;
301     //    scanf("%d", &T);
302     //    while (T--) {
303     //        int x;
304     //        scanf("%d", &x);
305     //        allowList.push_back(x);
306     //    }
307     //    for (int i = 0; i < board.size(); ++i) {
308     //        printf("\n%d\n", i);
309     //        printf("%lf %lf %lf\n", v[board[i][0]].x[0], v[board[i][0]].y[0],v[board[i][0]].z[0]);
310     //        printf("%lf %lf %lf\n", v[board[i][1]].x[0], v[board[i][1]].y[0],v[board[i][1]].z[0]);
311     //        printf("%lf %lf %lf\n", v[board[i][2]].x[0], v[board[i][2]].y[0],v[board[i][2]].z[0]);
312     //    }
313     //    printf("%d\n", board.size());
314         for (int i = 0; i < 2226; ++i) {
315             sort(connect[i].begin(), connect[i].end());
316         }
317         tar.clear();
318         tar.push_back(-(R-F)*cos(be)*cos(al)), tar.push_back(-(R-F)*cos(be)*sin(al)), tar.push_back(-(R-
    F)*sin(be));
319     //    printf("%lf %lf %lf\n", tar[0], tar[1], tar[2]);
320         std::srand(std::time(nullptr));
321         vector<int> cnt(3);
322         for (int i = 0; i < 1000000; ++i) {
323             double aa = randDouble();
324             double bb = randDouble();
325             double fun = aa*aa+bb*bb-(X0*aa+Y0*bb)*(X0*aa+Y0*bb)-150*150;
326             if (fun <= 1e-5) {
327                 int g = simulate(aa, bb, injectHeight);
328                 ++cnt[g];
329                 cout << g << endl;
```

```
330             }
331         }
332         simulate(0, 0, 0);
333         printf("%d %d %d\n", cnt[0], cnt[1], cnt[2]);
334         printf("%lf\n", (double)(1.0*cnt[1]/(1.0*cnt[1]+cnt[2])));
335         return 0;
336     }
```

10 致谢

感谢哈尔滨工业大学（深圳）对于同学参加数学建模竞赛的大力支持。感谢梁慧老师、陈国廷老师、陶菊春老师、包益欣老师、谢秉磊老师、耿平老师、马永辉老师、张进老师、应梦娴老师对我们深刻的指导与帮助。