基于理想抛物面优化的 FAST 工作面模型

摘要

FAST 是世界最大单口径射电望远镜，素有“天眼”之称，对其工作面进行优化在天文学上具有重大意义。在本题中，首先考虑连续的理想模型，算出两种不同的优化模型，再根据理想点法确定理想抛物面。接着使用模拟退火算法对该理想抛物面作逼近，考虑让残差最小，最后计算了这种逼近模型下的信号接收比，并与基准面作比较。

在问题一的讨论中，我们得出了不同意义下的最优理想抛物面，其中对于极端行程最优的情况通过纯理论分析得出焦距 $h = 140.322m$（在本文题中，系统对称性良好，所有抛物面的形状都可以由焦距和顶点位置唯一确定）。对于总行程最优的情况，我们定义了误差面积 S，它是一组可以初等表示的积分，使用 mathematica 程序计算出使 S 最小的精度较高的数值结果 $h = 140.364m$，最小误差面积为 152.692m。接着假定存在一种最理想的抛物面，使得促动器的极端行程和总行程都最优，然后使用软件找到与该理想点距离最近的情况 $h = 140.341m$。由于搜索范围很小，我们直接采取小步长枚举的办法搜索最优解，不会陷入局部最优。

在问题二中，我们首先使用旋转矩阵将一的结果旋转至实际需要的位置，焦距 140.341m 不变，抛物线顶点坐标变为 $(-49.378, -36.933, -294.366)$，接着理论分析了可动节点的范围，使得下一步计算时考虑的变量充分降维；接着使用模拟退火的算法搜索最小残差，将可动节点乘以旋转矩阵，划归到如题一的对称系统中，大大简化计算中的抛物面方程。使系统不断向节点均位于抛物面的情况逼近，得到调整后的节点位置，结果存于附件四。

在问题三中，我们脱离坐标系，使用向量表示整个反射系统，以面积来正比反映信息量。使用割点逼近的算法求得每个反射面板的信号贡献率，乘以接收到的信号总量，也就是在信号方向垂直平面的投影面积，得到单块面板的信息贡献，最终求和得到抛物面的信息接收比为 1.059%，介于基准面接受比 0.828% 和理想抛物面接受比 1.148% 之间。

本题中我们所建立的一系列模型均能够满足题目要求，层层递进、环环相扣，具有满意的精度与稳定性。且本文建立的研究方法既可以通过添加参考点的方式优化逼近，同时也适用于不同口径工作面的研究，对 FAST 系统的进一步优化具有实际的应用价值。

关键字： 旋转抛物面　理想点法　坐标系旋转　模拟退火　向量表示

目录

一、 问题重述

天眼 FAST 是我国具有自主知识产权的 500 米口径球面射电望远镜，对我国天体运动的观测和科学前沿的突破具有重要作用。FAST 主要由主动反射面、馈源舱（信号接收系统）以及相关的控制、测量和支撑系统组成，其中主动反射面是 FAST 的创新点之一。主动反射面由主索网、反射面板、下拉索、促动器以及周边支承结构组成，其实质是跟随所观测天体的运动调节部分促动器顶端的伸缩，促动器顶端连接下拉索，每个下拉索连接一个主索网上的主索节点，实现主索网的形态控制。

主动反射面可分为基准态和工作态。基准态时反射面为一半径固定为 300 米，口径为 500 米的确定球面；工作态时被来自目标天体的平行电磁波照射范围的部分主动反射面调节到指定的抛物面位置，使平行电磁波经反射面反射后始终汇聚到处于焦面上移动的馈源舱的有效区域，其中焦面所在球与基准球面为同心球且两球面的半径差为 $F = 0.466R$，馈源舱的有效区域为直径 1 米的中心圆盘 [1]。

对于工作态时反射面的调节，每个主索节点下连接的下拉索长度不变，通过调节促动器顶端来完成下拉索的调节。促动器沿基准球面径向安装，底端固定在地面，顶端可沿基准球面径向伸缩，沿基准球面径向趋向球心方向为正方向，其伸缩范围为 $-0.6 \sim +0.6$ 米。

1.1 问题一的描述

当观测天体 S 的方位角 $\alpha = 0°$, 仰角 $\beta = 90°$ 时，确定工作态时的理想抛物面。

1.2 问题二的描述

当观测天体 S 的方位角 $\alpha = 36.795°$, 仰角 $\beta = 78.169°$ 时，确定工作态时的理想抛物面以及其顶点坐标。建立反射面板的调节模型，使反射面尽量贴近理想抛物面，确定需要调节的反射面中主索节点编号、位置坐标以及连接的各个促动器的伸缩量结果。

1.3 问题三的描述

根据问题的调节模型，计算调节后馈源舱的接收比，并与基准反射球面的接收比作比较，其中接收比是有效区域收到的反射信号与工作态时反射面的反射信号之比。

二、 模型的假设

- 只考虑各个促动器的径向伸缩，下拉索和促动器共线，不考虑其横向移动；
- 不考虑下拉索的最大承受力，将其看作理想轻绳；

- 只考虑各个促动器的伸缩量对反射面贴近理想抛物面形状的影响，不考虑外部影响，如风力、温度作用等。
- 假设馈源舱只接收工作面采集到的信号，其余部分反射的信号忽略不计；
- 假设在 300 米口径范围内电磁波照射充足且均匀，截面上单位面积信息量恒定；
- 假设电磁波在反射和传播过程中没有信号的衰减，即不考虑空气和反射面的吸收、折射。

三、 符号说明

符号	意义
R	基准球面半径
F	基准球面与焦面的半径差
α	方位角
β	仰角
h	焦距
Γ_i	理想抛物面方程
d_h	焦距为 h 时理想抛物线的最大偏差值
$Z^{\pm}$,H_i	理想抛物线与基准圆弧的若干交点
S	抛物线与圆弧的误差面积
a	搜索时抛物线与圆弧允许的最大偏差
h_a	在 a 的条件下允许的焦距 h
$R_n(\theta)$	沿 $\vec{n}$ 旋转 θ 角的旋转矩阵
l_i	编号为 i 的节点径向移动距离
E	优化目标函数 σl_i^2
T	模拟退火算法中的温度
$\vec{A}$	沿信号入射方向的单位向量
σ	向量单位化函数
a,b,c	抽象出的反射面板三个节点的坐标（与之前的 a 不同）
dt	切分步长
r	馈源舱接收平面的半径

四、 问题分析

4.1 问题一的模型建立与求解

针对问题一，首先通过处理附件中所给数据建立坐标系，并通过计算得到较为精确的基准球面半径 R，再寻找理想抛物面的方程。由于该理想抛物面是近似旋转抛物面，我们可以简化在二维平面中找到理想抛物线，再通过绕轴旋转得出三维理想旋转抛物面方程。因为馈源舱所在球面与基准球面的半径差为定值，且工作抛物面的口径也为定值，因此在待观测天体移动时，理想抛物面只改变顶点位置，不改变其形状。根据题目假设“目标天体的电磁波信号为直线传播”，易知理想抛物面顶点、待观测天体和基准球面的球心 O 共线。在寻找理想抛物面的方程时，因为抛物面顶点所在的节点可以径向移动 $-0.6 \sim +0.6$ 米，焦距 h 可能会因此发生变化，所以需要通过优化算法寻求最佳的焦距 h，以确定最优的理想抛物面方程。

4.1.1 坐标系及基准球面半径的确定

根据附件中给定的节点位置坐标，可以确定题目中所建立的坐标系，并拟合出 FAST 基准态时的三维视图。

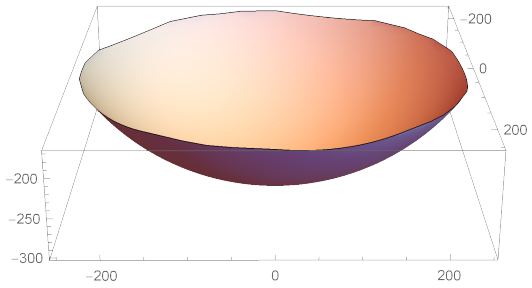

图 1　FAST 基准态三维视图

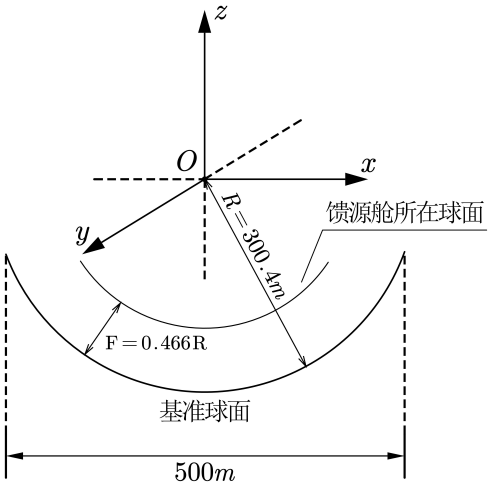

图 2　坐标轴下的基准态示意图

同时，由于题目所给的基准球面半径 $R \approx 300$ 米不够精确，需要进一步分析所给数据。由于已知四点坐标可以确定球的方程，因此程度每次任取四个所给节点的坐标来计算所成基准球面的方程，任取 1000 次取平均值后求得基准球面的半径为 300.4009 米。因此，在后面的讨论中，我们认为 $R = 300.4m$。

4.1.2 理想抛物面方程的确定

对于这样一套精密的系统，在确定“理想”抛物面时，总是希望它的运行更加平稳、柔和 [2]，因此我们首先选定的优化指标是在连续条件下的最大偏移值。

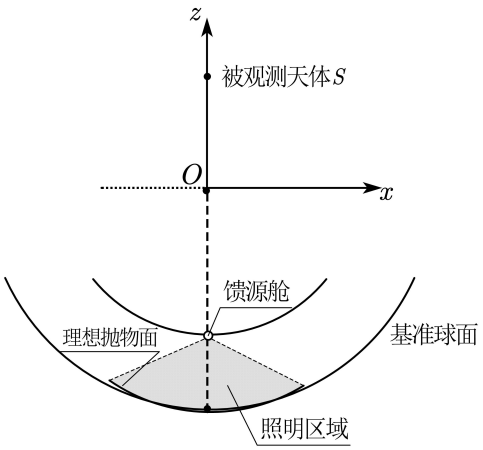

图 **3**　工作态示意图

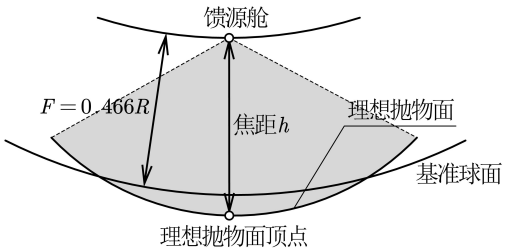

图 **4**　理想抛物面顶点位置局部图

整个系统关于 z 轴对称，因此只考虑过抛物线旋转轴的截面，将问题划归到 xz 二维平面上处理，由于焦点已知，只需求得抛物线顶点的位置就可得理想抛物线和理想旋转抛物面的方程。设抛物线的焦距为 h，则其截线方程为

$$y(x) = \frac{x^2}{4h} - (h + 160.414) \tag{1}$$

根据模型假设，节点只沿径向移动，因此只要考虑抛物线上每个点的径向位移，并希望其中的最大位移最小。对于每一点 (x, z), 记 $t = x^2 \in [0, 22500]$ 位移量为：

$$\begin{aligned} d_h(x) &= |300.4 - \sqrt{x^2 + z^2}| = |300.4 - \sqrt{P(t)}| \\ &= |300.4 - \sqrt{\frac{t^2}{16h^2} + (h + 160.414)^2 - \frac{160.414 - h}{2h}t}| \end{aligned} \tag{2}$$

其中 $P(t)$ 是我们为了方便研究引入的中间函数，它是一个二次函数，我们当然可以求得它的极值点以及极值

$$\begin{aligned} t^* &= 4h(160.414 - h) \\ P^* &= 4 \times 160.414h \end{aligned} \tag{3}$$

函数 d_h 关于 P 是凸的，于是为求 d_{min}，我们只需要考虑 P 在取值范围内所有可能的最大值或最小值。由因为 P 是二次函数，性质良好，只需要考虑其在区间端点和极值点处的取值 $P(0)$、$P(22500)$、P^* 即可。对于每个 h，$d_h(t)$ 的最大值一定产生在

$d_h(0)$,$d_h(22500)$,$d_h(t^*)$ 之间，我们希望找到一个 h 使得该最大值最小。即

$$\begin{aligned} h^* &= \arg\min_{h\in I} Max\left\{d_h(0), d_h(22500), d_h(t^*)\right\} \\ &= \arg\min_{h\in I} Max\begin{cases} |h-139.986| \\ |300.4-2\sqrt{160.414h}| \\ |300.4-\sqrt{\frac{22500^2}{16h^2}+(h+160.414)^2-\frac{11250(160.414-h)}{h}}| \end{cases} \end{aligned} \tag{4}$$

其中区间 $I=[139.386, 140.586]$。

由于该区间不是很长，我们可以取 $|I|/100000$ 为步长，编程使用枚举法搜寻最佳的 h^*，精度足够且不会陷入局部最优。得到 $h^*=140.322$，于是有抛物面方程：

$$\Gamma_1 : x^2+y^2-561.288z-168799.508=0 \tag{5}$$

但对于这个结果其实是不够满意的，因为在实际的使用中，还希望促动器移动距离总和尽可能小，以延长系统的寿命。尽管如此，此结果对下面的讨论也有很重要的指导作用，其说明了要使系统运行平稳，抛物面的顶点一定是向低于球面的方向变动的，即 $h>F=139.986$。

在理想情况下，由于我们把节点看作是连续的，那么移动距离之和自然可以从积分的角度看作阴影部分的体积。在系统想理想抛物面变化的过程中，我们基本可以认定质心是不变的，那么该体积自然可以看作始终为截面积的某个常数倍，于是我们又可以将问题划归到平面上解决，记抛物线表达式为 f，下半圆弧表达式为 g，圆点与抛物线左端点的连线方程为 l，则：

$$\begin{aligned} f_h(x) &= \frac{x^2}{4h}-(h+160.414) \\ g(x) &= -\sqrt{300.4^2-x^2} \\ p_h(x) &= \frac{x(h+160.414-\frac{x^2}{4h})}{150} \end{aligned} \tag{6}$$

于是有：

$$l\cap f = Z^- = (\frac{45060}{\sqrt{22500+(-160.414+5625/h-h)^2}}, y_H) \tag{7}$$

这里的 z_H 是纵坐标，我们并不关心。右端对称地有 Z^+，横坐标为 $-X_H$ 那么我们即要处理下图的阴影面积。

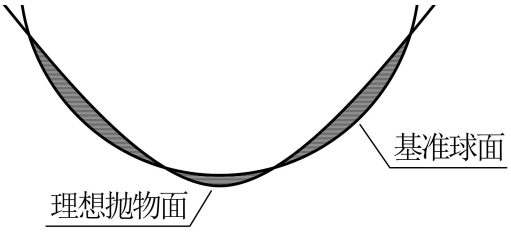

图 **5**　抛物面与球面的相交情况

那么首先我们需要知道抛物线与圆的相交情况。观察认为，抛物线与圆总有四个交点，为验证这一点，我们求得 f 与 g 的交点表达式如下：

$$H_i = \pm 0.5\sqrt{-16h(h-C_1) \pm 2\sqrt{64h^2(h-C_1)^2 - 64h^2(h^2+2C_1h-C_2)}} \tag{8}$$

其中 $i \sim (1,2,3,4)$，常数 $C_1 = 160.414$，$C_2 = 64507.509$

如此标号，是因为我们断言这四个交点在 h 的搜索范围内始终存在。事实上，我们通过实验验证这一点，当 $h \in [139.986, 140.586]$，零点的横坐标变化如下图所示（由于对称性，我们只需要考虑其中两个点）。

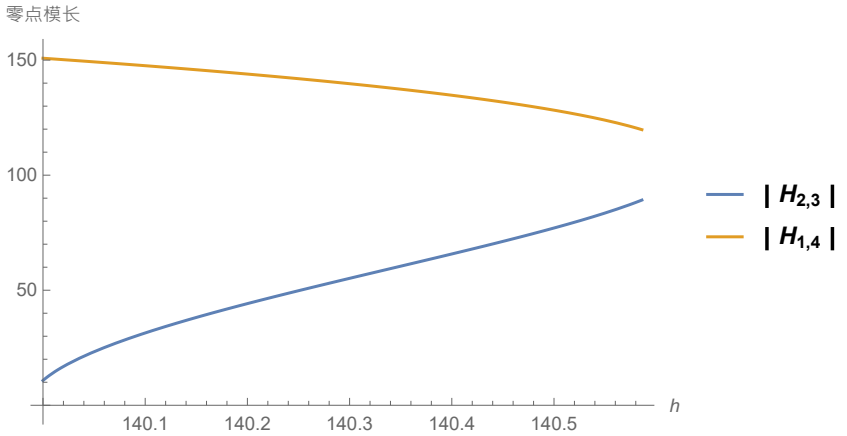

图 **6**　零点分析

可以看出，零点总在 $|x| \leq 150$ 的范围内存在。

于是我们就可以分段求得每一块阴影部分的面积。事实上，任何 h，$f_h - g$ 和 $f_h - p_h$ 的不定积分都是很初等的。忽略常数项，我们求得它们积分的原函数为：

$$\begin{aligned} I_1(x,h) &= \int (f_h - g)dx = C_1x + hx + C_2x^2 + C_3hx^2 + C_4\frac{x^3}{h} + C_5\frac{x^4}{h} \\ I_2 &= \int (p_h - f_h)dx = -C_1x - hx + \frac{x^3}{12h} + \frac{x}{2}\sqrt{C_6 - x^2} + C_7 \arcsin(C_8x) \end{aligned} \tag{9}$$

其中系数 $C_1 = 160.414$，$C_2 = 0.535$，$C_3 = 0.00333$，$C_4 = 0.0833$，$C_5 = 0.000417$，$C_6 = 90240.160$，$C_7 = 45120.080$，$C_8 = 0.00333$。

这时，我们就可以求解是的误差面积最小的 h 值，也即平均最优条件下的 h。设

$H_0 = -150$，$H_5 = 150$，

$$h^* = \arg\min_{h\in I}(\sum_{i=0}^{5}|I_1(H_i,h) - I_1(H_{i+1},h)| + \sum|I_2(Z^{\pm},h) - I_2(\pm 150,h)|) \tag{10}$$

其中 $I = [139.986, 140.586]$，在此区间中以 0.001 为步长，编程使用枚举法求得 $h^* = 140.364$。这时的抛物面方程为：

$$\Gamma_2 : x^2 + y^2 - 561.456z - 168873.388 = 0 \tag{11}$$

同时求得最小误差面积为 $S^* = 152.692$。

我们希望系统兼具效率和稳定性，于是考虑放松对最大偏差的约束，搜寻一种综合最优的抛物面。设最大偏差为 a，上面我们已经求得最小的最大偏差为 0.336；当平均最优时，最大偏差为 0.378。

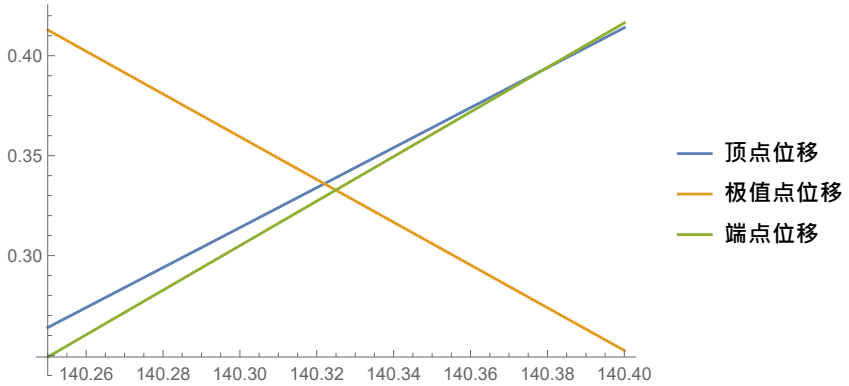

图 7　松弛性分析

我们考虑最开始讨论的最大偏差值函数，不难看出对于每个松弛的 a，h 的搜索范围为 $[140.322, 140.986 + a]$。在此范围中，对于每个 a，我们可以与上面的过程类似求得最佳的 h_a^*，并求出最小误差面积 S_a^*，以 a 为横坐标，S_a^* 为纵坐标作图如下。不难看出，在 h_a^* 达到 h^* 之前，S_a^* 与 a 近似是负线性相关的，那么我们只要考虑在 $a \in [0.336, 0.378]$ 中时，综合最优的一个点。我们将问题看作对 a 和 S_a^* 的一个多目标规划，使用理想点法求解，即求得：

$$\begin{aligned} h^{**} = \arg\min_{h\in I}((\frac{a-0.336}{0.336})^2 + (\frac{\sum_{i=0}^{5}|I_1(H_i,h) - I_1(H_{i+1},h)|}{152.692} \\ \frac{\sum|I_2(Z^{\pm},h) - I_2(\pm 150,h)| - 152.692}{152.692})^2) \end{aligned} \tag{12}$$

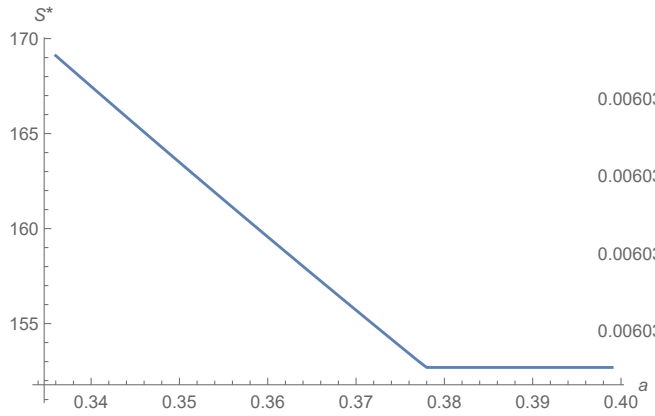

图 **8**　**a** 与 **S** 的负线性关系

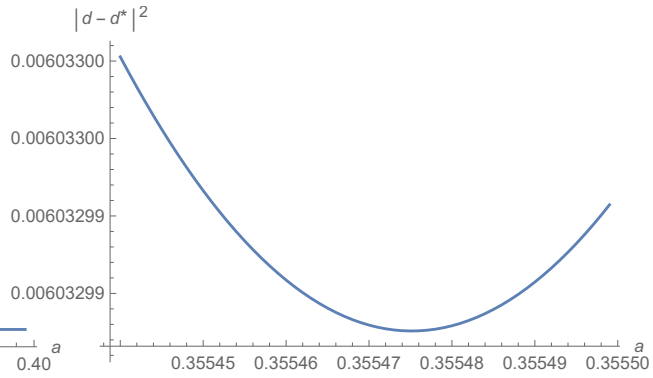

图 **9**　理想点距离搜寻

我们不断提高精度作图，最终观察到 $h^{**} = 140.341$。此时的理想抛物线为：

$$\Gamma_3 : x^2 + y^2 - 561.364z - 168832.805 = 0 \tag{13}$$

综上，我们分别找到了三种不同的理想抛物线，分别代表最小极大偏差、最小移动总程和综合最优：

表 **1**　理想抛物面最优方程

序号	最优目标	h	理想抛物面方程
Γ_1	极端值最优	140.322	$x^2 + y^2 - 561.288z - 168799.508 = 0$
Γ_2	总位移最优	140.364	$x^2 + y^2 - 561.456z - 168873.388 = 0$
Γ_3	综合最优	140.341	$x^2 + y^2 - 561.364z - 168832.805 = 0$

在之后的讨论中，我们选取 $\Gamma_3 : x^2 + y^2 - 561.364z - 168832.805 = 0$ 作为理想抛物面进行研究。

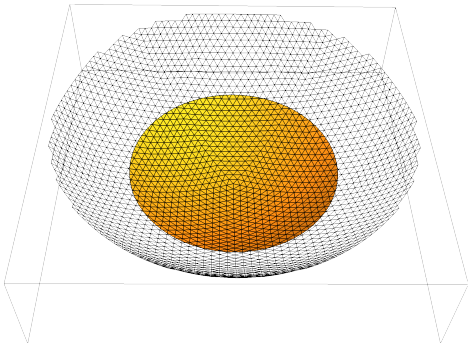

图 **10**　理想抛物面与基准球面的三维视图

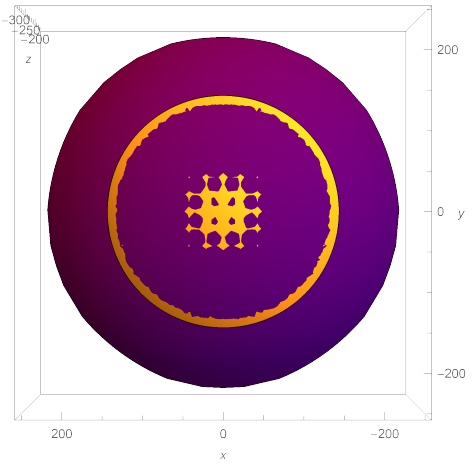

图 **11**　理想抛物面与基准球面的底视图

4.2 问题二的模型建立与求解

4.2.1 理想抛物面方程的确定

当方位角 $\alpha = 36.795°$ 和仰角（高度）$\beta = 78.169°$ 时，由于抛物面的形状不变，只需要将问题一中求出的抛物面方程分别沿坐标轴旋转至相应位置。

空间中绕 y 轴的旋转矩阵为：

$$R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{14}$$

这里的角度与右手螺旋方向相反，y 轴正半轴向上，在 xz 平面逆时针为正方向。

空间中绕 z 轴的旋转矩阵为：

$$R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

这里的角度与右手螺旋方向相反，z 轴正半轴向上，在 xy 平面逆时针为正方向。

由方位角和仰角的示意图可知，变换后的理想抛物面只需沿 z 轴顺时针旋转 α 角度，y 轴顺时针旋转 $90° - \beta$ 再沿 z 轴复原即可得到问题一理想抛物面顶点在 z 轴负半

轴的抛物面方程。因此，该题的旋转矩阵为：

$$
\begin{aligned}
&R_z^T(-\alpha)\,R_y(-(90-\beta))R_z(-\alpha)\\
&=\begin{bmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(90-\beta) & 0 & -\sin(90-\beta)\\ 0 & 1 & 0\\ \sin(90-\beta) & 0 & \cos(90-\beta)\end{bmatrix}\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ -\sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix}\\
&=\begin{bmatrix}\cos^2\alpha\sin\beta+\sin^2\alpha & \sin\alpha\cos\alpha(\sin\beta-1) & -\cos\alpha\cos\beta\\ \sin\alpha\cos\alpha(\sin\beta-1) & \sin^2\alpha\sin\beta+\cos^2\alpha & -\sin\alpha\cos\beta\\ \cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta\end{bmatrix}
\end{aligned}
\tag{16}
$$

可以得到问题一的理想抛物面原坐标 (x_1, y_1, z_1) 和问题二旋转后的理想抛物面坐标 (x, y, z) 的关系：

$$
\begin{bmatrix}x_1\\ y_1\\ z_1\end{bmatrix}=\begin{bmatrix}\cos^2\alpha\sin\beta+\sin^2\alpha & \sin\alpha\cos\alpha(\sin\beta-1) & -\cos\alpha\cos\beta\\ \sin\alpha\cos\alpha(\sin\beta-1) & \sin^2\alpha\sin\beta+\cos^2\alpha & -\sin\alpha\cos\beta\\ \cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta\end{bmatrix}\begin{bmatrix}x\\ y\\ z\end{bmatrix}
\tag{17}
$$

代入方程 (15)：$x_1^2+y_1^2-561.364z_1-168832.805=0$ 可以得到问题二的理想抛物面方程：

$$
\begin{aligned}
0.921x^2+0.957y^2+0.040z^2-92.882x-67.967y-549.439z\\
-0.116xy-0.301xz-0.222yz=168832.805
\end{aligned}
\tag{18}
$$

理想抛物面的顶点坐标为 $[-49.37824479, -36.93292644, -294.36555048]$。

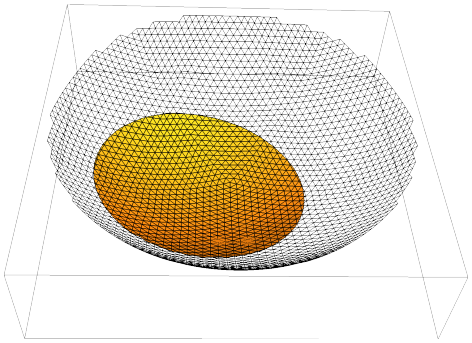

图 **12**　理想抛物面与基准球面的三维视图

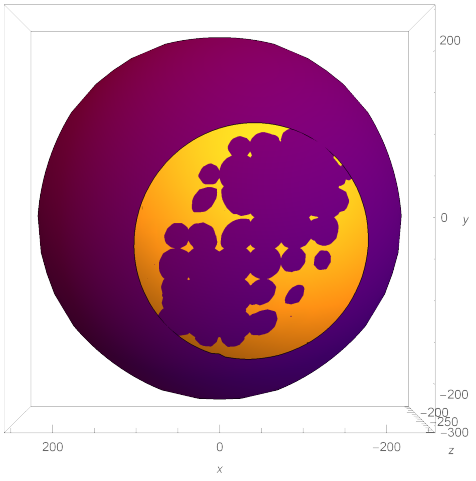

图 **13**　理想抛物面与基准球面的底视图

4.3 反射面板调节模型的建立

1. **确定可动节点的范围**

平面上的一段抛物线表达为 $y = kx^2 = \frac{x^2}{4h}$，$h$ 为焦距，则其在 $x \in [-d, d]$ 区间内的曲线长度 L 的方程为：

$$
\begin{aligned}
&L = 2\int_0^d \sqrt{y'^2 + 1}dx = 2\int_0^d \sqrt{4k^2x^2 + 1}dx = \frac{2}{k}\int_0^d \sqrt{4k^2x^2 + 1}d(kx) \\
&\text{设：} v = kx, v = \frac{1}{2}\tan u \Rightarrow dx = \frac{1}{2}\sec^2 u\text{，则，} \\
&L = \frac{2}{k}\int_0^{\arctan 2d} \frac{1}{\cos^3 u}du = 2(h\ln(\frac{d}{2h} + \sqrt{1 + \frac{d^2}{4h^2}}) + \frac{d}{2}\sqrt{1 + \frac{d^2}{4h^2}})
\end{aligned}
\tag{19}
$$

再考虑将这段弧对称地“贴到”圆上，则其对应的圆心角 $\theta = L/R$，因此得到允许的最大水平位移为

$$
d^* = Rsin(\frac{1}{R}h\ln(\frac{d}{2h} + \sqrt{1 + \frac{d^2}{4h^2}}) + \frac{d}{2}\sqrt{1 + \frac{d^2}{4h^2}}) \tag{20}
$$

代入本题数据得 $d^* = 149.83$，于是我们只考虑理想抛物线周围口径为 299.66 米的球面内的节点发生移动。

2. **计算各节点到抛物线的最短距离**

已知旋转后的抛物线顶点 H 的坐标为 $[-49.378, -36.933, -294.388]$，则对平面内任一点 X，可将其转化到一个以 H 为原点，过抛物面中轴的平面内。其到抛物面的最短距离，恰为其在该平面内到抛物线 $y = \frac{x^2}{4h}$ 的最短距离。由于对称性，转化后的横坐标正负不影响距离的值，因此我们只要考虑 x 在该平面的横纵截距即可，即 X 的平面坐标 X' 为：

$$
X' = (|\overrightarrow{HX} \times \vec{A}|, -\overrightarrow{HX} \cdot \vec{A}) \tag{21}
$$

易得平面内一点到抛物线的最短距离，考虑抛物线上与该点连线垂直于切向的点即可。求得可能有两个满足条件的点，只需考虑最近的一个点，即：

$$
\begin{aligned}
&L^2(X) = (\frac{x_i^2}{4h} + \overrightarrow{HX} \cdot \vec{A})^2 + (x_i^2 - |\overrightarrow{HX} \times \vec{A}|) \\
&\text{其中，} x_i = \underset{x \in R}{\arg} 4x(\frac{x^2}{4h} + \overrightarrow{HX} \cdot \vec{A}) + 2(x - |\overrightarrow{HX} \times \vec{A}|)
\end{aligned}
\tag{22}
$$

其中 x_i 可以用卡尔达诺公式求解。

4.4 促动器伸缩量的调节

首先，对附件 1 和附件 2 中主索节点坐标、促动器下端点坐标和促动器上端点坐标进行综合分析可知：

- 促动器的长度均为 1.98 米；下拉索长度在 1.23 ~ 49.45 米之间；

- 促动器伸缩方向及下拉索伸缩方向之间夹角为 0，即可以认为促动器两端点与主索节点共线。

于是，我们假设促动器仅能径向伸缩，不考虑其横向位移。同时，在本问题中需要考虑两个约束条件：

1. 促动器伸缩范围（即主索节点径向变化距离）在 $-0.6 \sim +0.6$ 米；
2. 调节后相邻主索节点之间的距离变化不超过 0.07%。

我们定义调节后的主索节点 i 到理想抛物面的径向距离为 l_i，主索节点共有 n 个，设计在上述约束条件下，为使主索节点尽可靠近理想抛物面，优化目标为：

$$arg\min_{l} E(l) = \sum_{i=1}^{n} l_i^2 \tag{23}$$

4.4.1 理想情况的探究

首先不考虑促动器伸缩量及约束条件，探索将全部可移动主索节点径向收缩到理想抛物面上时，约束条件的满足情况。考虑到球面和抛物面是旋转体存在对称性。将主索节点 P 乘上旋转矩阵变为 P'，将 P' 与圆心 O 的连线方程和旋转前的理想抛物面方程联立，解出交点坐标 Q'，再将 Q 乘上旋转矩阵的逆，得到点 P 与理想抛物面的交点 Q。

假设 P' 坐标为 (x_1, y_1, z_1)，联立方程为：

$$\begin{cases} x^2 + y^2 - 4hz = 4h(R + h - F) \\ \frac{x_1 - x}{x_1} = \frac{y_1 - y}{y_1} = \frac{z_1 - z}{z_1} \end{cases} \tag{24}$$

由此解出所有主索节点对应在理想抛物面上的坐标。此时我们计算所有促动器伸缩范围和相邻主索节点之间变化情况，发现促动器最大改变长度为 0.438 米符合范围要求，但是相邻节点最大变化幅度达到了 0.159%，超过了 0.07%。

接着，我们令所有促动器的径向收缩长度与其沿径向到达理想抛物面的距离成正比，以此初步估计促动器的调节方式。设其比例为 $\lambda(0 \leqslant \lambda \leqslant 1)$，从 0 开始逐步增大 λ 的值，得到当 $\lambda = 0.509$ 时，刚好到达相邻节点距离变化的约束条件临界值 0.07%。此时目标函数值为 2.946。我们也将此时的各个主索节点伸缩长度作为初步的调节方案作为后续参考。

4.4.2 模拟退火算法及其实现

为逃出前面得到的调解方案的局部最优限制，我们采用模拟退火算法探索全局最优解。模拟退火算法是一种基于蒙特卡洛迭代求解策略的随机寻优算法，出发点是基于固体物质的退火过程与一般组合优化问题之间的相似性。它是局部搜索算法的扩展，以一定的概率选择领域中目标值较大的状态，从而达到求解全局优化问题的目的，判断新解是否接受的依据为 Metropolis 准则：

1. 若 $\Delta E < 0$，则接受新解；
2. 否则接受新解的概率由下式确定：

$$p = e^{-\frac{\Delta E}{T}} \tag{25}$$

本题中模拟退火算法的流程图如下图表示：

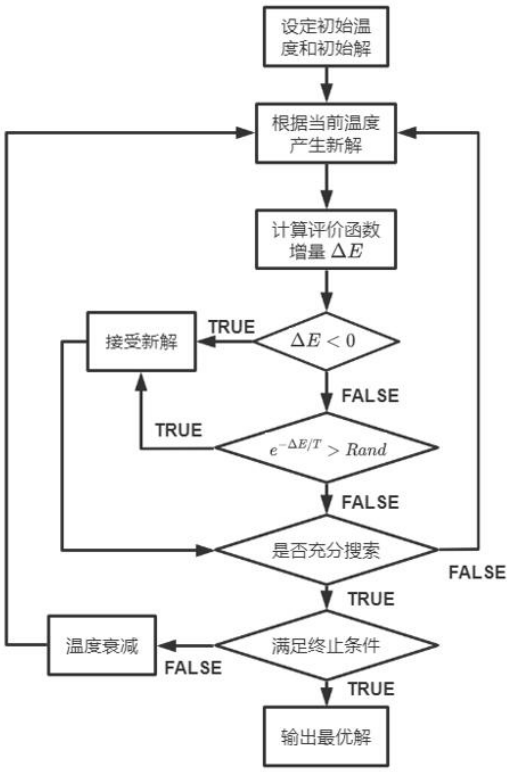

图 **14**　模拟退火流程图

- **参数和初始值设定：**初始温度 $T_i = 0.1$，终止温度 $T_f = 0.001$，每个温度下 Markov 长度为 1000，温度以 $T(j+1) = \alpha T_j$ 下降，$\alpha = 0.85$。初始值为之前得到的初步调节方案。
- **新解产生策略：**假设主索节点 P_j 径向到达理想抛物面交点 Q_j 的距离为 P_jQ_j，在温度 T 下的调节方案为 l_j，那么其下一步的变化范围是 $[l_j - \beta \cdot P_jQ_j \cdot T, l_j + (1-\beta) \cdot P_jQ_j \cdot T]$，其中 $\beta = 0.25$。即节点 P_j 有 β 的概率远离抛物面，有 $(1-\beta)$ 的概率远离抛物面。
- 随着迭代次数的增加，目标函数的值逐渐下降，最终下降到 2.5132，如下图所示：

此时我们得到了可接受的接近全局最优解的调节方案，完整解决方案保存在 result.xlsx 文件中，部分主索节点的变化如下表所示：

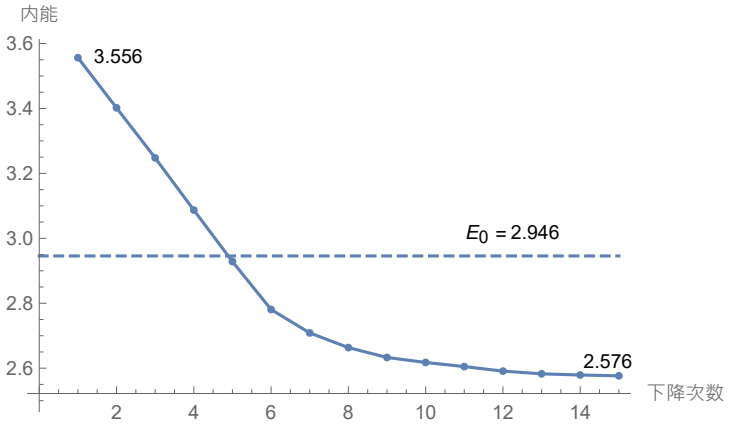

图 **15**　模拟退火迭代下降

节点编号	X 坐标（米）	Y 坐标（米）	Z 坐标（米）	伸缩量（米）
A0	0.000	0.000	-300.390	0.010
B1	6.106	8.405	-300.152	0.068
C1	9.881	-3.210	-300.171	0.050
D1	0.000	-10.392	-300.240	-0.020
E1	-9.884	-3.211	-300.256	-0.036
A1	-6.107	8.407	-300.203	0.017
A3	0.000	16.814	-299.859	0.070

4.4.3 模型分析

下图 16 是实际“抛物面”与理想抛物面的对比，可以看出总体位移呈中间下降，两端上升的形式。

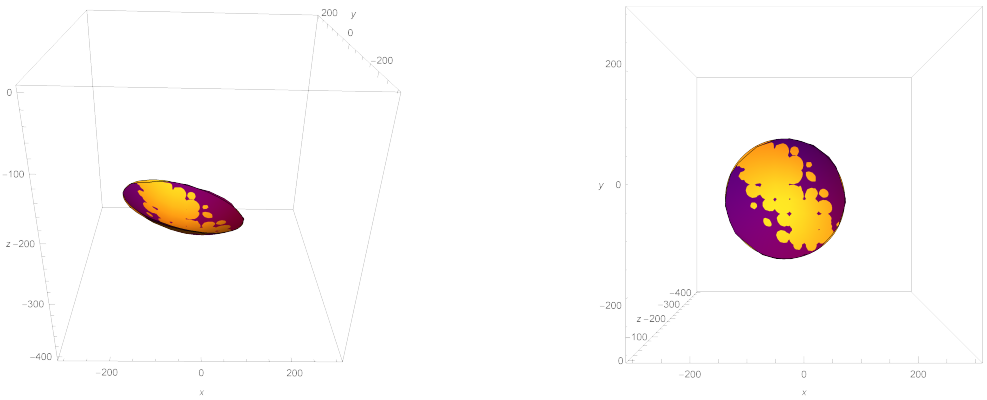

图 **16**　优化结果与理想抛物面的对比

对各个位置促动器的行程作图 17，可以看出促动器的伸长量明显呈现高-低-高-低-高的层次结构，即在两端较高且中间有一峰值，这与我们第一问的分析相互印证。这也说明，当追踪某一天梯时，工作面的压力总是两端和中间较高，这对我们进行设备的监控和维护有着重要的指导作用。

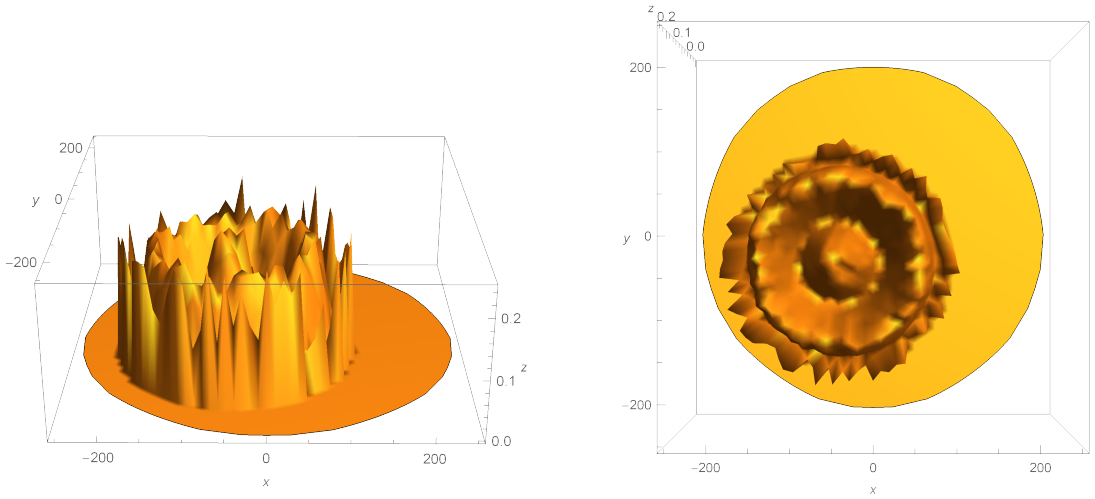

图 **17**　形变分析图

4.5 问题三的模型建立与求解

4.5.1 信号在三角反射面的反射过程

记某块给定的反射板为 $\triangle abc$，a、b、c 三点顺时针标定，接收信号是反射板在垂直于入射信号向量平面上的投影面积。假设入射信号的单位向量为 $\vec{A}$，反射面的单位外法向量为 $\vec{N}$，有 $\vec{N}=\sigma\left(\overrightarrow{ac}\times\overrightarrow{ab}\right)$，$\sigma$ 是向量单位化函数，定义为 $\sigma(\vec{\alpha}):=\frac{\vec{\alpha}}{|\vec{\alpha}|}$，易知反射板与投影平面的二面角 Θ 等于 $\vec{N}$ 和 $\vec{A}$ 的夹角，$\cos\Theta=-\vec{A}\cdot\vec{N}$。得到方程：

$$S_0=S_{abc}\cos\Theta=-\frac{1}{2}|\overrightarrow{ac}\times\overrightarrow{ab}|\vec{A}\cdot\sigma(\overrightarrow{ac}\times\overrightarrow{ab}) \tag{26}$$

计算每块板实际接收信号的比例时，首先要考虑空间平面镜的反射过程。假设反射信号的单位向量为 $\vec{B}$，从图中可知 $\vec{A}-\vec{B}\parallel\vec{N}$，得到方程：

$$\vec{B}-\vec{A}=\Gamma\vec{N} \tag{27}$$

其中常数 $\Gamma=2|\vec{A}|\cos\Theta=-2\vec{A}\cdot\vec{N}$，带入 (11) 中求得反射信号的单位向量

$$\vec{B}=-2\vec{A}\cdot\vec{N}\cdot\vec{N}+\vec{A} \tag{28}$$

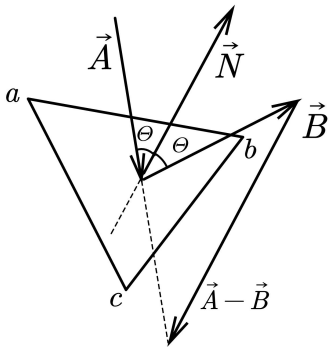

图 **18**　$\triangle abc$ 的信号反射

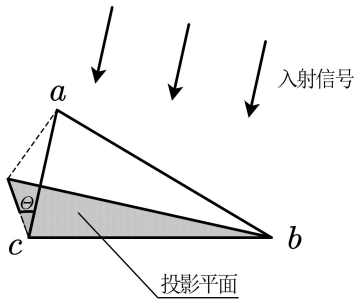

图 **19**　三角反射面在接收信号下的投影平面

4.5.2 对反射面上任意点的信号反射分析

考虑该平面上某点 x 获得信号并反射后，反射信号与接收平面的交点 x'。假设接收圆盘圆心为 O，只需考虑 a、b、c 三点的反射位置，并求出它们构成的三角形与 $\odot O$ 的重合面积与该三角形面积的比，即可得到接收平面在实际过程中接收信号的比例。

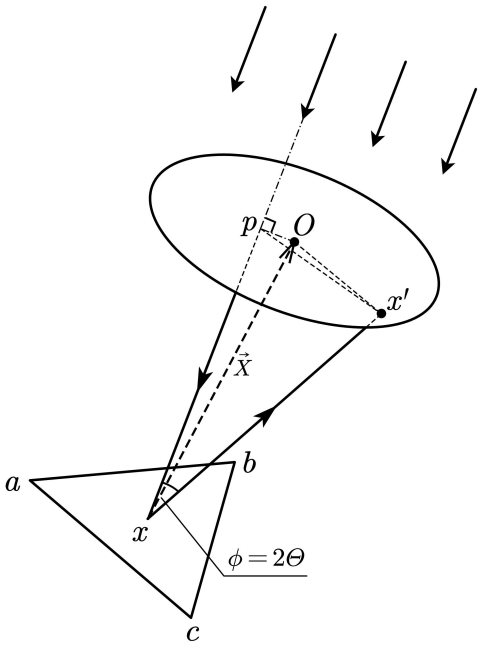

图 **20**　反射面上任意点 **x** 的信号反射

考虑点 $x\sim(a,b,c)$，并令 $\vec{X}=\overrightarrow{Ox}$。点 p 是点 x 在馈源舱接收平面上的投影，由于模型假设入射信号与接收平面垂直，$\overrightarrow{px}$ 即为入射信号，显然 $\overrightarrow{xp}=\kappa\vec{A}$，常数 $\kappa=-\vec{X}\cdot\vec{A}$。点 x' 是点 x 在接收圆盘上的信号反射点，先求出点 p 的位置，以便求出接收平面上点

x' 相对于 O 的方向。由于 $\overrightarrow{xp} \perp \odot O$，得到方程：

$$\begin{aligned} \overrightarrow{xp} &= -\vec{X} \cdot \vec{A} \cdot \vec{A} \\ \overrightarrow{Op} &= \vec{X} - \vec{X} \cdot \vec{A} \cdot \vec{A} \end{aligned} \tag{29}$$

再考虑反射点 x',$\triangle pxx'$ 显然与接收平面垂直，得到 $\overrightarrow{px'}$ 的单位向量 $\sigma(\overrightarrow{xp} \times \vec{B} \times \overrightarrow{xp})$。由几何关系可知，$|\overrightarrow{px'}| = |\overrightarrow{xp}| \tan\phi$，$\phi$ 为 $\overrightarrow{xp}$ 和 $\vec{B}$ 的夹角，得到方程：

$$|\overrightarrow{px'}| = |\overrightarrow{xp}| \tan\phi = |\overrightarrow{xp}| \frac{\sin\phi}{\cos\phi} = \frac{|\overrightarrow{xp}| \cdot |\vec{B} \times \overrightarrow{xp}|}{\vec{B} \cdot \overrightarrow{xp}} \tag{30}$$

因此，

$$\overrightarrow{px'} = \frac{|\overrightarrow{xp}| \cdot |\vec{B} \times \overrightarrow{xp}|}{\vec{B} \cdot \overrightarrow{xp}} \sigma(\overrightarrow{xp} \times \vec{B} \times \overrightarrow{xp}) \tag{31}$$

又因为 $\overrightarrow{Ox'} = \overrightarrow{Op} + \overrightarrow{px'}$, 计算得反射点 x' 的位置为：

$$\begin{aligned} x' &= \vec{O} + \vec{X} - \vec{X} \cdot \vec{A} \cdot \vec{A} + \frac{|\overrightarrow{xp}| \cdot |\vec{B} \times \overrightarrow{xp}|}{\vec{B} \cdot \overrightarrow{xp}} \sigma(\overrightarrow{xp} \times \vec{B} \times \overrightarrow{xp}) \\ &= \vec{x} - \vec{X} \cdot \vec{A} \cdot \vec{A} + \frac{|\overrightarrow{xp}| \cdot |\left(\vec{A} - 2\vec{A} \cdot \vec{N} \cdot \vec{N}\right) \times \overrightarrow{xp}|}{\left(\vec{A} - 2\vec{A} \cdot \vec{N} \cdot \vec{N}\right) \cdot \overrightarrow{xp}} \sigma(\overrightarrow{xp} \times \left(\vec{A} - 2\vec{A} \cdot \vec{N} \cdot \vec{N}\right) \times \overrightarrow{xp}) \end{aligned} \tag{32}$$

4.5.3 计算反射后三角形与接收圆盘的重合面积

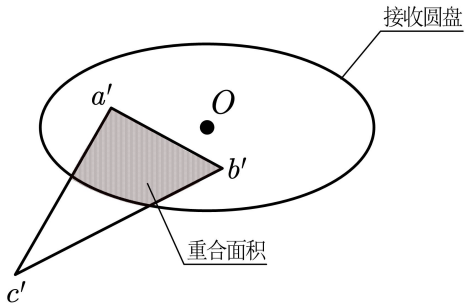

图 **21**　馈源舱接收平面与反射后三角形的重合面积

$\triangle a'b'c'$ 的坐标转换：

在实际计算时，只考虑对主动反射面反射到接收平面的信号进行求和。主动面以外的反射板也可能反射信号至接收平面，由于这部分信号接收比较差，几乎可以忽略不计。因此，计算抛物面和球面的接收比时，只考虑问题二中得到口径为 299.66m 的球面上包含的反射板。为了简化反射三角形和接收圆盘重合面积的计算，将 a'、b'、c' 的位置用 $\vec{O}$ 和 $\vec{A} = [A_1, A_2, A_3]$ 的坐标表示。

首先对反射板上任意点 x 反射后的坐标 x' 的坐标进行转换。取一组与 $\vec{A}=[A_1,A_2,A_3]$ 两两垂直的单位向量 $\vec{\gamma}$、$\vec{\eta}$，不妨设：

$$\begin{aligned}\vec{\gamma}&=\sigma([0,A_3,-A_2])\\ \vec{\eta}&=\sigma([-2\frac{A_2^2+A_3^2}{A_1},A_2,A_3])\end{aligned}\tag{33}$$

则 x' 转换后的坐标为 $x^*=[\overrightarrow{Ox'}\cdot\vec{\gamma},\overrightarrow{Ox'}\cdot\vec{\eta}]$，即

$$\overrightarrow{x^*}=(\overrightarrow{x'}-\vec{O})\begin{bmatrix}\sigma([0,A_3,-A_2])\\ \sigma([-2\frac{A_2^2+A_3^2}{A_1},A_2,A_3])\end{bmatrix}^T\tag{34}$$

重合面积的微小分割：

进行如上坐标变换后，我们就可以在平面中考虑解决该问题。对于反射三角形与 $\odot O$ 的重合面积，取微小步长 dt, 在 $\odot O$ 取足够多的点，判断其是否在 $\triangle a'b'c'$ 内，通过调整 dt 来控制精度。

对于一个圆心在原点，半径为 r 的圆 O，做出它的一个外切正方形，四条边分别垂直于坐标轴，如图。

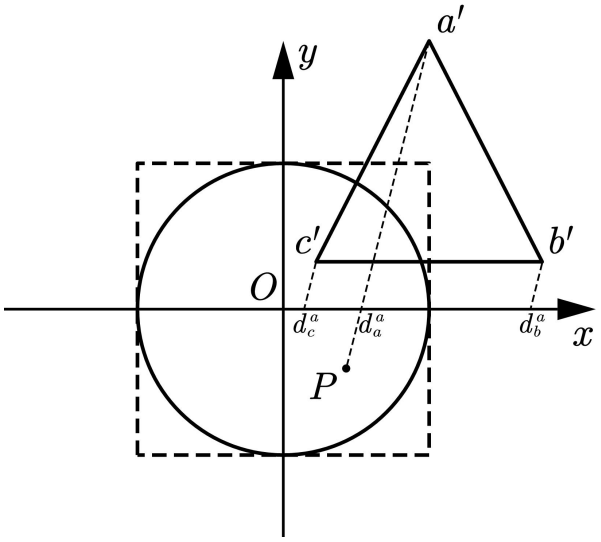

图 **22**　外切正方形示意图

将该正方形以步长 dt 切分为若干个点，对于某个点 P, 判断其 (1) 是否在圆内 (2) 是否在三角形内。(1) 是容易的，只需要比较 $P_x^2+P_y^2$ 与 r^2 即可。对于 (2)，如图，我们三角形 ABX 的一个顶点 A, 则可得到过 A、P 的直线斜率 k 为 $\frac{A_y-P_y}{A_x-P_x}$，在 x 轴的横坐标为 d_a^a；若 k=0，则计算其在 y 轴的纵坐标，也记为 d_a^a；过 B、C 两点分别作等斜率的直线，也同理得到在 x 轴或 y 轴的坐标 d_b^a、d_c^a，则该点在三角形内部等价于：

$$(d_a^a-d_b^a)(d_a^a-d_c^a)<0且(d_b^b-d_c^b)(d_b^b-d_a^b)<0且(d_c^c-d_b^c)(d_c^c-d_a^c)<0 \tag{35}$$

求得同时满足判断 (1)(2) 的点的数量 N，则重合部分面积

$$S=\frac{Ndt^2}{r^2} \tag{36}$$

4.5.4 计算信号接受率

最终，设圆盘的总分割点数为 N，落在某个给定的反射三角形内的分割点数为 N_i，我们获取第 k 个反射板的 S_0^k、S'^k、S_i^k、N_i^k、，计算得到信号接收率模型为:

$$\epsilon=\frac{\Sigma_k S_0^k S_i^k/S'^k}{\Sigma_k S_0^k} \tag{37}$$

其中：

$$\begin{cases} S_0=S_{abc}\cos\Theta=\dfrac{1}{2}|\overrightarrow{ac}\times\overrightarrow{ab}|\vec{A}\cdot\sigma(\overrightarrow{ac}\times\overrightarrow{ab}) \\ S'=\dfrac{1}{2}|\overrightarrow{a'c'}\times\overrightarrow{a'b'}| \\ x'=\vec{x}-\vec{X}\cdot\vec{A}\cdot\vec{A}+\dfrac{|\overrightarrow{xp}|\cdot|\left(\vec{A}-2\vec{A}\cdot\vec{N}\cdot\vec{N}\right)\times\overrightarrow{xp}|}{\left(\vec{A}-2\vec{A}\cdot\vec{N}\cdot\vec{N}\right)\cdot\overrightarrow{xp}}\sigma(\overrightarrow{xp}\times\left(\vec{A}-2\vec{A}\cdot\vec{N}\cdot\vec{N}\right)\times\overrightarrow{xp}) \\ \quad\begin{cases}\overrightarrow{xp}=-\vec{X}\cdot\vec{A}\cdot\vec{A} \\ x\sim(a,b,c) \\ \vec{N}=\sigma(\overrightarrow{ac}\times\overrightarrow{ab})\end{cases} \\ S_i^k=\dfrac{N_i^k dt^2}{r^2}\end{cases} \tag{38}$$

由程序代入数据后求得：

初始球面吸收率：0.828%

实际抛物面吸收率：1.059%

理想抛物面吸收率：1.148%

对比发现抛物面工作面的接收比明显高于原始球面，提高了 27.9%。但受制于现实原因，远远达不到抛物面 100% 聚光的效果。如果所有节点都能移动到理想抛物面上，则还能再提高 7%。

五、 模型的评价与推广

本文主要的研究手段是借助向量去简化一些计算，其实对于光学模型的分析本身是容易的。各个问题的解决手段特征主要如下：

1. **问题一：**

   从三个角度，用不同的方式定义并研究“最优”，既考虑了使得极端情况最小的模型，也考虑了使得促动器总行程最短的模型。假设最优情况是两者兼备，我们找到了与该理想点距离最短的情况，选择它作为理想抛物面。应该说这种选择综合性能是比较好的，既保证了系统的效率，也不影响系统的稳定性。不足之处在于很难在连续的情况下考虑节点之间的形变量，无法对抛物线的“张角”进行量化描述。事实上，这也是使我们第二问的模型无法达到理论最优的主要约束。

2. **问题二：**

   在本问题中，我们首先观察到对称性，利用旋转矩阵直接得到新的理想抛物面，接着使用模拟退火算法避免陷入局部最优解，假定所有节点都在抛物面上为最优，不断逼近这种理想情况，选取足量的初值最终找到了一个满意解。但这里有一个问题，在现实中对抛物面的最佳逼近是不是所有节点都在抛物面上呢？答案是否定的。但当线段与抛物线相截时，明显可以比两端点都在抛物线上更加理想，抛物面也是同理。一个可能的优化想法是算出所有节点的质心，也加入残差的计算中，再把质心看作新的节点生成一组新的质心参考点，如此迭代便能不断优化下去。

3. **问题三：**

   在本问题中，我们使用向量描述了整个系统，对编程计算起到了很好的简化作用。但应当注意的是，在实际使用中，工作面边缘的反射板并不总是能完整地接收到信号，它的投影边界可能是椭圆的，这研究起来比较复杂。同时，对于工作面之外的点我们选择忽略其反射，但根据实际的光学分析，完全可能有信号来自于非工作面节点。

4. **推广探究：**

   事实上在实际使用 FAST 时，工作面口径可以大于 300 米，这就意味着可以接收更大范围的信号。对于大于 300 米的口径，研究思想也与本文基本类似，这意味着我们可以将系统的工作面设计更加优化。

参考文献

[1] 钱宏亮. FAST 主动反射面支承结构理论与试验研究 [D]. 哈尔滨工业大学,2007.

[2] 王志远. 基于迭代学习理论的 FAST 整网控制策略的研究 [D]. 东北大学,2015.

[3] 孙纯, 朱丽春, 于东俊.FAST 主反射面节点运动控制算法 [J]. 科学技术与工程,2012,12(03):489-493.

[4] https://blog.csdn.net/ls9512/article/details/49764311

[5] https://kns.cnki.net/kcms/detail/detail.aspx?

附录 A　数据准备

```
import pandas as pd
import numpy as np

data1 = pd.read_csv('data10.csv')
data2 = pd.read_csv('data20.csv')

# alpha = 0; beta = 90 / 180 * np.pi
alpha = 36.795 / 180 * np.pi; beta = 78.169 / 180 * np.pi

print((data1.n0 != data1.n1).sum())
data1.drop('n1', axis = 1, inplace = True)

data1['dx1'] = data1.x0 - data1.xu
data1['dy1'] = data1.y0 - data1.yu
data1['dz1'] = data1.z0 - data1.zu
data1['dx2'] = data1.xu - data1.xd
data1['dy2'] = data1.yu - data1.yd
data1['dz2'] = data1.zu - data1.zd

data1['l1'] = (data1.dx1 ** 2 + data1.dy1 ** 2 + data1.dz1 ** 2) ** (1/2)
data1['l2'] = (data1.dx2 ** 2 + data1.dy2 ** 2 + data1.dz2 ** 2) ** (1/2)
data1['theta'] = np.arccos((data1.dx1 * data1.dx2 + data1.dy1 * data1.dy2 + data1.dz1 *
    data1.dz2) / (data1.l1 * data1.l2))

location = data1.iloc[ : , : 4]

data2 = (
   data2
   .merge(location, left_on = 'n1', right_on = 'n0', how = 'left')
   .drop('n0', axis = 1)
   .rename(columns={'x0' : 'x1', 'y0' : 'y1', 'z0' : 'z1'})
   .merge(location, left_on = 'n2', right_on = 'n0', how = 'left')
   .drop('n0', axis = 1)
   .rename(columns={'x0' : 'x2', 'y0' : 'y2', 'z0' : 'z2'})
   .merge(location, left_on = 'n3', right_on = 'n0', how = 'left')
   .drop('n0', axis = 1)
   .rename(columns={'x0' : 'x3', 'y0' : 'y3', 'z0' : 'z3'})
)

def distance(x1, y1, z1, x2, y2, z2):
   return ((x2 - x1) ** 2 + (y2 - y1) ** 2 + (z2 - z1) ** 2) ** (0.5)

data2['r12'] = distance(data2['x1'], data2['y1'], data2['z1'], data2['x2'], data2['y2'],
    data2['z2'])
```

```
data2['r13'] = distance(data2['x1'], data2['y1'], data2['z1'], data2['x3'], data2['y3'],
    data2['z3'])
data2['r23'] = distance(data2['x2'], data2['y2'], data2['z2'], data2['x3'], data2['y3'],
    data2['z3'])

df1 = data2.iloc[ : , [0, 1]]
df2 = data2.iloc[ : , [1, 2]].rename(columns = {'n2' : 'n1', 'n3' : 'n2'})
df3 = data2.iloc[ : , [0, 2]].rename(columns = {'n3' : 'n2'})
data3 = pd.concat([df1, df2, df3])

data3['test'] = np.where(data3['n1'] < data3['n2'], data3['n1'] + data3['n2'], data3['n2'] +
    data3['n1'])
data3.drop_duplicates('test', inplace = True)
data3.drop('test', axis = 1, inplace = True)

data3 = (
   data3
   .merge(location, left_on = 'n1', right_on = 'n0', how = 'left')
   .drop('n0', axis = 1)
   .rename(columns={'x0' : 'x1', 'y0' : 'y1', 'z0' : 'z1'})
   .merge(location, left_on = 'n2', right_on = 'n0', how = 'left')
   .drop('n0', axis = 1)
   .rename(columns={'x0' : 'x2', 'y0' : 'y2', 'z0' : 'z2'})
)

data3['r'] = distance(data3['x1'], data3['y1'], data3['z1'], data3['x2'], data3['y2'],
    data3['z2'])

from random import random
from scipy.optimize import fsolve

def equations(vars):
   a, b, c, r = vars
   eqs = []
   for i in range(4):
      eq = (data[i][0] - a) ** 2 + (data[i][1] - b) ** 2 + (data[i][2] - c) ** 2 - r ** 2
      eqs.append(eq)
   return eqs

s = [[],[],[],[]]
for i in range(1000):
   index = []
   for j in range(4):
      index.append((int) (random() * len(location)))

   data = location.iloc[index, 1:].values.tolist()
   a, b, c, r = fsolve(equations, (1, 1, 1, 1))
```

```
    s[0].append(a), s[1].append(b), s[2].append(c), s[3].append(abs(r))

[round(sum(x) / len(x), 4) for x in s]

from scipy.optimize import fsolve, root

R = 300.4
F = 0.466 * R
h = 140.341
r = 149.83

ca = np.cos(alpha); sa = np.sin(alpha); cb = np.cos(beta); sb = np.sin(beta)

A = np.array([- np.cos(beta) * np.cos(alpha),
              - np.cos(beta) * np.sin(alpha),
              - np.sin(beta)])

print('理想抛物面顶点坐标: {}'.format((h + R - F) * A))

T1 = np.array([[sa ** 2 + ca ** 2 * sb, - ca * sa + ca * sa * sb, - ca * cb],
               [- ca * sa + ca * sa * sb, ca ** 2 + sa ** 2 * sb, - cb * sa],
               [ca * cb, cb * sa, sb]])

T2 = np.transpose(T1)

center = A * (R ** 2 - r ** 2) ** (1 / 2)

def intersect(x1, y1, z1, x2, y2, z2):

    def equations(vars):
        x, y, z = vars

        eq0 = x ** 2 + y ** 2 - 4 * h * z - 4 * h * (R + h - F)
        eq1 = (x2 - x1) * (y - y1) - (y2 - y1) * (x - x1)
        eq2 = (y2 - y1) * (z - z1) - (z2 - z1) * (y - y1)
        eq3 = (x2 - x1) * (z - z1) - (z2 - z1) * (x - x1)
        if (x1 == x2) and (y1 == y2):
            return [eq0, eq2, eq3]
        elif (y1 == y2) and (z1 == z2):
            return [eq0, eq1, eq3]
        else:
            return [eq0, eq1, eq2]

    x, y, z = fsolve(equations, (x1, y1, z1))

    return x, y, z
```

```
data1[['ix', 'iy', 'iz']] = data1[['x0', 'y0', 'z0']]

count = 0

for index, row in data1.iterrows():

    x0 = row['x0']; y0 = row['y0']; z0 = row['z0']

    if (x0 - center[0]) ** 2 + (y0 - center[1]) ** 2 + (z0 - center[2]) < r ** 2:

        if alpha == 0:
            ix, iy, iz = intersect(x0, y0, z0, 0, 0, 0)
        else:
            new = np.matmul(T1, np.array([x0, y0, z0]))
            nix, niy, niz = intersect(new[0], new[1], new[2], 0, 0, 0)
            ix, iy, iz = np.matmul(T2, np.array([nix, niy, niz]))
            count = count + 1

    else:
        continue

    data1.loc[index, 'ix'] = round(ix, 4); data1.loc[index, 'iy'] = round(iy, 4);
        data1.loc[index, 'iz'] = round(iz, 4);

data1['l'] = distance(data1['x0'], data1['y0'], data1['z0'], data1['ix'], data1['iy'],
    data1['iz'])

data1['dx'] = (data1.ix - data1.x0) / data1.l
data1['dy'] = (data1.iy - data1.y0) / data1.l
data1['dz'] = (data1.iz - data1.z0) / data1.l

location_new = data1[['n0', 'ix', 'iy', 'iz']]

data3 = (
    data3
    .merge(location_new, left_on = 'n1', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns={'ix' : 'nx1', 'iy' : 'ny1', 'iz' : 'nz1'})
    .merge(location_new, left_on = 'n2', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns={'ix' : 'nx2', 'iy' : 'ny2', 'iz' : 'nz2'})
)

data3['nr'] = distance(data3['nx1'], data3['ny1'], data3['nz1'], data3['nx2'], data3['ny2'],
    data3['nz2'])
data3['cr'] = abs(data3['nr'] - data3['r']) / data3['r']
print('促动器最大改变长度: {} 米'.format(round(max(data1.l), 3)))
```

```
print('相邻节点最大变化幅度: {} %'.format(round(max(data3.cr) * 100, 3)))

df = data1.dropna().reset_index(drop = True)
init = np.array(df[['x0', 'y0', 'z0']].values)
direction = np.array(df[['dx', 'dy', 'dz']].values)
index1 = df[['n0']]
index2 = (
    data3[['n1', 'n2', 'r']]
    .merge(index1, left_on = 'n1', right_on = 'n0', how = 'left')
    .merge(index1, left_on = 'n2', right_on = 'n0', how = 'left')
    .dropna(subset = ['n0_x', 'n0_y'], how = 'all')
    .reset_index(drop = True)
    [['n1', 'n2', 'r']]
)
r = np.array(index2['r'])
index2 = index2[['n1', 'n2']]

data4 = (
    data2
    .merge(index1, left_on = 'n1', right_on = 'n0', how = 'left')
    .merge(index1, left_on = 'n2', right_on = 'n0', how = 'left')
    .merge(index1, left_on = 'n3', right_on = 'n0', how = 'left')
#     .dropna(subset = ['n0_x', 'n0_y', 'n0'], how = 'all')
    .dropna()
    .reset_index(drop = True)
    .drop(['n0_x', 'n0_y', 'n0', 'r12', 'r13', 'r23'], axis = 1)
)

loc_new = data1[['n0', 'ix', 'iy', 'iz']]

data4 = (
    data4
    .merge(loc_new, left_on = 'n1', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'ix' : 'ix1', 'iy' : 'iy1', 'iz' : 'iz1'})
    .merge(loc_new, left_on = 'n2', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'ix' : 'ix2', 'iy' : 'iy2', 'iz' : 'iz2'})
    .merge(loc_new, left_on = 'n3', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'ix' : 'ix3', 'iy' : 'iy3', 'iz' : 'iz3'})
)

data1.to_csv("data1.csv", index = False)
data2.to_csv("data2.csv", index = False)
data3.to_csv("data3.csv", index = False)
data4.to_csv("data4.csv", index = False)
```

附录 B　模拟退火

```
def rate(delta):

    m = init + delta.reshape(-1, 1) * direction
    value = pd.concat([index1, pd.DataFrame(m, columns = ['mx', 'my', 'mz'])], axis = 1)

    mloc = (
        index2
        .merge(value, left_on = 'n1', right_on = 'n0', how = 'left')
        .drop('n0', axis = 1)
        .rename(columns={'mx' : 'mx1', 'my' : 'my1', 'mz' : 'mz1'})
        .merge(value, left_on = 'n2', right_on = 'n0', how = 'left')
        .drop('n0', axis = 1)
        .rename(columns={'mx' : 'mx2', 'my' : 'my2', 'mz' : 'mz2'})
    )

    p1 = np.array(mloc[['mx1', 'my1', 'mz1']].values)
    p2 = np.array(mloc[['mx2', 'my2', 'mz2']].values)

    mr = np.linalg.norm(p1 - p2, axis = 1)

    percent = max(abs(mr - r) / r)

    return percent <= 0.0007

def restrict(delta):
    if np.max(delta) > 0.6 or np.min(delta) < -0.6:
        return False
    return rate(delta)

length = np.array(df.l.values)

def E(delta):
    return np.linalg.norm(length - delta)

step = 0.1; ratio = 0
while step >= 0.001:
    while True:
        temp_ratio = ratio + step
        temp_length = temp_ratio * length
        if restrict(temp_length):
            ratio = temp_ratio
```

```
        else:
            break
    step = step / 10

print('lambda = {}'.format(round(ratio, 3)))
print('E = {}'.format(E(ratio * length)))

import numpy as np
from random import random
from math import exp, sqrt

alpha = 0.85; delta = ratio * length
T_init = 0.1; T_final = 0.001; T = T_init
l = 1000

sol_current = delta
sol_best = sol_current
E_current = E(sol_current)
E_best = E_current
print(E_best)

while T > T_final:

    for i in range(l):

        sol_new = sol_current + (np.random.random(size = np.shape(delta)) - 0.25) * length * T

        if not restrict(sol_new):
            continue

        E_new = E(sol_new)

        if E_new < E_current:

            sol_current = sol_new
            E_current = E_new
            if E_current < E_best:
                sol_best = sol_current
                E_best = E_current
                print(E_best)

        elif random() < exp(-(E_new - E_current) / T):

            sol_current = sol_new
            E_current = E_new

    T = T * alpha
```

```
delta = sol_best
print('sol_best = {}'.format(sol_best))
print('E_best = {}'.format(E_best))

loc_new = init + np.transpose(np.array([delta, delta, delta])) * direction
loc_new = pd.concat([index1 , pd.DataFrame(loc_new, columns = ['nx', 'ny', 'nz'])], axis = 1)

data5 = (
    data4
    .merge(loc_new, left_on = 'n1', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'nx' : 'nx1', 'ny' : 'ny1', 'nz' : 'nz1'})
    .merge(loc_new, left_on = 'n2', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'nx' : 'nx2', 'ny' : 'ny2', 'nz' : 'nz2'})
    .merge(loc_new, left_on = 'n3', right_on = 'n0', how = 'left')
    .drop('n0', axis = 1)
    .rename(columns = {'nx' : 'nx3', 'ny' : 'ny3', 'nz' : 'nz3'})
)

for i in range(len(delta)):
    if direction[i][2] < 0:
        delta[i] = - delta[i]

result = pd.concat([loc_new, pd.DataFrame(delta, columns = ['delta'])], axis = 1)
result.to_csv("result.csv", index = False)
data5.to_csv("data5.csv", index = False)
```

附录 C　吸收效率

```
import pandas as pd
import numpy as np

R = 300.4; F = 0.466 * R
num = 100
# alpha = 0; beta = np.pi / 2
alpha = 36.795 / 180 * np.pi; beta = 78.169 / 180 * np.pi

A = np.array([- np.cos(beta) * np.cos(alpha),
              - np.cos(beta) * np.sin(alpha),
              - np.sin(beta)])

O = A * (R - F)
```

```
gamma = np.array([0, A[2], -A[1]])
eta = np.array([-2 * (A[1] ** 2 + A[2] ** 2) / A[0], A[1], A[2]])
gamma = gamma / np.linalg.norm(gamma)
eta = eta / np.linalg.norm(eta)

r = 0.5

def getSprime(a, b, c):

    return 0.5 * np.linalg.norm(np.cross(b - a, c - a))

def incircle(t):

    return t[0] ** 2 + t[1] ** 2 < r ** 2

def inintercept(a, b, c, t):

    if a[0] == b[0]:
        p1 = a[0]; p2 = c[0]; pt = t[0]
    elif a[1] == b[1]:
        p1 = a[1]; p2 = c[1]; pt = t[1]
    else:
        k = (a[1] - b[1]) / (a[0] - b[0])
        bb = a[1] - k * a[0]
        p1 = bb; p2 = c[1] - k * c[0]; pt = t[1] - k * t[0]

    return (pt <= p2 and pt >= p1) or (pt <= p1 and pt >= p2)

def intriangle(a, b, c, t):

    return inintercept(a, b, c, t) and inintercept(a, c, b, t) and inintercept(b, c, a, t)

def getSi(a, b, c):

    count = 0

    for i in np.linspace(-r, r, num):
        for j in np.linspace(-r, r, num):
            t = np.array([i, j])
            if incircle(t) and intriangle(a, b, c, t):
                count = count + 1

    return count / num ** 2 * 1

def getdata(a, b, c):

    ac = c - a
```

```
    ab = b - a
    N = np.cross(ac, ab) / np.linalg.norm(np.cross(ac, ab))
    if N[2] < 0: N = -N
    S0 = 0.5 * np.linalg.norm(np.cross(ac, ab)) * np.abs(np.dot(A, N))
    B = -2 * np.dot(A, N) * N + A

    def getxprime(x):

        X = x - O
        xp = - np.dot(X, A) * A
        Op = X - np.dot(X, A) * A
        pxprime = (
            np.linalg.norm(xp) * np.linalg.norm(np.cross(B, xp)) / np.dot(B, xp)
            * (np.cross(np.cross(xp, B), xp)
               / np.linalg.norm(np.cross(np.cross(xp, B), xp)))
        )
        xprime = O + Op + pxprime
        return xprime

    def getxstar(xprime):
        return np.array([np.dot(xprime - O, gamma), np.dot(xprime - O, eta)])

    astar = getxstar(getxprime(a))
    bstar = getxstar(getxprime(b))
    cstar = getxstar(getxprime(c))

    Sprime = getSprime(astar, bstar, cstar)
    Si = getSi(astar, bstar, cstar)

    return S0, S0 * Si / Sprime

data5 = pd.read_csv('data5.csv')

for index, row in data5.iterrows():

    a = np.array([row['x1'], row['y1'], row['z1']])
    b = np.array([row['x2'], row['y2'], row['z2']])
    c = np.array([row['x3'], row['y3'], row['z3']])

    na = np.array([row['nx1'], row['ny1'], row['nz1']])
    nb = np.array([row['nx2'], row['ny2'], row['nz2']])
    nc = np.array([row['nx3'], row['ny3'], row['nz3']])

    ia = np.array([row['ix1'], row['iy1'], row['iz1']])
    ib = np.array([row['ix2'], row['iy2'], row['iz2']])
    ic = np.array([row['ix3'], row['iy3'], row['iz3']])
```

```
    S0, Sv = getdata(a, b, c)
    nS0, nSv = getdata(na, nb, nc)
    iS0, iSv = getdata(ia, ib, ic)

    data5.loc[index, 'S0'] = S0; data5.loc[index, 'Sv'] = Sv
    data5.loc[index, 'nS0'] = nS0; data5.loc[index, 'nSv'] = nSv
    data5.loc[index, 'iS0'] = nS0; data5.loc[index, 'iSv'] = iSv

    if index % 100 == 0: print(index)

print('初始球面吸收率：{}%'.format(round(data5.Sv.sum() / data5.S0.sum() * 100, 3) ))
print('实际抛物面吸收率：{}%'.format(round(data5.nSv.sum() / data5.nS0.sum() * 100, 3)))
print('理想抛物面吸收率：{}%'.format(round(data5.iSv.sum() / data5.iS0.sum() * 100, 3)))
```

附录 D　支撑材料列表

1. data10.csv：对附件一和附件二数据的简单整合；
2. data20.csv：对附件三数据的简单整合；
3. 01 数据准备.ipynb：python，初步分析数据、转变为模型所需要的格式；
4. 02 模拟退火.ipynb：python，运行模拟退火算法；
5. 03 吸收效率.ipynb：python，计算馈源舱接收效率；
6. 最优面积.nb：计算三种意义下理想抛物面的焦距。

附录 E　最优面积

```
In[3]:= f[x_,h_]=x^2/(4h)-(h+160.414);
g[x_]=-Sqrt[300.4^2-x^2];
p[x_]=((h+160.414)-x^2/(4h))/150 *x;
Solve[f[x,h]==g[x],x];
h0=300.4*(150/Sqrt[150^2+(150^2/(4h)-(h+160.414))^2]);
h1=-0.5` \[Sqrt](-16.` h (-160.414`+1.` h)-2.` \[Sqrt](64.` h^2 (-160.414`+1.`
h)^2-64.` h^2 (-64507.508604`+320.828` h+1.` h^2)));
h4=0.5` \[Sqrt](-16.` h (-160.414`+1.` h)-2.` \[Sqrt](64.` h^2 (-160.414`+1.`
h)^2-64.` h^2 (-64507.508604`+320.828` h+1.` h^2)));
h2=-0.5` \[Sqrt](-16.` h (-160.414`+1.` h)+2.` \[Sqrt](64.` h^2 (-160.414`+1.`
h)^2-64.` h^2 (-64507.508604`+320.828` h+1.` h^2)));
h3=0.5` \[Sqrt](-16.` h (-160.414`+1.` h)+2.` \[Sqrt](64.` h^2 (-160.414`+1.`
h)^2-64.` h^2 (-64507.508604`+320.828` h+1.` h^2)));
(*写入函数表达式并求得交点坐标*)
Plot[{Abs[-0.5` \[Sqrt](-16.` h (-160.414`+1.` h)-2.` \[Sqrt](64.` h^2
(-160.414`+1.` h)^2-64.` h^2 (-64507.508604`+320.828` h+1.` h^2)))],Abs[-0.5` \
[Sqrt](-16.` h (-160.414`+1.` h)+2.` \[Sqrt](64.` h^2 (-160.414`+1.` h)^2-64.`
h^2 (-64507.508604`+320.828` h+1.` h^2)))]},{h,140,140.586},PlotLegends-
>"Expressions"]
(*判断交点的存在性*)
l[x_,h_]=-160.414` x-h x+x^3/(12 h)+0.5` x Sqrt[90240.15999999999` -1.`
x^2]+45120.079999999994` ArcSin[0.0033288948069241015` x];
k[x_,h_]=160.414` x+1.` h x+0.5347133333333334` x^2+0.033333333333333335` h
x^2-(0.08333333333333333` x^3)/h-(0.0004166666666666667` x^4)/h;
(*求积分并写出原函数，减少下一步计算的复杂度*)
In[183]:= Minimize[{Abs[l[h1,h]-l[-150,h]]+Abs[l[h1,h]-l[h2,h]]+Abs[l[h2,h]-
l[h3,h]]+Abs[l[h3,h]-l[h4,h]]+Abs[l[h4,h]-l[150,h]]+Abs[k[-h0,h]-
k[-150,h]]+Abs[k[h0,h]-k[150,h]],h>140,h<140.586},h]
Out[183]= {152.692,{h->140.364}}
(*枚举搜索最优解*)
In[14]:= Plot[{Abs[h-139.986],Abs[300.4-2Sqrt[160.414h]],Abs[300.4-
Sqrt[22500^2/(16*h^2)+(h+160.414)^2-11250*(160.414-h)/h]]},
{h,140.25,140.4},PlotLegends->"Expressions"]
Abs[140.364-139.986]
\[Placeholder]
(*判断a的松弛性对h松弛性的影响*)
In[26]:= L={};
For[a=0.336,a<0.4,a+=0.001,AppendTo[L,{a,Minimize[{Abs[l[h1,h]-
l[-150,h]]+Abs[l[h1,h]-l[h2,h]]+Abs[l[h2,h]-l[h3,h]]+Abs[l[h3,h]-
l[h4,h]]+Abs[l[h4,h]-l[150,h]]+Abs[k[-h0,h]-k[-150,h]]+Abs[k[h0,h]-
k[150,h]],h>140.321,h<139.986+a},h][[1]]}]]
Clear[a];
ListLinePlot[L,AxesLabel->{a,S^*}]
In[81]:= Clear[a]
In[34]:= L={};
For[a=0.35544,a<0.3555,a+=0.000001,AppendTo[L,{a,((a-0.336)/0.366)^2+
((Minimize[{Abs[l[h1,h]-l[-150,h]]+Abs[l[h1,h]-l[h2,h]]+Abs[l[h2,h]-
l[h3,h]]+Abs[l[h3,h]-l[h4,h]]+Abs[l[h4,h]-l[150,h]]+Abs[k[-h0,h]-
k[-150,h]]+Abs[k[h0,h]-k[150,h]],h>140.321,h<139.986+a},h]
[[1]]-152.69243144931534`)/152.69243144931534`)^2}]]
Clear[a]
ListLinePlot[L,AxesLabel->{a,Abs[d-d^*]^2}]
(*给出不同a约束下的优化值搜寻最优解*)
```