# 摘要

本文基于 FAST 的工作原理，通过机理分析、坐标变换等方法，建立了反射面调节优化模型，并利用 L-BFGS 估计 Hessian 矩阵的内敛法、蒙特卡洛算法等，对不同条件下反射光线的接收比进行了探究。

**对于问题一**，基于固定仰角 $\beta$、基准球面球心 $O$ 建立直角坐标系，选取**焦距作为变量**，以球心为原点建立直角坐标系，构建二维理想抛物线方程。以抛物线与基准球面的**均方径向距离最小**作为目标函数，促动器伸长距离不超过 0.6m 为约束条件，构建单**变量优化模型**。将直角坐 t 标转换为球坐标，利用三分搜索算法计算**最优 c 值为 300.8443**。代入抛物线方程，利用抛物面的中心对称性，将理想抛物线绕 $Z$ 轴旋转得理想抛物面方程为 $\mathbf{x^2 + y^2 = 561.7228(z + 300.8443)}$。并对约束条件加以检验，各促动器径向伸缩量均未超过 0.6m，验证了模型的合理性。

**对于问题二**，根据旋转矩阵对原有直角坐标系进行旋转，使 $Z$ 轴方向正对被观测体。其次，以主索节点与对应理想抛物面上的点的垂直距离平方和最小为目标函数；考虑促动器伸缩长度不超过 0.6m，相邻节点相对变化不超过 0.07% 为约束条件，构建优化模型。应用 L-BFGS 算法进行求解。得到理想抛物面**顶点坐标为 (-49.392,-36.944,-294.453)**。工作节点与理想抛物面的径向**误差不超过 $\mathbf{1\times10^{-5}m}$**。最后对结果加以验证，得出促动器伸长量在-0.5m 到 0.3m 内；主索相对变化量均在 0.07% 内，验证了结果的准确性。

**对于问题三**，经过**旋转变换**，将倾斜入射的光线转化为垂直入射光线。其次，根据蒙特卡洛随机的思想，在三个主索节点构成的三角平面上随机取点，并将其**投影至对应球面反射板上**，得到真实的反射点坐标。通过入射反射光线间向量关系，构建反射光线的空间直线方程。再次，联立反射光线与接收平面所在高度，得出其是否能被馈源舱接收。将接收比的计算转化为，能被接收光线与全部光线的**数量之比**。应用**蒙特卡洛算法**，分别对反射球面以及基准球面的接收比进行求解。得到基准球面**接收比为 5.5%**，调节后的反射球面**接收比为 67.5%**。

**本文优点**：(1) 运用三分搜索算法替代遍历搜索算法，降低了时间复杂度。(2) 应用 L-BFGS 估计 Hessian 矩阵的内敛法，减少了在计算过程中所占用的内存空间。(3) 将三角平面上的随机点投影到球面反射板上，降低了随机点选取的难度。



**关键字**： FAST 主动反射面　优化模型　L-BFGS 估计 Hessian 矩阵　蒙特卡洛算法

# 目录

# 一、 问题重述

## 1.1 背景资料

FAST（Five-hundred-meter Aperture Spherical Telescope）是中国建造的世界上最大的单口径射电望远镜，也是当前全球最敏感、最具竞争力的射电望远镜之一。它位于中国贵州省的大方县，于 2016 年开始建设，于 2016 年 9 月 25 日正式投入使用。FAST 采用了巨大的球面反射面，其直径达 500 米，超过了之前世界上最大的射电望远镜——位于波多黎各的阿雷西博天文台（Arecibo Observatory）的直径。这种巨大的口径赋予 FAST 极高的灵敏度和分辨率，使其在探测宇宙射电信号和研究天文学的各个领域方面有着极其重要的作用。

**问题一：**当待观测天体 $S$ 位于基准球面正上方，$\alpha = 0°$,$\beta = 90°$ 时，结合考虑反射面板调节因素，确定理想抛物面。

**问题二：**当待观测天体　位于 $\alpha = 36.795°$,$\beta = 78.169°$ 时，确定理想抛物面。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。

**问题三：**基于问题二的反射面调节方案，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。

# 二、 问题分析

## 2.1 问题一分析

题目要求当 $\alpha = 0°$,$\beta = 90°$ 时求解理想抛物面，可以利用抛物面中心对称的性质，求解抛物线并将其沿坐标轴旋转一周得到。对于理想抛物线的求解，可以转化为对最优焦距值的优化问题进行求解。

## 2.2 问题二分析

题目要求当待观测天体位于 $\alpha = 36.795°$，$\beta = 78.169°$ 时，确定理想抛物面。可以通过坐标系的旋转，使得被观测体在反射球面正上方。以理想抛物面与反射球面的误差值最小为目标函数，构建优化模型进行求解。

## 2.3 问题三分析

为求整个反射平面接收比，首先考虑单个反射面板的光线接收情况。且注意到每块反射板并非为平面，而是近似球面形状。在此前提下将面积之比，转化为能否被馈源舱接收的反射点数量之比。

# 三、 模型的假设

- 光线在反射面仅进行一次反射。
- 在光线反射时不考虑反射板间隙的影响。
- 光线沿直线传播。
- 被观测体在近似无穷远处，入射光线为平行光。
- 光线反射为全反射。

# 四、 符号说明

| 符号 | 意义 |
|---|---|
| $R$ | 射电望远镜球面半径 |
| $F$ | 抛物面及旋转抛物面的焦距 |
| $L_c(i)$ | 第 $i$ 个促动器的基态长度 |
| $L_s(i)$ | 第 $i$ 个促动器顶端与主索节点距离 |
| $(x_i, y_i, z_i)$ | 序号为 $i$ 的主索节点坐标 |
| $(x_i^-, y_i^-, z_i^-)$ | 序号为 $i$ 的促动器底端坐标 |
| $(x_i^\circ\ , y_i^\circ\ , z_i^\circ\ )$ | 序号为 $i$ 的促动器顶端基态坐标 |
| $(x_i^+, y_i^+, z_i^+)$ | 序号为 $i$ 的促动器顶端工作态坐标 |
| $\eta_g$ | 反射球面接收比 |
| $\eta_b$ | 基准球面接收比 |

# 五、 模型的建立与求解

## 5.1 问题一模型的建立及求解

### 5.1.1 模型建立

**1. 直角坐标系下构建旋转抛物面方程**

题目要求求解考虑反射面板调节因素下的理想抛物面。为求解该抛物面方程，首先构建抛物线方程，再将其绕对称轴进行旋转得出抛物面方程。因此本文首先基于 $\beta = \pi/2$，构建二维抛物线方程，再将其绕 $Z$ 轴旋转，确定三维理想抛物面方程。

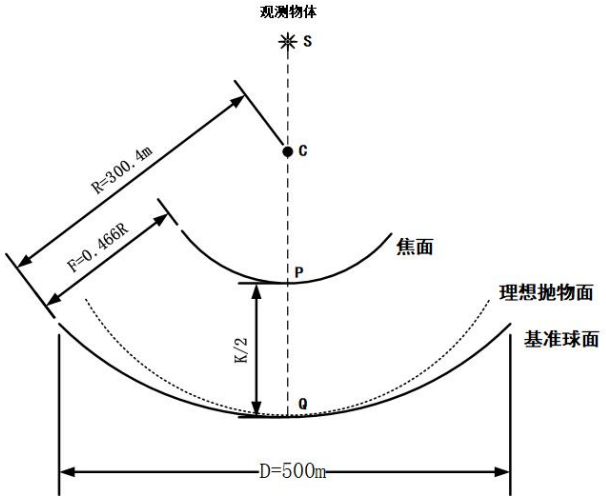



图 **1** **FAST** 截面示意图

本文以基准球面的球心为坐标原点，构建上述截面图中的二维直角坐标系，以水平向右为 $X$ 轴正方向，沿直线 $CS$ 竖直向上为 $Z$ 轴正方向。不难得出，抛物线交点为 $P$。其坐标如下。

$$(x_P, z_P) = (0, -(R - F)) \tag{1}$$

由抛物线的性质可知, 设抛物线焦距为 $K$，则顶点与交点间的距离为 $0.5K$。因此可得其顶点 $Q$ 坐标如下。

$$(x_Q, z_Q) = (0, -(R - F) - 0.5K) \tag{2}$$

设该抛物线方程通式为

$$x^2 = 2p(z + c) \tag{3}$$

根据上式，代入顶点坐标可得 $c = (R - F) + 0.5K$。因为抛物线焦距为 $K$，则通式中 $p = K$。则对于此抛物线，其方程如下。

$$x^2 = 2K[z - (R - F) - 0.5K] \tag{4}$$

将该抛物线 $F(x, z)$ 绕 $Z$ 轴旋转一周，即将式中 $x^2$ 变为 $x^2 + y^2$，至此得到直角坐标系下的抛物面方程。

$$x^2 + y^2 = 2K[z - (R - F) - 0.5K] \tag{5}$$

**2. 理想抛物面**

为求解理想抛物面，首先对 FAST 截面加以分析，将求解所得理想抛物线绕 $Z$ 轴旋转一周即可得到理想抛物面。在题目所给示例中，基准球面的半径为 300m，但在所给附件中，将全部节点半径取平均值为 300.4m。则在此题计算中基准球面半径取 $R$=300.4m。

在此基础上即可求出馈源舱，即焦点坐标如下。

$$(x_P, z_P) = (0, 0, -160.4136) \tag{6}$$

则有 $\frac{p}{2} - c = -160.4136$。因此，为了更接近理想抛物面，将该问题转化为求解 $c$ 的优化问题。

**(1) 目标函数**

为求解理想抛物线，应首先确定目标函数。对于不同的 $c$ 值，即不同的焦距 $K$ 时，抛物线与基准球面的均方径向距离不同。而对于理想抛物线，其更应贴合理想球面。示意图如下。因此，理想抛物线与基准球面的均方径向距离应最小。考虑到该问题中的距

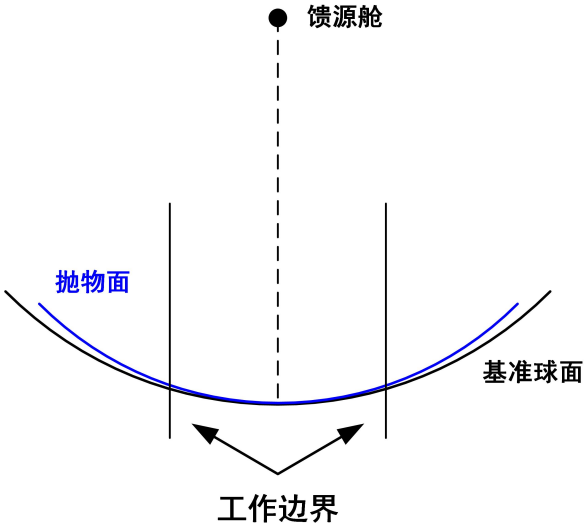



图 **2**　理想抛物线

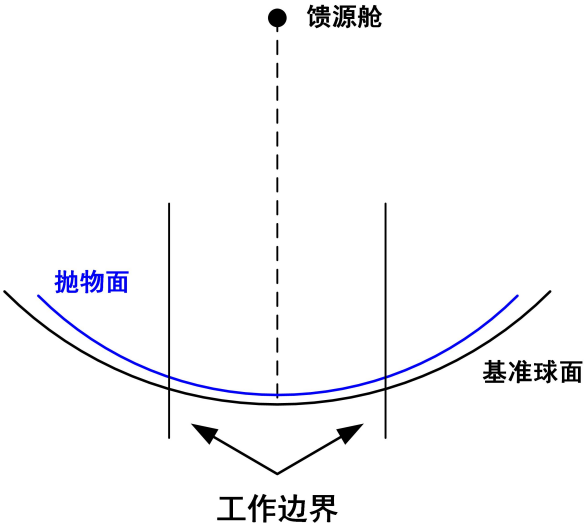



图 **3**　非理想抛物线

离均为径向距离，则将直角坐标系转换为球坐标系进行求解。

$$\begin{cases} \rho = \sqrt{x^2 + y^2} \\ \tan\theta = \frac{y}{x} \end{cases} \tag{7}$$

则在球坐标系下，该目标函数转换为如下形式。

$$min \ \frac{1}{m} \sum_{i=1}^{m} r_i(\theta_i) \tag{8}$$

$$r(\theta) = \left( \frac{2p\sin\theta + \sqrt{4p^2\sin^2\theta + 8p\cos^2\theta}}{2\cos^2\theta} - 300.4 \right)^2 \tag{9}$$

$$p = 2(c - 160.4136) \tag{10}$$

**(2) 约束条件**

由题目可知，径向距离不得超过 0.6m。则有

$$r_i(\theta_i) \leq 0.6^2, i = 1, 2, \cdots, m \tag{11}$$

**3. 模型总述**

综上所述，求解理想抛物面的优化模型如下。

$$\begin{aligned} & min \quad \frac{1}{m}\sum_{i=1}^{m} r_i(\theta_i) \\ s.t. & \begin{cases} r(\theta) = \left(\frac{2p\sin\theta+\sqrt{4p^2\sin^2\theta+8p\cos^2\theta}}{2\cos^2\theta} - 300.4\right)^2 \\ r_i(\theta_i) \leq 0.6^2, i = 1, 2, \cdots, m \end{cases} \end{aligned} \tag{12}$$

### 5.1.2 模型求解

根据上述模型建立过程可知，抛物面焦距 $K$ 的大小影响其与基准球面的相似程度，若 $K$ 值过大，抛物面会低于基准球面；反之，则会高于基准球面。则应选取算法对上述目标函数进行求解，得出最优 $c$ 值，进而得出最优焦距的大小，使抛物面与基准球面的误差最小。对此，本文选用了三分搜索算法进行求解。算法如下：

**Algorithm 1:** 三分搜索算法

**Input:** 函数 $f(x)$，搜索区间 $[a, b]$，精度 $e$

**Output:** 极小值点 $x^*$

```
while b − a > e do
    c1 ← a + (b−a)/3; c2 ← b − (b−a)/3;
    if f(c1) < f(c2) then
        b ← c2
    end
    else
        a ← c1
    end
end
x* ← (a+b)/2;
return x*;
```

求解具体步骤如下。

- step1: 将所给节点坐标转化为球坐标形式。
- step2: 筛选出在工作区域内的主索节点，即 $\theta_i \leq -1.074rad$ 的点。
- step3: 运用三分搜索算法求解结果。

### 5.1.3 结果分析与检验

运用上述三分搜索算法对最优 $c$ 值进行求解，得到如下图像。

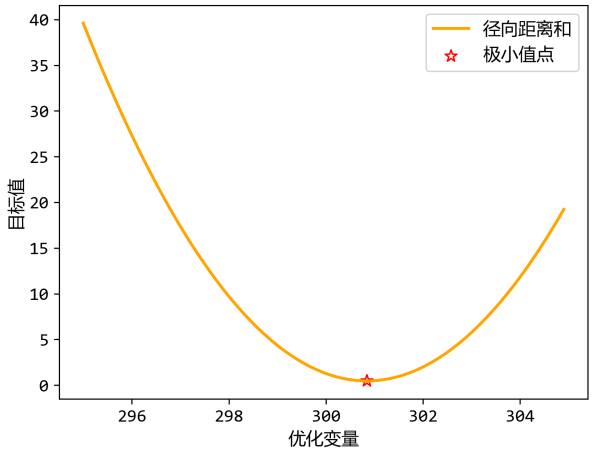



图 **4**　求解结果

由上图可知在 $c = 300.8443$ 时，目标函数得到最优解，此时抛物面与基准球面的最小误差为 0.4649。最后将 $c$ 值代入式 (3) 中可得理想抛物线的表达式如下。

$$x^2 = 561.7228(z + 300.8443) \tag{13}$$

将该抛物线绕 $Z$ 轴旋转一周得到理想抛物面表达式如下。

$$x^2 + y^2 = 561.7228(z + 300.8443) \tag{14}$$

对所得理想抛物面表达式进行可视化分析。得出理想抛物面最低点距离焦点 300.8443m。为检验结果的合理性，我们计算了各个促动器的径向伸缩距离。

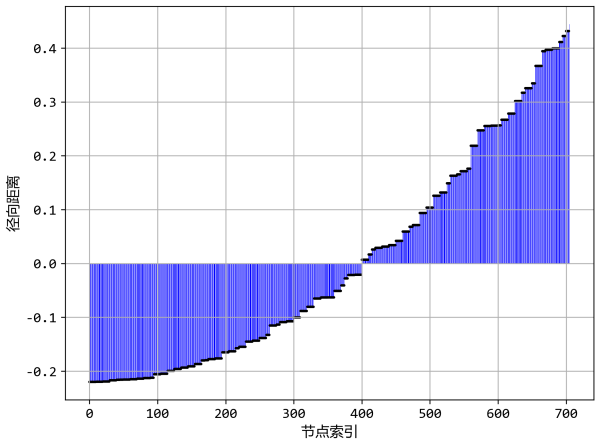
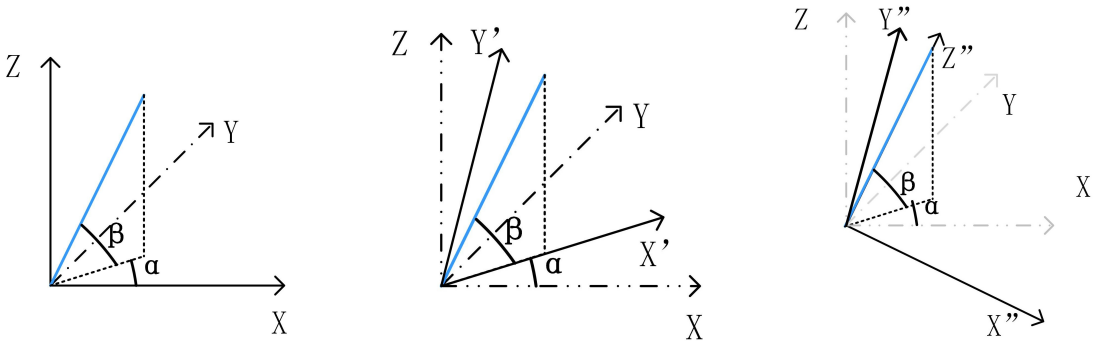



图 **5**　各促动器径向伸缩距离

由上图可知各个促动器的径向伸缩距离均未超过 0.6m，因此所得结果满足该约束条件。结果具有较高的合理性。

### 5.2 问题二模型的建立及求解

本题中, 由于观测天体 $S$ 的方位对于基准球心 $C$ 存在与竖直方向的倾角, 这给构建新的理想抛物面带来了一定的困难。但在求解理想抛物面时, 对于不同轴线方向的抛物面来说, 其焦距、焦点等性质都是不变的。故将现有空间坐标系, 以基准球面的球心 $C$ 为中心、沿基准球面的剖面方向进行旋转, 使得问题二中理想抛物面的轴线与旋转后空间直角坐标系 $Z$ 轴所在直线重合，此时主索节点与理想抛物面在径向方向的距离差值可沿用问题一中的公式。




**(a)** 旋转前　**(b)** 绕 **Z** 轴旋转 $\alpha$　**(c)** 绕 **Y** 轴旋转 $\frac{\pi}{2}-\beta$

图 **6**　坐标系旋转

#### 5.2.1 模型建立

**1. 坐标系旋转**

本题中天体位于 $\alpha=36.795°, \beta=78.169°$，处，则在问题一的基础上，应对原有直角坐标系进行旋转，使 $Z$ 轴正方向沿直线 $CS$ 方向。因此引入旋转矩阵 $R_Z(\alpha)$ 和 $R_Y(\beta)$。由上述旋转图可知，该直角坐标系绕 Z 轴旋转了 $\alpha$ 角，则此时旋转矩阵 $R_Z(\alpha)$ 如下。

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

在经过绕 Z 轴旋转后，再将其绕 $Y$ 轴旋转 $\frac{\pi}{2}-\beta$ 角，其旋转矩阵如下。

$$R_y(\beta)=\begin{bmatrix}\cos\left(\frac{\pi}{2}-\beta\right) & 0 & -\sin\left(\frac{\pi}{2}-\beta\right)\\ 0 & 1 & 0\\ \sin\left(\frac{\pi}{2}-\beta\right) & 0 & \cos\left(\frac{\pi}{2}-\beta\right)\end{bmatrix}=\begin{bmatrix}\sin\beta & 0 & -\cos\beta\\ 0 & 1 & 0\\ \cos\beta & 0 & \sin\beta\end{bmatrix} \tag{16}$$

则坐标系中各个点的坐标，经两次旋转后，变化如下。

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_z(\alpha)R_y(\beta)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\alpha\sin\beta & \sin\alpha\sin\beta & -\cos\beta \\ -\sin\alpha & \cos\alpha & 0 \\ \cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{17}$$

**2. 优化问题**

**(1) 目标函数**

根据变换后的坐标系构建目标函数。将求解抛物面的问题转化为优化问题进行求解。因此，目标是工作区域内的每一个三角球面都尽可能地贴近理想抛物面，这个目标显然过于抽象，于是可近似将所有的主索节点尽可能贴近理想抛物面。因为存在约束条件，所以实际不是所有节点都能再理想抛物面上。因此将目标转化为，求解理想抛物面的 $Z$ 轴坐标与反射面的 $Z$ 轴坐标的平均误差最小。

设主索节点 $D_i(x_i, y_i, z_i)$, 与抛物面上的一点 $P_i(x_i, y_i, h_i)$。并将抛物面进行变换得

$$f(x, y) = \frac{1}{2p}(x^2 + y^2 - 2pc) = k_1x^2 + k_1y^2 + k_2 \tag{18}$$

其中 $k_1 = 0.00178024$，$k_2 = -300.8443$；因此可以构建上述目标函数如下。

$$min \quad \frac{1}{m}\sum_{i=1}^{m}(z_i - k_1x_i^2 - k_1y_i^2 - k_2) \tag{19}$$

**(2) 约束条件**

**• 相对变化不超过 0.07%**

由题目所给附录可知主索节点调节后，相邻节点之间的距离可能会发生微小变化，但变化幅度不超过 0.07%。定义任意两节点 $D_i(x_i, y_i, z_i), D_j(x_j, y_j, z_j)$ 间的距离函数，且相对变化不能超过 0.07%。引入中间优化变量，$e_{ij}$ 代表 $D_i, D_j$ 之间的原始距离，$e'_{ij}$ 代表 $D'_i, D'_j$ 之间的距离。

$$e_{ij} = d(D_i, D_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$e_{ij} = d(D_i, D_j) = \sqrt{(x'_i - x'_j)^2 + (y'_i - y'_j)^2 + (z'_i - z'_j)^2}$$

$$\left|\frac{e'_{ij} - e_{ij}}{e_{ij}}\right| \le 0.07\%$$

$$\begin{cases} e'^2_{ij} \le (1 + 7\times10^{-4})^2 e^2_{ij} \\ e'^2_{ij} \ge (1 - 7\times10^{-4})^2 e^2_{ij} \end{cases} \tag{20}$$

**• 促动器的伸缩长度不超过 0.6**

促动器可以进行控制和位置反馈的伸缩装置，一端与地锚铰接，一端与连接索网活动节点的下拉索铰接。根据控制信号命令促动器产生下拉索拉力，通过改变自身长度而改变地锚点与索网活动节点下拉索端头的间距，实现 FAST 主动反射面的面形调整。

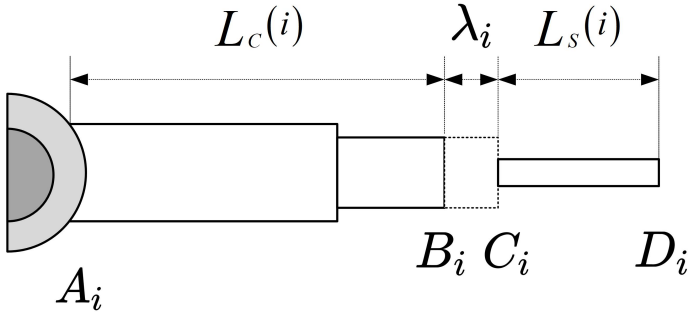



图 **7**　第 $i$ 个促动器示意图

其中 $A_i$ 为基座、$B_i$ 为促动器处于基态时的顶端、$C_i$ 为促动器处于工作态时的顶端、$D_i$ 为主索节点。上图中 $L_c(i)$、$L_s(i)$ 为第 $i$ 个促动器的基态长度以及顶端距主索节点的距离，二者均为常量。

$$L_c(i) = \left|\overrightarrow{AB}\right|$$

$$L_s(i) = \left|\overrightarrow{CD}\right|$$

促动器工作时，其伸缩量 $\lambda_i$ 发生变化。对于单个促动器进行分析，设 $A_i(x_i^-, y_i^-, z_i^-)$，$B_i(x_i^\circ, y_i^\circ, z_i^\circ)$，$C_i(x_i^+, y_i^+, z_i^+)$，$D_i(x_i, y_i, z_i)$。

$$\lambda_i\overrightarrow{AB} = L_c(i)\overrightarrow{BC} \tag{21}$$

$$\begin{cases} x_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)}x_i^\circ - \dfrac{\lambda_i}{L_c(i)}x_i^- \\ y_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)}y_i^\circ - \dfrac{\lambda_i}{L_c(i)}y_i^- \\ z_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)}z_i^\circ - \dfrac{\lambda_i}{L_c(i)}z_i^- \end{cases} \tag{22}$$

由于 $L_s(i)$ 为定值，则有

$$(x_i - x^+)^2 + (y_i - y^+)^2 + (y_i - y^+)^2 = L_s(i)^2 \tag{23}$$

根据题目要求，促动器的伸缩长度不得超过 0.6m，因此伸缩量 $\lambda_i$ 应在-0.6m 到 +0.6m 之间，故应有如下约束。

$$-0.6 \le \lambda_i \le 0.6 \tag{24}$$

**3. 模型总述**

综上所述，本文建立的反射面板优化调节模型如下。

$$
\begin{aligned}
&min \quad \frac{1}{m}\sum_{i=1}^{m}(z_i - k_1 x_i^2 - k_1 y_i^2 - k_2) \\
s.t. &\begin{cases}
\begin{cases} e'_{ij} \le (1 + 7\times 10^{-4}) e_{ij} \\ e'_{ij} \ge (1 - 7\times 10^{-4}) e_{ij} \end{cases} \\
\begin{cases} x_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)} x_i^\circ - \dfrac{\lambda_i}{L_c(i)} x_i^- \\ y_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)} y_i^\circ - \dfrac{\lambda_i}{L_c(i)} y_i^- \\ z_i^+ = \dfrac{L_c(i)+\lambda_i}{L_c(i)} z_i^\circ - \dfrac{\lambda_i}{L_c(i)} z_i^- \end{cases} \\
(x_i - x^+)^2 + (y_i - y^+)^2 + (y_i - y^+)^2 = L_s(i)^2 \\
-0.6 \le \lambda_i \le 0.6 \\
k_1 = 0.00178024, k_2 = -300.8443
\end{cases}
\end{aligned} \tag{25}
$$

### 5.2.2 模型求解

在此问题中，经过上述过程化简，将 2226 个主索节点的 $xyz$ 坐标，减少至仅需求解 692 个主索节点的坐标，对应 1986 条边。因此解决该数量较大，且具有非线性等式与不等式的问题，需要寻找速度快，适应性高的算法进行求解。

BFGS 方法需要存储和计算 $B_k$，而在 L-BFGS 方法中，不需要直接存储和计算 $B_k$，而是使用保存在存储器中的 m 个最新向量对 $\{s_k, y_k\}$ 对 $B_k$ 进行更新，只需要存储两个 $m \times d$ 的矩阵，节省了大量的存储空间。

**Algorithm 2:** L-BFGS 算法

**Input:** 初始点 $w_0$，误差 $e > 0$，最大迭代次数 $k_{\max}$，存储最近的 $m$ 次迭代数据.

**Output:** 优化后的参数 $w^*$

$k = 1$;
计算梯度 $g_k = \nabla f(w_k)$;
**while** $||g_k|| > e$ **and** $k \le k_{\max}$ **do**
    计算搜索方向：$p_k = -H_k g_k$;
    根据一维线搜索找到步长 $\alpha_k > 0$，使得 $f(w_k + \alpha_k p_k) = \min_{\alpha \ge 0} f(w_k + \alpha p_k)$;
    更新参数：$w_{k+1} = w_k + \alpha_k p_k$;
    $k = k + 1$;
**end**
**return** $w^* = w_k$;

### 5.2.3 结果分析与检验

首先根据工作口径为 300m，筛选得出在工作区域内主索节点共有 692 个，对应下拉索共有 1986 条。在坐标进行旋转变换的条件下，得到理想抛物面的顶点坐标为

$(-49.39297184 - 36.94394168 - 294.45334494)$。在 MATLAB 中运用上述算法进行求解得到其工作区域以及主索节点的相对高度如下图所示。

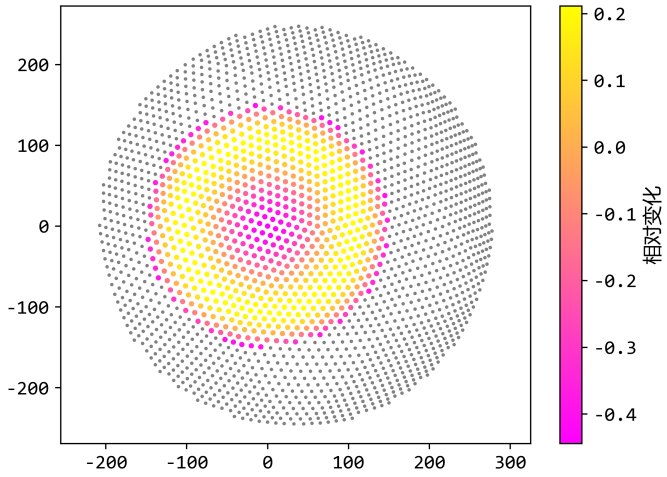



图 **8**　主索节点相对高度

为了进一步验证该模型计算结果的准确性，我们计算了每个主索节点与该理想抛物面的垂直距离。如下图所示，垂直距离差值在 $-0.3 \times 10^{-5}$m 到 $+1 \times 10^{-5}$m 内，且绝大部分主索节点与理想抛物面间的差值极小。因此，该模型的准确性得以验证且其结果较为准确。

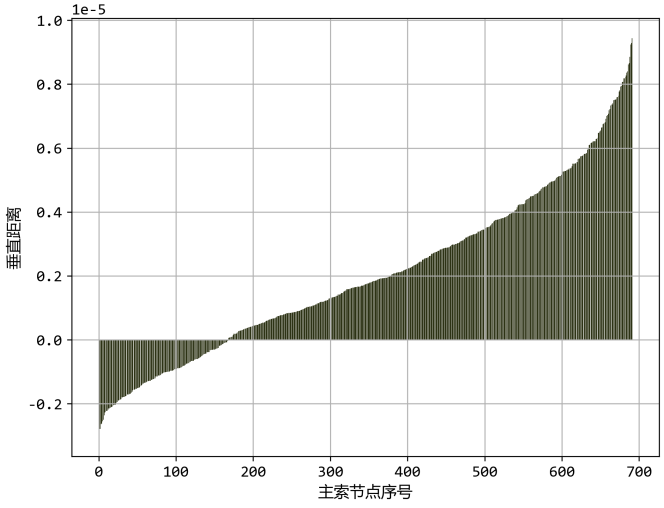



图 **9**　主索节点到理想抛物面的垂直距离

由于该模型存在主索节点相对变化不超过 0.07%，促动器伸缩量不超过 0.6m 的约束，则该结果仍需满足上述条件。因此，我们对每个主索节点的相对变化以及促动器的伸缩量进行检验，结果如下。

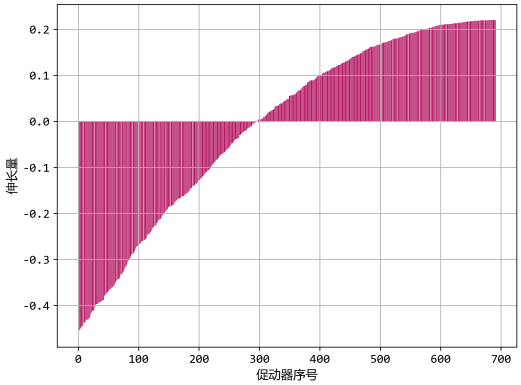



图 **10**　促动器伸缩量

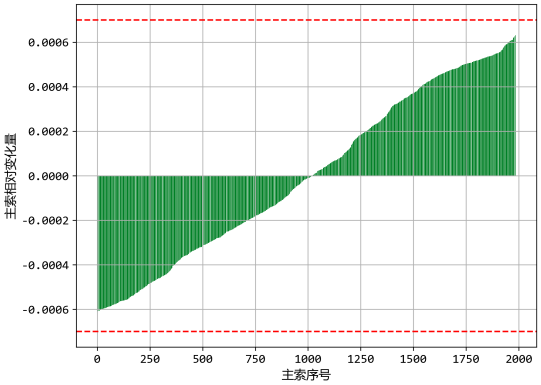



图 **11**　主索相对变化量

由上图可知，促动器的伸缩量在-0.5m 到 +0.3m 内，且主索的相对变化量均未超过 0.07%。满足上述约束条件，具有较高的准确性。因此该最优抛物面在实际应用中结果合理且正确。

### 5.3 问题三模型的建立及求解

题目要求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。首先将坐标系进行旋转变换，将倾斜入射的光线变为垂直入射，以简化问题的分析难度。利用光线的反射定律计算入射与出射光线方程，与馈源舱所在平面相交。具体思路流程如下。

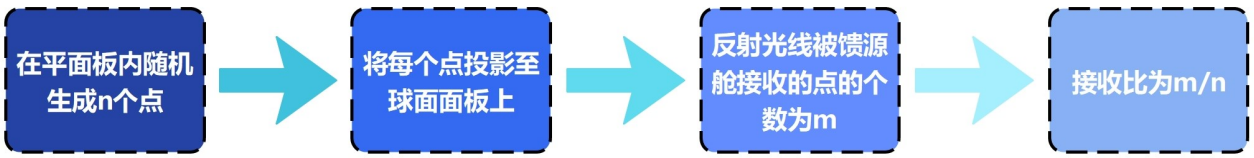



图 **12**　求解接收比流程

#### 5.3.1 模型建立

**1. 生成随机点**

生成 $[0,1]$ 内的两个随机数记为 $s_1$，$s_2$。令

$$a = min\{s_1, s_2\}$$

$$b = max\{s_1, s_2\} - min\{s_1, s_2\}$$

$$c = 1 - max\{s_1, s_2\}$$

设三个主索节点坐标为 $D_1(x_1,y_1,z_1)$,$D_2(x_2,y_2,z_2)$,$D_3(x_3,y_3,z_3)$，则随机点 $(x_t,y_t,z_t)$ 如下。

$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{26}$$

**2. 随机坐标的投影**

对于每个球面反射板，根据三个主索节点坐标，即可求出三个点构成三角形平面时，所在平面的法向量，记为 $N(n_x,n_y,n_z)$。则该平面法线的空间直线方程的参数形式如下。

$$\begin{cases} x = x_t + n_x t \\ y = y_t + n_y t, \quad t\text{为参数} \\ z = z_t + n_z t \end{cases} \tag{27}$$

将其与三个主索节点所在的反射球面方程联立：

$$\begin{cases} x = x_t + n_x t \\ y = y_t + n_y t \\ z = z_t + n_z t \\ (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2 \end{cases} \tag{28}$$

此时，对于三角平面内随机选取的点，可以得到其投影到球面反射板上时，真实反射点的坐标 $K(x_r,y_r,z_r)$。

**3. 反射球面反射模型**

对于理想抛物面，平行于旋转轴入射的光线会聚于焦点处，将馈源舱置于焦点处，则理想抛物面的接收比为 100%. 但在实际工作中，由于每块反射板的形状为一部分球面，即近似弧形。因此，光线经过反射不会全部会聚于馈源舱处，则需对个反射板进行分析。

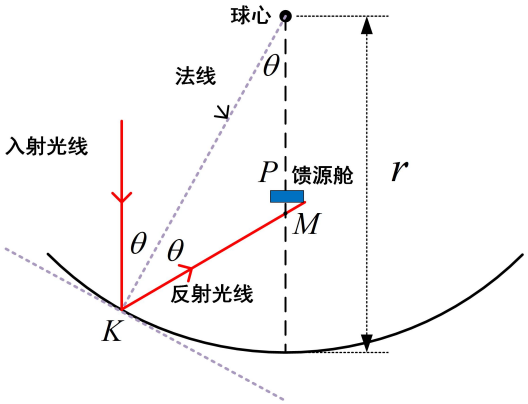



图 **13**　单个球面反射板二维图像

设反射板的三个主索节点坐标分别为 $D_1(x_1,y_1,z_1)$,$D_2(x_2,y_2,z_2)$,$D_3(x_3,y_3,z_3)$。其所在球面的球心为 $O(x',y',z')$ 且半径为 $R \approx 300.4\text{m}$。依此可以求得反射板的球心坐标

$O$。

$$\begin{cases}(x_1-x')^2+(y_1-y')^2+(z_1+z')^2=R^2\\(x_2-x')^2+(y_2-y')^2+(z_2+z')^2=R^2\\(x_3-x')^2+(y_3-y')^2+(z_3+z')^2=R^2\end{cases} \tag{29}$$

设入射光线的方向向量为 $\vec{a}$，出射光线的方向向量为 $\vec{c}$，指向圆心的法线的单位方向向量为 $\vec{b}$。根据几何关系得出

$$\vec{c}=\vec{a}-2\vec{a}\cdot\overrightarrow{bb} \tag{30}$$

由于在问题二中，经过坐标系的旋转变换，此时入射光线垂直于 $Z$ 轴入射。设光线在球面反射板的反射点坐标为 $K(x_r,y_r,z_r)$ 各点关系可以得到 $\vec{a}$、$\vec{b}$、$\vec{c}$ 的具体坐标如下。

$$\vec{a}=(0,0,-k)$$

$$\vec{b}=(\frac{x'-x_r}{R},\frac{y'-y_R}{R},\frac{z'-z_r}{R})$$

$$\vec{c}=(c_{i1},c_{i2},c_{i3})$$

由此可得反射光线所在直线的方程。

$$\frac{x-x_t}{c_{i1}}=\frac{y-y_t}{c_{i2}}=\frac{z-z_t}{c_{i3}} \tag{31}$$

设馈源舱所在平面的高度为 $H$。则将 $z_P=-R+F$ 代入反射光线所在的直线方程，可得该直线与馈源舱所在平面的交点 $E$ 的坐标。

$$\begin{cases}x_E=\frac{c_{i1}}{c_{i3}}[z_p-z_r]+x_r\\y_E=\frac{c_{i1}}{c_{i3}}[z_p-z_r]+y_r\\z_E=z_P\end{cases} \tag{32}$$

馈源舱接收信号的有效区域为直径 1 米的中心圆盘，则要使得反射光线被馈源舱接收，还应有如下约束。

$$x^2+y^2\le 0.5^2 \tag{33}$$

对于求解接收比，可以将其转化为反射面反射光线在馈源舱所在平面的投影面积 $S_a$，与反射面反射光线在馈源舱所在平面的投影面积与馈源舱重叠部分的面积 $S_{ap}$。令接收比为 $\eta_g$。

$$\eta_g=\frac{S_{ap}}{S_a} \tag{34}$$

因为反射板面积无法很好的确定，所以我们将面积之比转换为反射点个数之比。

$$\eta_g = \frac{S_{ap}}{S_a} = \frac{m_1}{n_1} \tag{35}$$

其中，$n_1$ 为所有反射点的个数；$m_1$ 为反射光线能被馈源舱接收时，反射点的个数。在进行上述转换后，可以应用蒙特卡洛算法进行快速求解。

**4. 基准球面反射模型**

由于球面具有对称性，在考虑接收比时即可将问题化简为被观测体在 $\alpha = 0°$，$\beta = 90°$ 时的接收比。

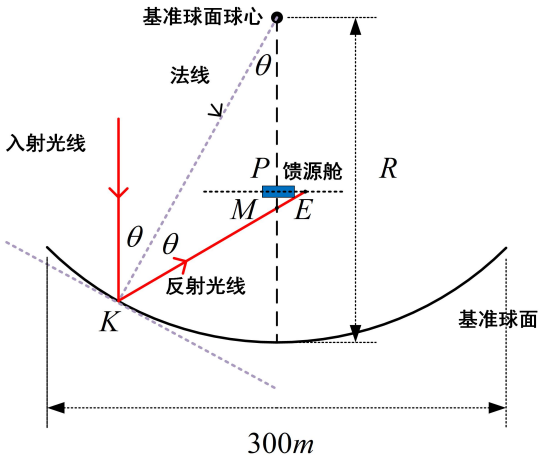



图 **14**　基准球面反射二维图像

与球面反射板反射模型类似，对处在工作平面内的每块板进行反射求解。其过程与球面反射板反射模型相同，但此时各个主索节点坐标为基态时的坐标。在三个主索节点构成三角平面内随机取点，并将其投影至球面反射板中。最后应用球面反射板反射模型进行求解。其接收比表示为

$$\eta_b = \frac{S_v}{150\pi^2} = \frac{m_2}{n_2} \tag{36}$$

其中 $n_2$，为在三角平面内随机取点的个数；$m_2$ 为反射光线能被馈源舱接收的反射点的个数。

### 5.3.2 模型求解

对于该问题中面积比的求解方法，我们选用了蒙特卡洛算法进行求解。在该算法中，要对反射板上的反射点进行随机选取，由于反射板均为球面板，在随机生成反射点较为困难，考虑在平面反射板上随机取点，再将其投影至球面反射板上。其大致流程如下。

即可求得平面上的点投影至球面的点 $K_t(x_t, y_t, z_t)$，并将其作为反射点对反射光线进行求解。算法伪代码如下。

**Algorithm 3:** 蒙特卡洛算法

**Input:** 蒙特卡洛采样次数 $N$，待估计函数 $f(x)$

**Output:** 积分的近似值 $\hat{I}$

$\hat{I} = 0$; **for** $i = 1$ to $N$ **do**

  从 $f$ 的定义域中随机采样一个点 $x_i$; 计算函数值：$y_i = f(x_i)$; $\hat{I} = \hat{I} + y_i$;

**end**

$\hat{I} = \frac{1}{N}\hat{I}$; **return** $\hat{I}$;

### 5.3.3 结果分析与检验

**1. 结果分析**

上述模型及算法进行求解，对每个反射面板上取 2000 模拟光线的照射，分别绘制出了基准球面与调节后的反射球面的反射光线，与馈源舱交点的热力图如下。

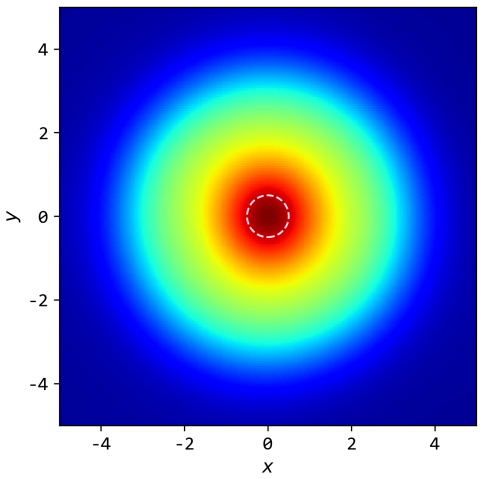



图 **15**　基准球面接收情况

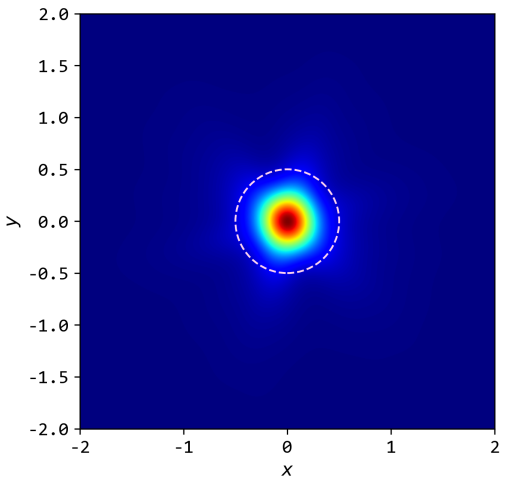



图 **16**　反射球面接收情况

最终求得基准球面的接收比为 5.5%，反射球面接收比为 67.5%。由上图发现球面调整前后，接收区域内的光线密度变化较为明显。接收比提高了约 62%。因此该反射模型的优化效果较优。

**2. 结果检验**

为了检验该模型及算法的准确性及合理性，我们根据蒙特卡洛算法进行了 50 次模拟，并计算每次计算结果与第一次所得结果差值。

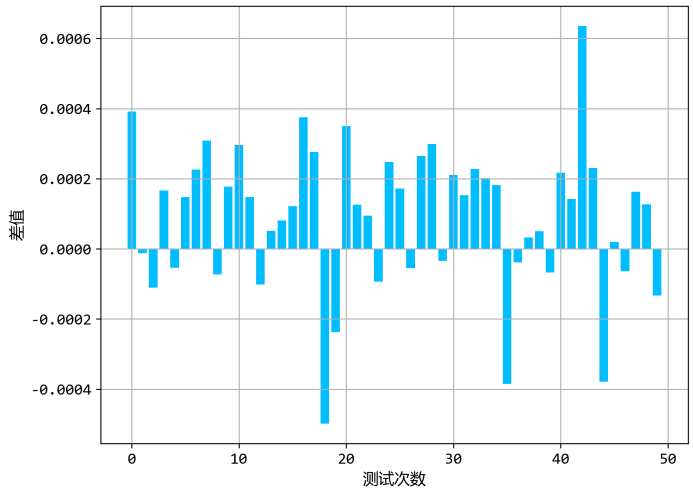



图 **17**　蒙特卡洛算法数值波动

由上图可知，在工作状态下，多次蒙特卡洛算法的数值波动在-0.0006 到 +0.0006 以内，其波动较小。因此，在该模型算法下的结果较为准确与合理。

# 六、 模型评价

## 6.1 模型的优点

- 三分搜索算法替代遍历搜索算法，时间复杂度从 $O(n)$ 降为 $O(\log n)$
- 求解理想抛物面的指标选取合理，并验证了结果的准确性与合理性.
- 应用了 L-BFGS 算法，克服了传统 BFGS 算法占用内存的缺陷, 且避免计算 Hessian 矩阵。
- 求解反射球面的接收比时，对每个反射板单独进行处理，提高了结果的精确程度。应用的蒙特卡洛算法难度较小，但计算速度较快。
- 对接收比的准确性进行分析，发现求解结果精度较高。

## 6.2 模型的缺点

- 在计算反射球面上每个反射板的球心时，半径均取的平均值。因此若对每个反射板的半径单独进行计算，可以进一步提高结果的精确程度。

# 参考文献


[1] 王丹, 毛紫阳, 吴孟达.“FAST”主动反射面的调节 [J]. 数学建模及其应用,2022,11(01):45-53.DOI:10.19943/j.2095-3070.jmmia.2022.01.06.

[2] 南仁东.500m 球反射面射电望远镜 FAST[J]. 中国科学 G 辑: 物理学、力学、天文学,2005(05):3-20.

[3] 南仁东, 李会贤.FAST 的进展——科学、技术与设备 [J]. 中国科学: 物理学力学天文学,2014,44(10):1063-1074.

[4] 陆震, 杨光, 王启明等.FAST 望远镜主动反射面促动机构运动学研究 [J]. 北京航空航天大学学报,2006(02):233-238.DOI:10.13700/j.bh.1001-5965.2006.02.024.

[5] Beavis B , Dobbs I .Optimisation and Stability Theory for Economic Analysis[J].Cambridge Books, 1990.DOI:10.1017/CBO9780511559402.002.

# 七、 优秀论文学习和评述

在完成论文的过程中，我们参考了三篇 2021 年的优秀 A 篇论文，分别是 A028、A115、A217。三篇论文的基本模型都比较相似，主要都是基于几何法和优化规划模型。但是他们在模型的细节处理上以及算法的思路上都各有特点。下面从模型，算法，写作三个角度，对三篇文章的优缺点进行赏析。

在问题一的优化模型的目标确立时，处于对理想抛物面的不同理解，A028 以促动器的总伸缩量最小最为理想抛物面最优的标准，A115 以主索节点变化幅度尽可能小作为优化模型的目标，A217 以理想抛物面到原点的积分与基准球面半径的差值平方的积分作为优化模型的目标。此外，在模型的求解部分，所采用的方法各不相同。A028 采用逐点搜索法，将连续的自变量分解为有限个离散的点，将连续的函数用离散的点来近似。A115 采用变步长的搜索算法。A217 采用二分查找算法。在求解的速度上，A115 明显会更快，并且也能保证一定的精度。而 A028、A217 则会相对慢一些。当解空间更大时，时间上的差距将会更加明显。此外，A028 设定了两类理想抛物面，并给出了相应的方程，能体现其思考的全面。但遗憾的是，由于伸缩量水平只是一个估计值，猜想的正确性需要在问题二中验证，没有办法在该题中直接得到。在结果的分析上，A115 的格式化的处理相对更好，绘制出了理想抛物面与基准球面主索节点变化的三维图和主索节点变化俯视图，更加漂亮直观。而 A028、A217 绘制出了差值的变化曲线，但仍需文字辅助才能更好的理解所给图片的含义。

在问题二的处理上，都是基于第一问的模型，结合坐标系的旋转。但 A217 在算法设计上采用 BFGS 算法，相较于问题一，求解速度有了提升。在结果的可视化方面，A028、A115 给出了包含主索节点编号、各坐标数值的三线表。而 A115 在此基础上绘制了工作抛物面在主动反射面上的位置的三维图，更加直观。而 A217 只给出了主网节点与理想抛物面的球面径向误差及主网节点距离变化率的图像，不够直观美观。

在问题三模型建立部分，三篇文章对于调节后馈源舱的接收比求解思路均有差别。A028 考虑到反射信号不一定被馈源舱完全吸收，故依次定义了单位接收比、单位接收比权重，通过求和得到反射信号接收比。而 A115 则假设反射信号能被全部吸收，虽然简化了模型，提升了计算速度，但结果在精度上被影响，不能更贴切实际。A217 相较于 A028、A115，更重视数学部分，将新坐标系反射点在反馈面板的坐标全部标识出来，给出反射光的标准方程，最后得到了可得接收与反射信号之比的解析式。在算法方面，A028、A217 均采用了蒙特卡洛方法，减少计算难度的同时，保证了结果的精度。另外 A115 给出了敏感度分析，能够说明模型的稳定性。

在写作方面，三篇论文在内容重点突出、结果展示方面都比较出色。行文中均使用三线表，辅以图像，使文章排版美观大方，结果更加直观更具可读性；行文中均对内容重点进行加黑处理，突出文章亮点，增加文章可读性，而三篇论文在算法原理介绍以及

图片配色方案方面有所区别。在算法原理介绍方面，A115、A217 在写作过程中往往较为详尽地介绍算法原理的每一个步骤，而 A028 多是简单概括。在图片配色方面，A115 文章中插图色彩明艳，视觉冲击力强，有力增加了文章的可读性与趣味性，加强了结果的可视性，同时也缓解了读者的视觉疲劳，相比之下，A028、A217 的插图多为黑白配色，较为单调。除此之外，A115 在行文中，将优化模型的各个约束条件汇总在一起，更加的明白直接。

## 八、 赛题解析及经验总结

本题涉及的相关背景和概念较多，作答前应该仔细阅读题目，掌握 fast 望远镜各部分的功能和机理才能更好的把握与解决关键问题。本题题目阅读时需注意一些问题包括：使用题面提供的圆心数据 300m 而不是附件数据的 300.4m；将促动器的工作方向视为径向运动而忽略了其他方向的移动；将反射镜当做平面计算，忽略了反射镜是球面的事实；忽略了下拉索长度不变，而且忽视了主索长度的相对变化。这些问题若处理不当容易造成求解结果错误甚至无法求解，导致时间的滥用。

分析这些现象的原因，并结合本次真题解析中遇到的具体问题，进行了以下三方面的总结。

建模方面，赛题的理解对于模型的正确性和模型求解的性能到最终求解结果的优劣都至关重要。2021A 赛题本身需要用到的数学知识和工具不复杂，总体难度并不太大，主要的困难是求解调节主动反射面的优化模型相对规模较大，于是建模时应力求减小模型的体量，通过变量代换等方法减少优化变量的数目。尽可能地使用强约束，减少冗余约束；尽可能使用等号约束，减少不等约束。本次解析中，通过对模型的优化，去掉了促动器顶端坐标一共 692x3 个优化变量，并将向量的比例约束 (692x3 个) 转换为了下拉索距离不变的约束 (692 个)，极大地优化了模型，使得目标函数由 6m 优化到了 1e-12m。

编程方面，python 的数据处理能力和效果图生成能力强于 matlab，反之 matlab 在专业数学求解的功能与效率上优于 python，两者结合使用可以加快工作率效率。对于优化问题的求解首先应该选对求解器，本次解题耗时最多之处在于求解器的应用上。商用求解器 Gurobi 的通用求解能力行业领先，但是它并不适合用于求解大规模二次规划；智能算法全局搜索能力强，但是在庞大的优化变量和约束条件下也显得疲软。此问题无论是目标函数还是约束条件在数学上都有很好的性质，使用一些基于数学的解法，比如序列二次规划、bfgs 估计的内点法都能有效的进行求解。

写作方面，该赛题所涉及到的概念较多，容易混淆。尤其是各个点的含义，要将它们阐述清楚并不容易。通过对三篇优秀论文的阅读，看出他们的共同点是条理非常清晰，这是我们值得借鉴的地方。首先是摘要部分，第一段写了本篇论文解决的问题以及解法，作为总述。其次，对三个问题分别进行概述，每一段均采用总，分，总的格式，并

给出题目结果。模型建立求解部分，模型的细节阐释得都很清楚，每个符号的含义均作了较为清晰的解释说明。最主要的问题在于图示的清晰程度，图片没有较强的色彩，但每张图都简洁明了，能够以最快的速度帮助读者理解其中的含义，这也是在写作方面我们需要学习的地方。

# 附录 A　问题一三分求解及结果输出

```
import pandas as pd
import numpy as np
import math
import random
import matplotlib
import matplotlib.pyplot as plt
import itertools
import gurobipy as gp

from matplotlib.ticker import ScalarFormatter
from mpl_toolkits.mplot3d import Axes3D
from numpy import sqrt, sin, cos, pi
from numpy.linalg import inv
from numba import jit

plt.style.use('default')                          # 使用默认风格
plt.rcParams['figure.facecolor'] = 'white'        # 将图形的背景颜色设置为白色
plt.rcParams['font.family'] = 'YaHei Consolas Hybrid' # 字体确认
plt.rcParams['font.size'] = 12                    # 字体大小
plt.rcParams['axes.unicode_minus'] = False        # 正常显示负号


NodesDf1 = pd.read_csv('附件1.csv',encoding='GB18030')
TractorInfo = pd.read_csv('附件2.csv',encoding='GB18030')
connectInfo = pd.read_csv('附件3.csv',encoding='GB18030')

## 函数
def dist(p1, p2):
    # 提取点的坐标
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2] - p2[2])**2)

def drawPic(NodesList):
    plt.figure()
    ax1 = plt.axes(projection='3d')
    ax1.set_axis_off()
    plt.tight_layout()
    ax1.scatter(NodesList.iloc[:,1],NodesList.iloc[:,2],NodesList.iloc[:,3], marker = '.',s =
        0.3,edgecolors = 'black',facecolors = 'black')
    ax1.view_init(elev=30, azim=43)
    # 进行坐标轴缩放，使得望远镜看起来更合理
    plt.gca().set_box_aspect((1, 1, 0.4))

    # 三个箭头的起点
    x = np.array([0, 0, 0])
```

```
    y = np.array([0, 0, 0])
    z = np.array([0, 0, 0])

    # 箭头的方向
    arrlen = 10
    u = np.array([10, 0, 0]) # X轴箭头
    v = np.array([0, 10, 0]) # Y轴箭头
    w = np.array([0, 0, 10]) # Z轴箭头

    # 绘制箭头
    ax1.quiver(x, y, z, u, v, w, length=arrlen, normalize=False, color='black')
    # 箭头文字
    # 在箭头终点添加标注文字
    ax1.text(110, 0, 0, 'X', color='black', fontsize=12, ha='center', va='center')
    ax1.text(0, 110, 0, 'Y', color='black', fontsize=12, ha='center', va='center')
    ax1.text(0, 0, 110, 'Z', color='black', fontsize=12, ha='center', va='center')

    # 散点之间连线
    node_dict = {}              # 建立节点对应序号字典

    nodes = NodesList.iloc[:,0]
    for ind,rows in NodesList.iterrows():
        key,x,y,z = rows
        node_dict[key] = (x,y,z)

    for ind, rows in connectInfo.iterrows():
        for (x,y) in list(itertools.combinations(rows,2)):
            X1 = node_dict[x]
            X2 = node_dict[y]
            ax1.plot([X1[0], X2[0]], [X1[1], X2[1]], [X1[2], X2[2]], color='blue',linewidth =
                0.2)

    plt.savefig('fast望远镜.png',format = 'png',dpi=400)


## 第一问：三分求解
NodesDf2 = pd.DataFrame(np.zeros((NodesDf1.shape[0], NodesDf1.shape[1] - 1)),
                        columns=['rho', 'phi', 'theta'])
for ind, rows in NodesDf1.iterrows():
    key, x, y, z = rows
    rho = (x**2 + y**2 + z**2)**0.5
    if (x**2 + y**2)**0.5 == 0:
        phi = 0
    else:
        phi = np.arccos(x / (x**2 + y**2)**0.5)
    theta = np.arcsin(z / (x**2 + y**2 + z**2)**0.5)
    NodesDf2.iloc[ind] = [rho, phi, theta]
```

```
the0 = -np.arccos(150 / 300.4)
WorkingNodes = NodesDf2[NodesDf2['theta'] <= the0]


def dis(the, c):
    p = 2 * (c - 160.4136)
    if np.abs(the + np.pi / 2) < 1e-3:    #垂直情况不能使用通式
        d = (c - 300.4)
    else:
        d = ((2 * p * sin(the) +
            (4 * p * p * sin(the) * sin(the) + 8 * p * c * cos(the) * cos(the))**0.5) /
            (2 * cos(the) * cos(the)) - 300.4)
    return d

def Q1Loss(c):
    loss = 0
    for ind, rows in WorkingNodes.iterrows():
        the = rows[2]
        loss += dis(the, c)**2
    return loss / WorkingNodes.shape[0]

## 使用三分算法求解

l = 295
r = 305
eps = 1e-6

while r - l > eps:
    mid1 = l + (r - l) / 3
    mid2 = r - (r - l) / 3
    if Q1Loss(mid1) < Q1Loss(mid2):
        r = mid2
    else:
        l = mid1

q1ans = (l + r) / 2
print([Q1Loss((l + r) / 2), q1ans])

## 出图准备

l = 295
r = 305
dis_sum = []
for c in np.arange(l,r,0.1):
    dis_sum.append(Q1Loss(c))
```

```
q1sse = []
for ind, rows in WorkingNodes.iterrows():
    the = rows[2]
    q1sse.append(dis(the, q1ans))
q1sse.sort()

## 第一问：结果输出

# 表：验证促动器上端点、主索节点、圆心是否共线

# 计算两个向量
vector1 = (NodesDf1.x - TractorInfo.lx, NodesDf1.y - TractorInfo.ly, NodesDf1.z -
    TractorInfo.lz)
vector2 = (0 - NodesDf1.x, 0 - NodesDf1.y, 0 - NodesDf1.z)
# 向量夹角计算
dot_product = vector1[0] * vector2[0] + vector1[1] * vector2[1] + vector1[2] * vector2[2]
magnitude_vector1 = sqrt(vector1[0]**2 + vector1[1]**2 + vector1[2]**2)
magnitude_vector2 = sqrt(vector2[0]**2 + vector2[1]**2 + vector2[2]**2)
theta = np.arccos(dot_product / (magnitude_vector1 * magnitude_vector2))
deg = theta * 180 / pi

arg = pd.DataFrame(data=[NodesDf1.iloc[np.where(deg >
    5)[0],0].values,deg.values[np.where(deg>=1)]], index=['point','deg'])
arg.to_excel('检查是否共线.xlsx')

# 绘制fast图(时间较长)
# drawPic(NodesDf1)

# 创建折线图
plt.plot(np.arange(1,r,0.1),dis_sum, linestyle='-',color='orange',linewidth=2)
plt.scatter(q1ans,min(dis_sum), marker='*', edgecolors='red', facecolors='none', s=70)
# 添加标题和标签
plt.xlabel('优化变量')
plt.ylabel('目标值')
plt.legend(['径向距离和','极小值点'])
plt.savefig('./imgs/q1遍历折线图.png',dpi=300)

## 结果检验：检验个点的径向距离
plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, WorkingNodes.shape[0]), q1sse, color='b',width=0.6)
plt.scatter(np.arange(0, WorkingNodes.shape[0]-1),q1sse[:-1],marker='.',edgecolors='black',
    facecolors='none',s=5)
plt.xlabel('节点索引')
plt.ylabel('径向距离')
plt.grid(True)
plt.savefig('./imgs/q1各点径向距离.png',dpi=300)
```

# 附录 B　问题二数据处理

```
## 第二问：数据处理

## 确认变换矩阵，求出旋转后抛物面顶点坐标
alpha = 36.795 * np.pi / 180
beta = 78.169 * np.pi / 180

R = np.array([[cos(alpha) * sin(beta),
             sin(alpha) * sin(beta), -cos(beta)],
            [-sin(alpha), cos(alpha), 0],
            [cos(alpha) * cos(beta),
             sin(alpha) * cos(beta),
             sin(beta)]])

## 对所有的主索节点和促动器进行坐标变换

# 主索节点
tranNodes = pd.DataFrame(np.zeros((NodesDf1.shape)),
                         columns=['NodeNames', 'x', 'y', 'z'])
tranNodes.iloc[:, 0] = NodesDf1.iloc[:, 0]
tranNodes.iloc[:, 1:] = np.dot(NodesDf1.iloc[:, 1:], R.T) # 这里只能写成 X @ R.T 它等价于R @ X


# 促动器
tranTractors = pd.DataFrame(np.zeros((TractorInfo.shape)),
                         columns=['NodeNames', 'lx', 'ly', 'lz','ux','uy','uz'])
tranTractors.iloc[:, 0] = TractorInfo.iloc[:, 0]
tranTractors.iloc[:, 1:4] = np.dot(TractorInfo.iloc[:, 1:4], R.T) # 对促动器下端点进行变换
tranTractors.iloc[:, 4:7] = np.dot(TractorInfo.iloc[:, 4:7], R.T) # 对促动器上端点进行变换

## 筛选出工作区域内的主索节点以及促动器

the0 = -np.arccos(150 / 300.4)
tranWorkingNodes = tranNodes[np.arcsin(tranNodes['z'] / (tranNodes['x']**2 + tranNodes['y']**2
    + tranNodes['z']**2)**0.5) <= the0]
tranWorkingTractors = tranTractors.iloc[tranWorkingNodes.index,:]

## 行号重排

tranWorkingNodes = tranWorkingNodes.reset_index(drop = 1)
tranWorkingTractors = tranWorkingTractors.reset_index(drop = 1)

## 计算lc ls
```

```
Lc = np.sqrt((tranWorkingTractors['ux'] - tranWorkingTractors['lx']) ** 2 +
             (tranWorkingTractors['uy'] - tranWorkingTractors['ly']) ** 2 +
             (tranWorkingTractors['uz'] - tranWorkingTractors['lz']) ** 2)

Ls = np.sqrt((tranWorkingNodes['x'] - tranWorkingTractors['ux']) ** 2 +
             (tranWorkingNodes['y'] - tranWorkingTractors['uy']) ** 2 +
             (tranWorkingNodes['z'] - tranWorkingTractors['uz']) ** 2)

## 计算基态工作区域内主索节点对应抛物面上的点
# 定义变换函数
def transform1(row):
    c1 = 561.7228
    c2 = 300.8443
    x, y, z = row[['x','y','z']]
    p = (c1 * x * z + x * ((c1 ** 2 * z ** 2 + 4 * c1 * c2 * (x ** 2 + y ** 2)) ** 0.5)) / (2 *
        (x ** 2 + y ** 2))
    q = (c1 * y * z + y * ((c1 ** 2 * z ** 2 + 4 * c1 * c2 * (x ** 2 + y ** 2)) ** 0.5)) / (2 *
        (x ** 2 + y ** 2))
    r = (c1 * z ** 2 + z * ((c1 ** 2 * z ** 2 + 4 * c1 * c2 * (x ** 2 + y ** 2)) ** 0.5)) / (2
        * (x ** 2 + y ** 2))
    return pd.Series({'p': p, 'q': q, 'r': r})

tranWorkingNodes[['p', 'q', 'r']] = tranWorkingNodes.apply(transform1, axis=1)

## 验证B66号主索节点在坐标变换前的下拉杆长度是否为：8.51507819400385

pos = np.where(NodesDf1.iloc[:,0] == 'B66')
testlen = np.sqrt((TractorInfo['ux'] - NodesDf1['x']) ** 2 +
             (TractorInfo['uy'] - NodesDf1['y']) ** 2 +
             (TractorInfo['uz'] - NodesDf1['z']) ** 2)

########################

# 使用tupledict可以提高检索速度
node_dict = gp.tupledict()                          # 建立节点对应坐标字典
workingEdge_dict = gp.tupledict()                   # 工作区域内主索节点连接情况
working_node_names = set(tranWorkingNodes['NodeNames']) # 工作区域内的点集
node_row_dict = {}                                  # 工作区域内点集对应的行号

for i, node_name in enumerate(tranWorkingNodes['NodeNames']):
    node_row_dict[node_name] = i

for ind,rows in NodesDf1.iterrows():
    key,x,y,z = rows
    node_dict[key] = (x,y,z)

for ind, rows in connectInfo.iterrows():
```

```
    for (x,y) in list(itertools.combinations(rows,2)):     # 三个点取两个
        if x in working_node_names and y in working_node_names:
            #
                准确来说这里应该使用or而不是and，因为边界的主索也应该满足相对长度变化的限制，但是这样子太麻烦了，可以
            x_row = node_row_dict.get(x)
            y_row = node_row_dict.get(y)
            # 对行标进行排序，避免重复
            if x_row > y_row :
                x_row, y_row = y_row, x_row
            workingEdge_dict[(x_row, y_row)] = dist(node_dict[x],node_dict[y])


Nodenums = len(tranWorkingNodes)
Edgenums = len(workingEdge_dict)
c1 = 561.7228
c2 = 300.8443
k1 = 1 / c1
k2 = -c2
R0 = 300.4

# 导出
merged_df = pd.merge(tranWorkingNodes,tranWorkingTractors,on='NodeNames')
merged_df.to_excel('第二问促动器与主索节点信息.xlsx', index=False)
merged_df.iloc[:,1:].to_csv('第二问促动器与主索节点信息.csv', index=False,header=False)

table_data = [(x, y, value) for (x, y), value in workingEdge_dict.items()]
df = pd.DataFrame(table_data, columns=['First Element', 'Second Element', 'Value'])
workingEdge_df = pd.DataFrame(table_data, columns=['node1', 'node2', 'len'])
workingEdge_df.to_excel('第二问工作区内主索长度.xlsx', index=False)
workingEdge_df.to_csv('第二问工作区内主索长度.csv', index=False,header=False)

## 第二问：事前内容输出

testlen.values[pos]
print('旋转后的抛物面顶点坐标为：', [0,0,-300.8443] @ np.linalg.inv(R.T))
print(f"工作区域内主索节点有{Nodenums}个，工作区域内的主索一共有{Edgenums}条")
plt.plot(Lc,linewidth=0.7,color='#DAA21C')
plt.gca().yaxis.set_major_formatter(ScalarFormatter(useOffset=False))
plt.xlabel('主索节点编号')
plt.ylabel('促动器长度')
plt.savefig('./imgs/q2促动器长度.png',dpi=300)
plt.figure()
plt.plot(Ls,linewidth=0.7,color='#002876')
plt.ylabel('下拉索长度')
plt.xlabel('主索节点编号')
plt.savefig('./imgs/q2下拉索长度.png',dpi=300)
```

```
plt.figure()
plt.axis('equal')
plt.scatter(tranNodes.iloc[:,1],tranNodes.iloc[:,2], marker = '.',s = 5,edgecolors =
    'green',facecolors = 'white')
plt.scatter(tranWorkingNodes.iloc[:,1],tranWorkingNodes.iloc[:,2], marker = '*',s =
    7,edgecolors = 'blue',facecolors = 'blue')
plt.savefig('./imgs/q2工作区域.png',dpi=500)


## 第二问：结果读取与输出
# 结果读取
vals = np.loadtxt('./q2result.txt')
wn = np.zeros((Nodenums, 3))
wn[:,0],wn[:,1],wn[:,2] = vals[:Nodenums],vals[Nodenums:2*Nodenums],vals[2*Nodenums:3*Nodenums]
lamb = vals[3*Nodenums:4*Nodenums]

# 绘制伸长量柱状图
plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Nodenums), sorted(lamb), color='#B41764')
plt.xlabel('促动器序号')
plt.ylabel('伸长量')
plt.grid(True)
plt.savefig('./imgs/q2促动器伸长量.png',dpi=300)

# 绘制各主索节点到理想抛物面距离
idealH = k1 * wn[:,0]**2 + k1 * wn[:,1]**2 + k2
dh = wn[:,2] - idealH
# A = k1 * (wn[:, 0]**2 + wn[:, 1]**2)
# B = 2 * A - wn[:, 2]
# C = A + k2 - wn[:, 2]
# t1 = (-B + np.sqrt(B**2 - 4 * A * C)) / (2 * A)
# t2 = (-B - np.sqrt(B**2 - 4 * A * C)) / (2 * A)
# pn = np.array([
#     wn[:, 0] + wn[:, 0] * t1,
#     wn[:, 1] + wn[:, 1] * t1,
#     wn[:, 2] + wn[:, 2] * t1
# ]).T
# ds = np.where(wn[:, 2] - pn[:, 2] > 0, -1, 1) * np.sqrt((wn[:, 0] - pn[:, 0])**2 + (wn[:, 1]
    - pn[:, 1])**2 + (wn[:, 2] - pn[:, 2])**2)
plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Nodenums), sorted(dh), color='#2D3315')
plt.xlabel('主索节点序号')
plt.ylabel('垂直距离')
plt.grid(True)
plt.savefig('./imgs/q2主索节点到理想抛物面的垂直距离.png',dpi=300)

# 绘制俯视图(主索节点相对基态移动距离图)
```

```
# 计算相对距离
dh = wn[:,2] - tranWorkingNodes.z.values
cmap = plt.cm.get_cmap('spring')
dh_min = np.min(dh)
dh_max = np.max(dh)
norm = plt.Normalize(vmin=dh_min, vmax=dh_max)
# cmap = plt.cm.get_cmap('spring')
# ds_min = np.min(ds)
# ds_max = np.max(ds)
# norm = plt.Normalize(vmin=ds_min, vmax=ds_max)
plt.figure()
plt.axis('equal')
plt.scatter(tranNodes.iloc[:,1],tranNodes.iloc[:,2], marker = '.',s = 5,color='gray')
plt.scatter(tranWorkingNodes.iloc[:,1], tranWorkingNodes.iloc[:,2], c=dh, cmap=cmap,
    marker='.', s=20, norm=norm)
cbar = plt.colorbar()
cbar.set_label('相对变化')
plt.savefig('./imgs/q2主索节点相对高度热力图.png',dpi=400)

# 绘制主索相对变化量柱状图
edge_change = []
for key, val in workingEdge_dict.items():
    i, j = key
    curlen = dist((wn[i,0],wn[i,1],wn[i,2]),(wn[j,0],wn[j,1],wn[j,2]))
    edge_change.append((curlen - val) / val)

plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Edgenums), sorted(edge_change), color='#008026')
plt.axhline(y=-0.0007, color='red', linestyle='dashed')
plt.axhline(y=0.0007, color='red', linestyle='dashed')
plt.xlabel('主索序号')
plt.ylabel('主索相对变化量')
plt.grid(True)
plt.savefig('./imgs/q2主索相对变化量.png',dpi=300)

# 具体数据导出
# 将结果变换回去
inv_wn = np.dot(wn, inv(R.T))
outputDf = pd.DataFrame({'Nodename':tranWorkingNodes.iloc[:, 0].values, 'x':inv_wn[:,0],
    'y':inv_wn[:,1], 'z':inv_wn[:,2], 'lamb':lamb})
outputDf.to_excel('q2_result.xlsx',index=False)
```

# 附录 C　问题二 L-BFGS 求解

```
clear
```

```
warning('off')

tranNodes = readtable('第二问促动器与主索节点信息.xlsx');
workingEdge = readtable('第二问工作区内主索长度.xlsx');

Nodenums = size(tranNodes,1);
Edgenums = size(workingEdge,1);
c1 = 561.7228;
c2 = 300.8443;
k1 = 1 / c1;
k2 = -c2;
% 计算lc ls
Lc = sqrt((tranNodes.ux - tranNodes.lx).^2 + ...
        (tranNodes.uy - tranNodes.ly).^2 + ...
        (tranNodes.uz - tranNodes.lz).^2);

Ls = sqrt((tranNodes.x - tranNodes.ux).^2 + ...
        (tranNodes.y - tranNodes.uy).^2 + ...
        (tranNodes.z - tranNodes.uz).^2);

% 优化变量初始值
initial_vars = [tranNodes.x; tranNodes.y; tranNodes.z; zeros(Nodenums, 1)];
%initial_vars = [tranNodes.x; tranNodes.y; k1 .* tranNodes.x.^2 + k1 .* tranNodes.y.^2 + k2;
    zeros(Nodenums, 1)];
result = initial_vars;
% 定义变量上下界
lb = [-305 * ones(3 * Nodenums, 1); -0.6 * ones(Nodenums, 1)];
ub = [155 * ones(3 * Nodenums, 1); 0.6 * ones(Nodenums, 1)];
% 定义目标函数
objective = @(vars) obj(vars, Nodenums);
% 定义非线性约束函数
nonlcon = @(vars) cons(vars, Lc, Ls, tranNodes, workingEdge);

% 求解器选项
options = optimoptions(@fmincon);
options.Algorithm = "interior-point";
options.MaxIterations = Inf;
options.MaxFunctionEvaluations = Inf;
options.Display = 'iter-detailed';
% options.SubproblemAlgorithm = 'cg';
options.HessianApproximation = 'lbfgs';
options.SpecifyConstraintGradient = true;
options.SpecifyObjectiveGradient = false;
options.EnableFeasibilityMode = true;
options.HonorBounds = true;
options.ConstraintTolerance = 1e-12;
options.StepTolerance = 1e-12;
```

```
options.OptimalityTolerance = 1e-12;
% 梯度检查与函数值检查
options.CheckGradients = false;
option.FunValCheck = 'off';

[result, fval] = fmincon(objective, result, [], [], [], [], lb, ub, nonlcon, options);

% 保存结果到文件
writematrix(result,'q2result.txt');

function [f] = obj(vars, Nodenums)
    % 目标函数
    % f:目标函数值
    % grad:梯度
    c1 = 561.7228;
    c2 = 300.8443;
    k1 = 1 / c1;
    k2 = -c2;
    wn = reshape(vars(1:Nodenums*3), Nodenums, 3);

    dh = wn(:, 3) - k1 .* wn(:, 1).^2 - k1 .* wn(:, 2).^2 - k2;
    f = sum(dh.^2);

%     % 对各自变量(3460个)求偏导
%     grad = [-2 .* k1 .* dh .* wn(:,1); ...
%             -2 .* k1 .* dh .* wn(:,2); ...
%             2 .* k1 .* dh; ...
%             zeros(Nodenums, 1)];
end

function [c, ceq, gradc, gradceq] = cons(vars, Lc, Ls, tranNodes, workingEdge)
    % c(x) <= 0
    % ceq(x) = 0
    % 设定常量
    Nodenums = size(tranNodes, 1);
    Edgenums = size(workingEdge, 1);
    % 提取变量
    wn = reshape(vars(1:Nodenums*3), Nodenums, 3);
    lamb = vars(Nodenums*3+1:end);

    % 等式约束
    xpuls = (tranNodes.ux - tranNodes.lx) .* lamb ./ Lc + tranNodes.ux;
    ypuls = (tranNodes.uy - tranNodes.ly) .* lamb ./ Lc + tranNodes.uy;
    zpuls = (tranNodes.uz - tranNodes.lz) .* lamb ./ Lc + tranNodes.uz;
    ceq = (wn(:,1) - xpuls).^2 + ...
          (wn(:,2) - ypuls).^2 + ...
          (wn(:,3) - zpuls).^2 - Ls.^2;
```

```
    % 等式梯度(每一列一个目标函数，每一行一个自变量的偏导)
    dxpuls_dlamb = (tranNodes.ux - tranNodes.lx) ./ Lc;
    dypuls_dlamb = (tranNodes.uy - tranNodes.ly) ./ Lc;
    dzpuls_dlamb = (tranNodes.uz - tranNodes.lz) ./ Lc;
    gradceq = [diag(2 .* (wn(:,1) - xpuls)); ...
               diag(2 .* (wn(:,2) - ypuls)); ...
               diag(2 .* (wn(:,3) - zpuls)); ...
               diag(-2.*(wn(:,1) - xpuls).*dxpuls_dlamb + ...
                    -2.*(wn(:,2) - ypuls).*dypuls_dlamb + ...
                    -2.*(wn(:,3) - zpuls).*dzpuls_dlamb)];

    % 不等约束
    node1 = workingEdge.node1+1;
    node2 = workingEdge.node2+1;
    len = workingEdge.len;
    c = [(0.9993 * len).^2 - ...
         ((wn(node1,1) - wn(node2,1)).^2 + ...
         (wn(node1,2) - wn(node2,2)).^2 + ...
         (wn(node1,3) - wn(node2,3)).^2);
         ...
         ((wn(node1,1) - wn(node2,1)).^2 + ...
         (wn(node1,2) - wn(node2,2)).^2 + ...
         (wn(node1,3) - wn(node2,3)).^2) - ...
         (1.0007 * len).^2];

    % 不等式梯度
    gradc = zeros(length(vars), size(c,1));
    for k = 1:Edgenums
        i = node1(k);
        j = node2(k);
        addposi = [i, Nodenums + i, 2 * Nodenums + i];
        addposj = [j, Nodenums + j, 2 * Nodenums + j];
        gradc(addposi, k) = [-2*(wn(i,1)-wn(j,1)); -2*(wn(i,2)-wn(j,2)); -2*(wn(i,3)-wn(j,3))];
        gradc(addposj, k) = [2*(wn(i,1)-wn(j,1)); 2*(wn(i,2)-wn(j,2)); 2*(wn(i,3)-wn(j,3))];
        gradc(addposi, k + Edgenums) = [2*(wn(i,1)-wn(j,1)), 2*(wn(i,2)-wn(j,2)),
            2*(wn(i,3)-wn(j,3))];
        gradc(addposj, k + Edgenums) = [-2*(wn(i,1)-wn(j,1)), -2*(wn(i,2)-wn(j,2)),
            -2*(wn(i,3)-wn(j,3))];
    end

end
```

# 附录 D　问题二结果读取与输出

```
## 第二问：结果读取与输出
# 结果读取
vals = np.loadtxt('./q2result.txt')
wn = np.zeros((Nodenums, 3))
wn[:,0],wn[:,1],wn[:,2] = vals[:Nodenums],vals[Nodenums:2*Nodenums],vals[2*Nodenums:3*Nodenums]
lamb = vals[3*Nodenums:4*Nodenums]

# 绘制伸长量柱状图
plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Nodenums), sorted(lamb), color='#B41764')
plt.xlabel('促动器序号')
plt.ylabel('伸长量')
plt.grid(True)
plt.savefig('./imgs/q2促动器伸长量.png',dpi=300)

# 绘制各主索节点到理想抛物面距离
idealH = k1 * wn[:,0]**2 + k1 * wn[:,1]**2 + k2
dh = wn[:,2] - idealH
# A = k1 * (wn[:, 0]**2 + wn[:, 1]**2)
# B = 2 * A - wn[:, 2]
# C = A + k2 - wn[:, 2]
# t1 = (-B + np.sqrt(B**2 - 4 * A * C)) / (2 * A)
# t2 = (-B - np.sqrt(B**2 - 4 * A * C)) / (2 * A)
# pn = np.array([
#     wn[:, 0] + wn[:, 0] * t1,
#     wn[:, 1] + wn[:, 1] * t1,
#     wn[:, 2] + wn[:, 2] * t1
# ]).T
# ds = np.where(wn[:, 2] - pn[:, 2] > 0, -1, 1) * np.sqrt((wn[:, 0] - pn[:, 0])**2 + (wn[:, 1]
    - pn[:, 1])**2 + (wn[:, 2] - pn[:, 2])**2)
plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Nodenums), sorted(dh), color='#2D3315')
plt.xlabel('主索节点序号')
plt.ylabel('垂直距离')
plt.grid(True)
plt.savefig('./imgs/q2主索节点到理想抛物面的垂直距离.png',dpi=300)

# 绘制俯视图(主索节点相对基态移动距离图)
# 计算相对距离
dh = wn[:,2] - tranWorkingNodes.z.values
cmap = plt.cm.get_cmap('spring')
dh_min = np.min(dh)
dh_max = np.max(dh)
norm = plt.Normalize(vmin=dh_min, vmax=dh_max)
# cmap = plt.cm.get_cmap('spring')
# ds_min = np.min(ds)
```

```
# ds_max = np.max(ds)
# norm = plt.Normalize(vmin=ds_min, vmax=ds_max)
plt.figure()
plt.axis('equal')
plt.scatter(tranNodes.iloc[:,1],tranNodes.iloc[:,2], marker = '.',s = 5,color='gray')
plt.scatter(tranWorkingNodes.iloc[:,1], tranWorkingNodes.iloc[:,2], c=dh, cmap=cmap,
    marker='.', s=20, norm=norm)
cbar = plt.colorbar()
cbar.set_label('相对变化')
plt.savefig('./imgs/q2主索节点相对高度热力图.png',dpi=400)

# 绘制主索相对变化量柱状图
edge_change = []
for key, val in workingEdge_dict.items():
    i, j = key
    curlen = dist((wn[i,0],wn[i,1],wn[i,2]),(wn[j,0],wn[j,1],wn[j,2]))
    edge_change.append((curlen - val) / val)

plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, Edgenums), sorted(edge_change), color='#008026')
plt.axhline(y=-0.0007, color='red', linestyle='dashed')
plt.axhline(y=0.0007, color='red', linestyle='dashed')
plt.xlabel('主索序号')
plt.ylabel('主索相对变化量')
plt.grid(True)
plt.savefig('./imgs/q2主索相对变化量.png',dpi=300)

# 具体数据导出
# 将结果变换回去
inv_wn = np.dot(wn, inv(R.T))
outputDf = pd.DataFrame({'Nodename':tranWorkingNodes.iloc[:, 0].values, 'x':inv_wn[:,0],
    'y':inv_wn[:,1], 'z':inv_wn[:,2], 'lamb':lamb})
outputDf.to_excel('q2_result.xlsx',index=False)
```

# 附录 E　问题三蒙特卡洛求解与误差分析

```
### 第三问：数据处理

def calc_circle_center_and_radius(p1, p2, p3):
    # 求空间中三点的外接圆圆心和半径
    x1 = p1[0]
    y1 = p1[1]
    z1 = p1[2]
    x2 = p2[0]
    y2 = p2[1]
```

```
    z2 = p2[2]
    x3 = p3[0]
    y3 = p3[1]
    z3 = p3[2]
    a1 = (y1*z2 - y2*z1 - y1*z3 + y3*z1 + y2*z3 - y3*z2)
    b1 = -(x1*z2 - x2*z1 - x1*z3 + x3*z1 + x2*z3 - x3*z2)
    c1 = (x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2)
    d1 = -(x1*y2*z3 - x1*y3*z2 - x2*y1*z3 + x2*y3*z1 + x3*y1*z2 - x3*y2*z1)
    a2 = 2 * (x2 - x1)
    b2 = 2 * (y2 - y1)
    c2 = 2 * (z2 - z1)
    d2 = x1*x1 + y1*y1 + z1*z1 - x2*x2 - y2*y2 - z2*z2
    a3 = 2 * (x3 - x1)
    b3 = 2 * (y3 - y1)
    c3 = 2 * (z3 - z1)
    d3 = x1*x1 + y1*y1 + z1*z1 - x3*x3 - y3*y3 - z3*z3
    x = -(b1*c2*d3 - b1*c3*d2 - b2*c1*d3 + b2*c3*d1 + b3*c1*d2 - b3*c2*d1) /\
        (a1*b2*c3 - a1*b3*c2 - a2*b1*c3 + a2*b3*c1 + a3*b1*c2 - a3*b2*c1)
    y = (a1*c2*d3 - a1*c3*d2 - a2*c1*d3 + a2*c3*d1 + a3*c1*d2 - a3*c2*d1) /\
        (a1*b2*c3 - a1*b3*c2 - a2*b1*c3 + a2*b3*c1 + a3*b1*c2 - a3*b2*c1)
    z = -(a1*b2*d3 - a1*b3*d2 - a2*b1*d3 + a2*b3*d1 + a3*b1*d2 - a3*b2*d1) /\
        (a1*b2*c3 - a1*b3*c2 - a2*b1*c3 + a2*b3*c1 + a3*b1*c2 - a3*b2*c1)
    r = math.sqrt((x1 - x)*(x1 - x) + (y1 - y)
                    * (y1 - y) + (z1 - z)*(z1 - z))
    return x, y, z, r

## 优化前的做一次
workingReflectorLst = []
# 遍历所有反射镜，求出工作区域内的
for ind, rows in connectInfo.iterrows():
    # 三个节点都是工作节点
    if rows[0] in node_row_dict and rows[1] in node_row_dict and rows[2] in node_row_dict:
        # 获取三点位置
        node1 = list(tranWorkingNodes.loc[node_row_dict[rows[0]],['x','y','z']].values)
        node2 = list(tranWorkingNodes.loc[node_row_dict[rows[1]],['x','y','z']].values)
        node3 = list(tranWorkingNodes.loc[node_row_dict[rows[2]],['x','y','z']].values)
        # 计算平面法向量
        n1n2 = (node1[0] - node2[0],node1[1] - node2[1],node1[2] - node2[2])
        n1n3 = (node1[0] - node3[0],node1[1] - node3[1],node1[2] - node3[2])
        n = np.cross(n1n2, n1n3)
        # 计算n的单位向量
        ne = n / (n[0]**2+n[1]**2+n[2]**2)**0.5
        # 计算球心坐标和半径
        x0, y0, z0, r = calc_circle_center_and_radius(node1,node2,node3)

        if (np.array([x0, y0, z0]) + (300.4**2 - r**2)**0.5 * ne)[2] > node1[2]:
            # 圆心的Z值大于三个顶点的Z值
```

```
            X0, Y0, Z0 = np.array([x0, y0, z0]) + (300.4**2 - r**2)**0.5 * ne
        else:
            X0, Y0, Z0 = np.array([x0, y0, z0]) - (300.4**2 - r**2)**0.5 * ne

        workingReflectorLst.append({'name':rows[0]+rows[1]+rows[2],
                                    'node1':node1,'node2':node2,'node3':node3,
                                    'nx':ne[0],'ny':ne[1],'nz':ne[2],
                                    'x0':X0,'y0':Y0,'z0':Z0})

workingReflector = pd.DataFrame(workingReflectorLst)

## 优化后的
workingReflector_optimed = pd.DataFrame(workingReflectorLst)

# 第二问结果读取
vals = np.loadtxt('./q2result.txt')
wn = np.zeros((Nodenums, 3))
wn[:,0],wn[:,1],wn[:,2] = vals[:Nodenums],vals[Nodenums:2*Nodenums],vals[2*Nodenums:3*Nodenums]

workingReflectorLst_optimed = []
# 遍历所有反射镜，求出工作区域内的
for ind, rows in connectInfo.iterrows():
    # 三个节点都是工作节点
    if rows[0] in node_row_dict and rows[1] in node_row_dict and rows[2] in node_row_dict:
        # 获取三点位置
        node1 = wn[node_row_dict[rows[0]]]
        node2 = wn[node_row_dict[rows[1]]]
        node3 = wn[node_row_dict[rows[2]]]
        # 计算平面法向量
        n1n2 = (node1[0] - node2[0],node1[1] - node2[1],node1[2] - node2[2])
        n1n3 = (node1[0] - node3[0],node1[1] - node3[1],node1[2] - node3[2])
        n = np.cross(n1n2, n1n3)
        # 计算n的单位向量
        ne = n / (n[0]**2+n[1]**2+n[2]**2)**0.5
        # 计算球心坐标和半径
        x0, y0, z0, r = calc_circle_center_and_radius(node1,node2,node3)

        if (np.array([x0, y0, z0]) + (300.4**2 - r**2)**0.5 * ne)[2] > node1[2]:
            # 圆心的Z值大于三个顶点的Z值
            X0, Y0, Z0 = np.array([x0, y0, z0]) + (300.4**2 - r**2)**0.5 * ne
        else:
            X0, Y0, Z0 = np.array([x0, y0, z0]) - (300.4**2 - r**2)**0.5 * ne

        workingReflectorLst_optimed.append({'name':rows[0]+rows[1]+rows[2],
                                    'node1':node1,'node2':node2,'node3':node3,
                                    'nx':ne[0],'ny':ne[1],'nz':ne[2],
                                    'x0':X0,'y0':Y0,'z0':Z0})
```

```
# 优化后的
workingReflector_optimed = pd.DataFrame(workingReflectorLst_optimed)



import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy.ndimage.filters import gaussian_filter
from matplotlib.patches import Circle

## 第三问：蒙特卡洛
def myplot(x, y, s=16, bins=800):
    heatmap, xedges, yedges = np.histogram2d(x, y, bins=bins)
    heatmap = gaussian_filter(heatmap, sigma=s)

    extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
    return heatmap.T, extent

@jit(nopython=True)
def MonteCarlo_Reflector(node1, node2, node3, nx, ny, nz, x0, y0, z0, mc_times):
    # node1,node2,node3：反射镜三个点
    # nx,ny,nz：上述三点组成的法向量
    # p1,p2,p3：球面镜的三个点
    # mc_times：蒙特卡洛次数
    # return：命中次数

    R0 = 300.4
    F = 0.466 * R0
    # 馈源舱的高度ze
    ze = -R0 + F
    success_times = 0
    recordPoints_x = []
    recordPoints_y = []
    for i in range(mc_times):
        # 使用重心坐标法在三点间随机取点
        split1 = random.random()
        split2 = random.random()
        split1, split2 = min(split1, split2), max(split1, split2)
        a = split1
        b = split2 - split1
        c = 1 - split2
        #print(a,b,c,a+b+c)
        # 计算随机点
        xt = a * node1[0] + b * node2[0] + c * node3[0]
        yt = a * node1[1] + b * node2[1] + c * node3[1]
```

```
        zt = a * node1[2] + b * node2[2] + c * node3[2]
        # 求空间直线与圆的交点
        A = nx**2 + ny**2 + nz**2
        B = 2 * (nx * (xt - x0) + ny * (yt - y0) + nz * (zt - z0))
        C = (xt - x0)**2 + (yt - y0)**2 + (zt - z0)**2 - R0**2
        delta = B**2 - 4 * A * C
        if delta >= 0:
            t1 = (-B + delta**0.5) / (2 * A)
            t2 = (-B - delta**0.5) / (2 * A)
        else:
            print('方程无解')
            return

        if abs(t1) > abs(t2):
            t = t2
        else:
            t = t1

        xr = xt + nx * t
        yr = yt + ny * t
        zr = zt + nz * t

        # 计算投影处圆的法向量b
        vec_b = np.array([(xr - x0) / R0, (yr - y0) / R0, (zr - z0) / R0])
        # 垂直入射向量a
        vec_a = np.array([0, 0, -1])
        # 计算反射向量c
        u = 2 * (vec_a[0] * vec_b[0] + vec_a[1] * vec_b[1] + vec_a[2] * vec_b[2])
        vec_c = np.array([vec_a[0] - u * vec_b[0],
                          vec_a[1] - u * vec_b[1],
                          vec_a[2] - u * vec_b[2]])

        # 计算直线在馈源舱的交点(xe, ye, ze)
        xe = xr + (ze - zr) * vec_c[0] / vec_c[2]
        ye = yr + (ze - zr) * vec_c[1] / vec_c[2]
        recordPoints_x.append(xe)
        recordPoints_y.append(ye)
        # 检验是否命中馈源舱
        if xe**2 + ye**2 <= 0.5**2:
            success_times += 1

    return success_times, recordPoints_x, recordPoints_y

def q3solve(workingReflector, save_name, xlim=[-2,2], ylim=[-2,2], MC_times=10000,sig = 16):
    # 对于每一行进行10000次蒙特卡洛操作
    recordPoints_x = []
    recordPoints_y = []
```

```
    hitSignal = 0
    totSignal = MC_times * workingReflector.shape[0]
    for ind, rows in workingReflector.iterrows():
        ones_hitSignal, ones_recordPoints_x, ones_recordPoints_y = MonteCarlo_Reflector(
                        np.array(rows.node1),np.array(rows.node2),np.array(rows.node3),
                        rows.nx,rows.ny,rows.nz,
                        rows.x0,rows.y0,rows.z0,
                        MC_times)
        hitSignal += ones_hitSignal
        recordPoints_x.extend(ones_recordPoints_x)
        recordPoints_y.extend(ones_recordPoints_y)

    if save_name != 'off':
        plt.figure()
        img, extent = myplot(recordPoints_x, recordPoints_y, s=sig)
        plt.imshow(img, extent=extent, origin='lower', cmap=cm.jet)
        plt.xlim(xlim)
        plt.ylim(ylim)
        plt.xlabel('$x$')
        plt.ylabel('$y$')
        # 添加一个半径为0.5的圆
        circle = Circle((0, 0), radius=0.5, edgecolor='#FFD4F0', facecolor='none',
            linestyle='dashed', linewidth = 1.2)
        plt.gca().add_patch(circle)

        plt.savefig(save_name,dpi = 300)

    return hitSignal / totSignal

q3baseAns = q3solve(workingReflector, save_name = './imgs/q3基态热力图.png',xlim=[-5,5],
    ylim=[-5,5], MC_times=2000)
q3optimAns = q3solve(workingReflector_optimed, save_name = './imgs/q3优化后热力图.png',
    xlim=[-2,2], ylim=[-2,2],MC_times=2000)
print('基态: ', q3baseAns)
print('优化后: ', q3optimAns)


## 蒙特卡洛稳定性检验
valtimes = 50
q3MonteCarloTest = []
for i in range(valtimes):
    q3MonteCarloTest.append(q3optimAns - q3solve(workingReflector_optimed, save_name = 'off',
        MC_times=2000))
    print(f'第{i+1}次检验, 差值{q3MonteCarloTest[-1]}')

plt.figure(figsize=(8, 6))
plt.bar(np.arange(0, valtimes), q3MonteCarloTest, color='#00BDFF')
```

```
plt.xlabel('测试次数')
plt.ylabel('差值')
plt.grid(True)
plt.savefig('./imgs/q3蒙特卡洛检验.png',dpi=300)
```