基于多目标优化的 FAST 主动反射面形状调节模型

摘　要

FAST 射电望远镜为近似球面结构，用于接收和跟踪宇宙中的电磁信号。为使其在工作状态下拥有更大的观测角和更强的信号接收能力，FAST 采用主动反射面调节的技术，使其表面反射面板能在指定范围内通过促动器的径向伸缩调整至近似抛物面，利用抛物面的聚焦性质提高信号的接收水平和收集能力。本文建模分析了其主动反射面调节的变形策略和优化指标。

对问题一，要求待测天体在基准球面正上方时的理想抛物面模型，首先根据望远镜工作原理确定抛物面焦点位置，并由促动器伸缩范围得到顶点及焦距的可选范围，用**空间解析几何列式**求出带参数的抛物面系通解。根据题目数据和机械结构的变化范围作为限定条件建立**多目标优化模型**。构建图模型描述各节点间连接关系，用**广度优先搜索**的方法确定给定区域内可行的解，再对可行的抛物面系进行范围内数值解的优化条件分析，主要包括：用主索节点伸缩量的均值和方差指标来指示生成抛物面的整体平缓程度，用边缘曲率优化抛物面与球面的过渡处的平滑连接，用被调节节点数目衡量抛物面的有效反射面积。多目标优化后得到天体在正上方时的理想抛物面方程为：$x^2+y^2=561.6(z+300.6)$。

对问题二，对天体给定位置求出理想抛物面模型，考虑到位置角是特殊值，不关于坐标轴对称，可设目标抛物面为**空间曲面一般方程**进行求解。利用焦点与球心连线（即抛物面主光轴上的方向向量）和焦点坐标确定曲面方程中的参数，同样以顶点偏移量作为参数确定抛物面系的通解。利用问题一的目标优化模型确定理想抛物面的方程为：$(x+26.302)^2+(y+19.673)^2+(z+156.797)^2=(0.164x+0.123y+0.979z+441.78)^2$ 由促动器上下端点坐标和主索节点坐标可确定所有主索节点所在过球心的直线，利用**参数方程**将其与解得的抛物面方程联立，可以解得主索节点沿径向伸缩到抛物面上的伸缩量，用基准球面上主索节点的位置加上解得的促动器伸缩量就可以得到拟合理想抛物面的新主索节点坐标。

对问题三，计算馈源舱的接收比，考虑对用于拟合理想抛物面的所有三角形反射面板分别建模，用面板各顶点坐标和面板法向量求解反射信号函数表达式。将每个面板的反射信号函数与焦点处切平面方程联立，解出电磁波信号经面板反射在该平面上的覆盖区域，讨论其与馈源舱有效区域的关系可得到能将信号反射到馈源舱的反射面板数，与抛物面范围内的反射面板总数作比，得到**馈源舱的接收比**为78%。再用基准反射球面接收比进行分析，过程与抛物面的分析类似，对所有反射面板进行顶点的反射光路解析，得到能将信号反射到馈源舱区域的反射面板数为 148 块，**基准反射球面的接收比**为 3.44%。经过比较发现拟合抛物面对信号的反射效果好于基准球面，该主动反射面调节模型成立。

最后，对文中模型按附录中说明的形变偏差做分析，利用蒙特卡洛模拟检验，发现抛物面最优解的结果基本不受到影响，证明了模型具有一定的完备性和稳定性。对模型的优点、不足及改进方向做出了分析说明。

关键词：多目标优化　数学规划　图论-广度优先搜索　蒙特卡洛模拟

一、 问题重述

“FAST”500 米口径球面射电望远镜是中国建设的世界上单口径最大，灵敏度最高的射电望远镜，被称为“中国天眼”，在天文学，宇宙观测，脉冲星导航和引力波探测等学科领域具有重要作用和科学意义。

FAST 的结构主体上由球面形的主动信号反射面系统，以馈源舱为核心的信号接收系统和配套的控制与测量结构组成。其主动反射面主要包括主索网、下拉索及促动器和反射面板，可以在促动器径向伸缩−0.6~0.6米的范围上实现反射面板的构型调节，进而提升信号的接收比。

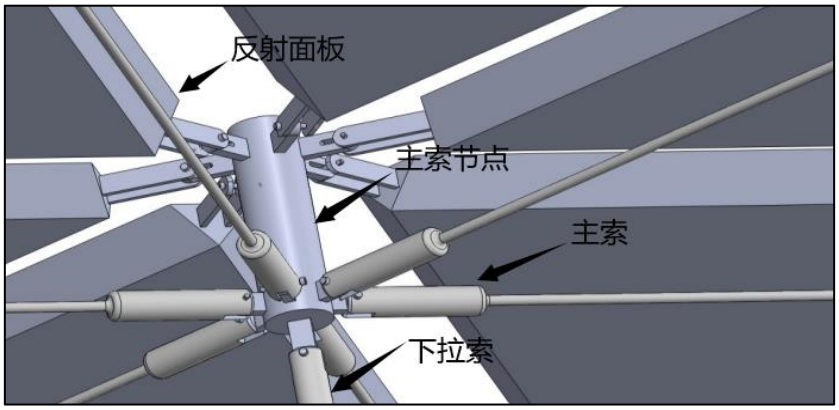

图 1　主索结点结构示意图[1]

在实际观测天体时，FAST 的馈源舱接收系统在其固定焦面上移动至天体和球心的连线延长线处，并以此作为目标理想抛物面的焦点，即工作抛物面的主光轴与天体在基准球面球心坐标系中的方向向量一致。通过整体调节一定区域内促动器的径向伸缩量，配合下拉索结构，使基准球面上范围内反射面板通过主动变形技术，由球面型的基准态转变为把面板形成有效照明口径为 300 米、焦比为 0.4665 的工作态近似旋转抛物面，利用抛物面将平行信号汇聚到焦点的性质提升聚焦效果，增加有效积分面积，还可以增加望远镜可以观测的俯仰范围，从而增加有效积分时间，以达到最佳的天体电磁波反射接收的目的。

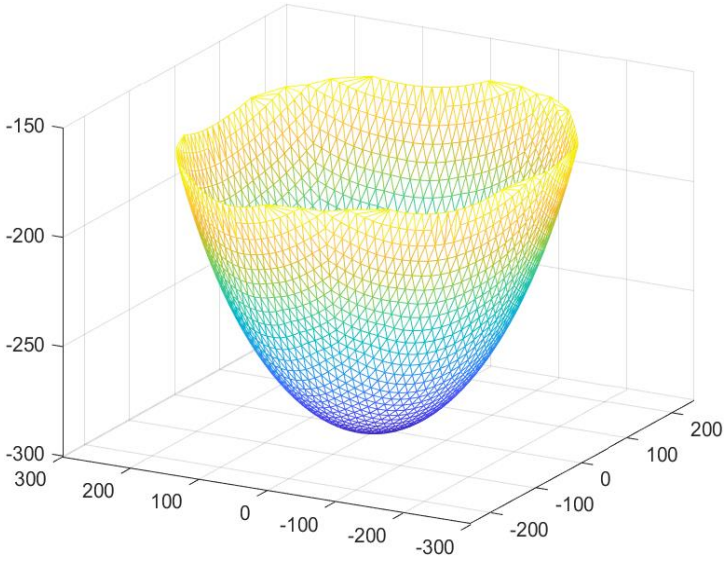

图 2　FAST 射电望远镜主索结构示意图

根据以上信息及附件中提供的 FAST 射电望远镜主索节点坐标及编号，促动器上下端点的空间坐标，通过数学建模的方法研究以下问题：

问题一：若有待观测天体位于 FAST 天文射电望远镜基准球面的正上方，即方位角$\alpha = 0°$，俯仰角$\beta = 90°$的位置，结合考虑反射面板的调节范围因素，确定理想抛物面。

问题二：若有待观测天体位于方位角$\alpha = 36.795°$，俯仰角$\beta = 78.169°$的位置，确定理想抛物面。并在此基础上建立反射面板调节模型，调节促动器的伸缩量，使反射面贴近理想抛物面，并记录调节后三百米口径内的主索节点编号、位置坐标、各促动器的伸缩量等结果。

问题三：在问题二所得反射面调节方案的基础上，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。

二、 问题分析

2.1 问题一分析

对于问题一，要求对给定位置的天体求出其所对应的理想抛物面，并需考虑反射面板调节的因素。给定的待观测天体位置位于 FAST 基准球面的正上方，则天体 S 与球心 C 的连线通过基准球面、馈源舱 P 所在焦面的中心位置，以馈源舱在焦面的中心位置作为目标抛物面的焦点，根据焦点与基准球面的坐标位置信息，可以通过解析几何的方法求出理想抛物面的方程。进一步地，考虑反射面板调节时，促动器在径向有伸缩范围，则所求抛物面的顶点可以在伸缩范围内变化，可得到含参数的抛物面系方程，其中描述的抛物面在竖直截面上的抛物线关系如下图所示。

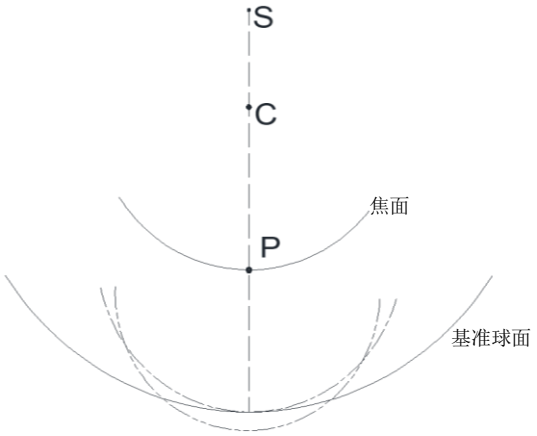

图 3　抛物面系示意图

为寻找最理想的抛物面解，需建立带约束的优化模型，其优化目标包括：

a) 在所有促动器的伸缩范围为±0.6 米的区间内保证反射面板构成的工作抛物面的照明区域为 300 米，保证所求抛物面可以由促动器径向牵引形成；
b) 主索节点的径向位移最小，在反射面板的机械运动过程中，调节量小可增加系统的响应速度和使用寿命，且使所成抛物面的整体趋势更为平缓；
c) 工作抛物面边缘与基准球面平滑过渡，射电望远镜整体在各主索节点处相互连接，在构成工作抛物面时使边缘平滑可减少对其他反射面板的影响[2]。
d) 工作抛物面的面积趋于范围内最大，提升反射面板面积可以收集更多信号，提

升射电望远镜的工作效率。

根据上述条件建立优化模型并寻找抛物面方程的最优解，即对应该位置工作抛物面的最理想模型。

2.2 问题二分析

对于问题二，需要在题目给定的天体方位下求出理想抛物面，并建立反射面板调节模型使其能尽量贴近该理想抛物面。由于问题二中给定的角度不适合用常规方法求解，在此考虑利用抛物面的性质“抛物面上的点到焦平面和焦点的距离相等”来求解，设抛物面为空间中曲面的一般方程，从天体的角度数据可以得到理想抛物面对称轴的方向向量和焦点坐标，由此可得到该曲面方程中的系数，再由附件中的促动器坐标信息可知各促动器对应下拉索的斜率及所在直线的参数方程，利用抛物面的性质可以解出工作抛物面范围内所有与下拉索延长线的交点坐标以及焦距。根据问题一中使用的最优化求解原则找到最佳的顶点位置即可确定抛物面方程中的所有参数，即为理想抛物面。

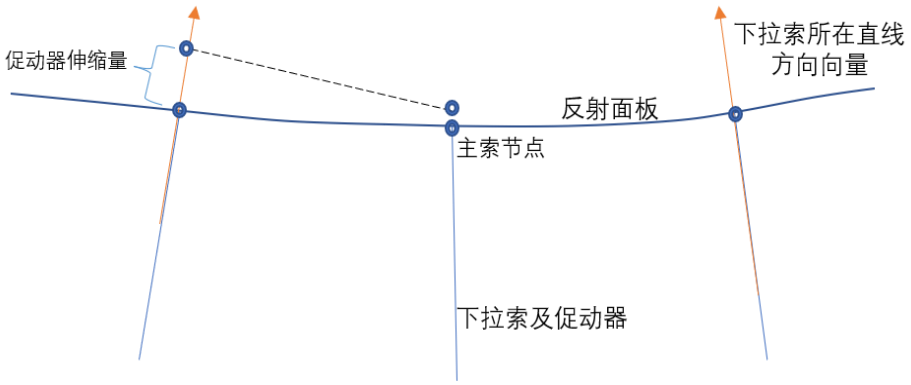

图 4　问题二结构示意图

求解反射面板调节模型，在需要调节的反射面板中，求已得理想抛物面与基准球面在主索节点延长线上交点的距离即为促动器的伸缩量信息。从已求得的主索节点变化后坐标位置利用空间中点的距离公式进行求解，即可得到各点的调节信息。

2.3 问题三分析

对于问题三，要求在问题二的面板调整基础上计算馈源舱的接收比，即工作抛物面区域内反射的信号与馈源舱一米直径范围内接收到的信号比值，并进一步比较其与基准反射球面的接收比。一般认为一个标准抛物面会将所有入射的平行信号反射至抛物面焦点的位置，对于 FAST 射电望远镜模型即为其馈源舱所在位置，但对 FAST 的结

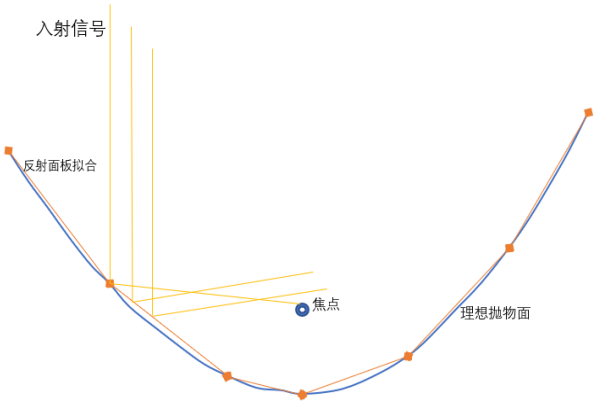

图 5　问题三结构示意图

构进行分析可知，其工作抛物面结构是用很多小型三角形的反射面板近似拟合而成，其反射信号不能完全依照理想抛物面的性质进行处理，所以需要对工作抛物面范围内的反射面板进行独立的分析，得到在问题二的拟合策略下馈源舱范围内的接收信号。

根据电磁波信号反射的原理，相当于无穷远处入射的信号可视作是平行入射，对每一块三角形反射面板做建模分析，求出其三个顶点处反射信号的传播情况，以进入抛物面区域的信号为全部，与馈源舱的一米直径中接收到的信号进行比较，得到实际面板反射信号到达接收范围的比例。再与基准反射球面在调节前的接收和有效反射信号比例进行比较，量化得到抛物面建模的效果以及控制主索节点进行拟合的有效程度。

三、 模型假设

1. 假设促动器的控制精度满足被控面板的移动要求。
2. 假设整个射电望远镜结构的强度能满足不受外界自然环境影响产生轻微形变。
3. 不考虑 FAST 整体的周边支撑结构连接的部分反射面板。
4. 假设反射面板载荷对主索节点的影响可以忽略不计。
5. 不考虑反射面板上的直径小于 5 毫米的小孔对结构的影响。
6. 反射面板的厚度和主索节点与反射面板顶点间的间隙忽略不计。
7. 假设面板的面积差距在反射信号时可忽略，认为各板对信号的反射能力相同。

四、 符号说明

符号	意义	单位
f	理想抛物面的焦距	m
h	理想抛物面顶点到基准球面的距离	m
R	基准球面的半径，$R=300$	m
F	焦面与基准面的距离，$F=0.466R$	m
D	抛物面在垂直于轴线的面上投影的口径直径，D=300	m
ΔL_{max}	促动器径向最大调节量，$\Delta L_{max}=\pm 0.6$	m
Δu_i	主索节点相对基准球面的径向偏移量，$i=1,2,\cdots,n$	m
ρ	抛物面在 300m 口径边缘处曲率	1/m
n_c	抛物面在口径范围内的反射面板数量	个
n_d	抛物面在口径范围内的主索节点数量	个

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型的建立

旋转抛物面在空间直角坐标系下有标准方程：

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=2z \tag{1}$$

其中a,b为任意的常数。

由问题一中的天体方位和射电望远镜工作原理，可知馈源舱位于焦面的中心位置，

取空间直角坐标系圆心为基准球面的球心，可确定目标抛物面焦点的坐标为$[0,0,-160.2]$。

调节反射面板的促动器装置径向伸缩范围为$\Delta L_{max}=\pm 0.6$米，则与抛物面顶点处对应主索节点可在$[-0.6,0.6]$的区间上变动。焦面与基准面的距离$F=0.466R$，故抛物面的焦距 f 可以在$[139.2,140.4]$的区间上变动。由以上条件可得出旋转抛物面的通式：

$$z=\frac{x^2+y^2}{4h+1.864R}-R-h \tag{2}$$

其中 h 为理想抛物面顶点到基准球面的距离。

根据分析，有以下几个条件进行约束和优化：

a) 调节反射面板形成工作抛物面时，在口径为 D 的照明区域内变动的各促动器其径向伸缩量Δu_i不能超过ΔL_{max}：

$$\Delta u_i \le \Delta L_{max}(D \le 300) \tag{3}$$

b) 所有做径向位移的主索节点移动量应尽可能小，使得工作抛物面的整体趋势更平缓：

$$min\left\{\sum_{i=1}^{n}\Delta u_i\right\} \tag{4}$$

c) 为使工作抛物面的边缘成型完整，且对未移动的基准球面面板影响尽可能小，要求其边缘与球面平滑过渡，因为抛物面与球面均具有各向同性和对称性，故要求其截面抛物线与圆的相交处的曲率 ρ 趋于最小：

$$min\{\rho\} \tag{5}$$

d) 在口径 300 米的范围内使反射面板数量n_c尽可能大，则对应的工作抛物面面积就更大，取得的反射信号效果更好，可简化考虑范围内的主索节点数目n_d，则：

$$max\{n_d\}$$

综合以上(2)，(3)，(4)，(5)式得到一个有约束的目标优化问题，优化模型[3]为：

$$find\ h=\frac{x^2+y^2}{4h+1.864R}-R-z$$

$$min\sum_{i=1}^{n}\Delta u_i,\rho;\quad max\ n_d$$

$$s.t.\ \Delta u_i \le \Delta L_{max}(D \le 300)$$

该模型的约束调节不适宜用解析的方法求出最优解，可通过数值分析的方法，对一定区间内的解做筛选和依指标处理，得出最优解的数值答案。

5.1.2 模型整体思路

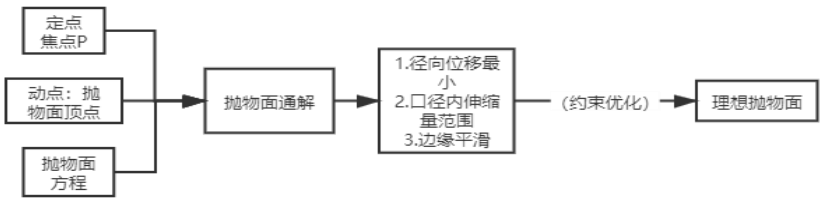

图 6　问题一模型思路

5.1.3 模型的求解

- 求解过程

通过解析几何方法解出含有参数的抛物面方程式后，即可根据约束条件和优化目标找出最理想模型的 h 参数。由于针对该参数的限定条件是由实际工程中的限定和分析进行总结，不适合在数学推导中求出解析解，故采用循环搜索算法对各数值解进行是否满足约束条件的分析，再寻找最满足优化条件的解。

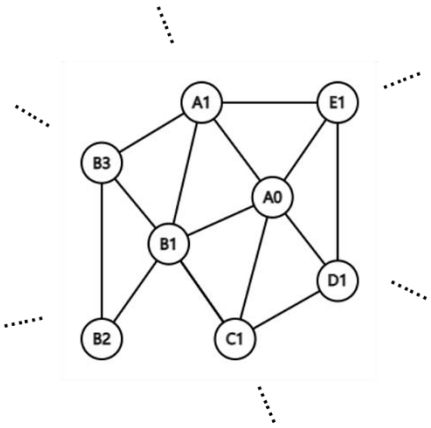

图 7　主索节点图模型示意图

根据附件三中提供的相邻主索节点之间关系的信息，可建立图模型将相邻主索节点的位置关系绑定，从目标抛物面顶点处对应的主索节点开始，使用广度优先搜索的方法对其周围节点进行约束条件的逐一验证，并向外拓展，直到对应抛物面方程得到的口径三百米范围内所有的主索节点其径向伸缩量都不超过促动器的径向伸缩范围，即该抛物面解是可行的。

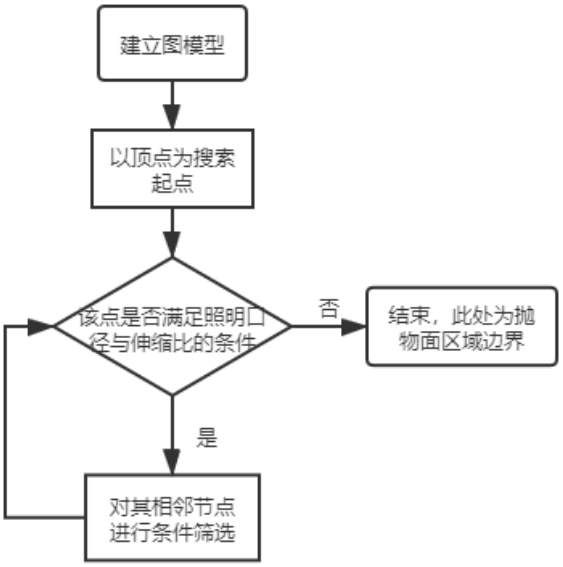

图 8　广度优先搜索算法流程图

在经过约束条件筛选后的可行解中，再对其主索节点径向伸缩量的平均值、方差和边缘处的曲率进行计算比较，选出其中的抛物面整体最为平缓，对主索节点的平均

移动量最低，边缘处最为光滑且有效面积最大的解[4]。

- 模型数据分析

对顶点距离基准球面上方[0,0.6]米的抛物面方程，经数据分析可知均不能满足构成三百米口径抛物面的条件，故不能作为有效解。

对顶点距离基准球面下方[0,0.4]米的抛物面方程，其部分求得抛物面的解的评价指标情况如下表：

表格 1　下方$[0,0.4]$内部分抛物面解情况

h	ρ	$\overline{\sum u_i}$
0	0.000103491	0.508
-0.08	0.000103441	0.4766
-0.24	0.000103343	0.4147
-0.32	0.000103294	0.3821
-0.4	0.000103245	0.3524

由于在口径三百米范围内仍没有足够数量的反射面板构成抛物面，且径向节点的平均伸缩量偏大，故不是最优的理想抛物面方程。

对顶点距离基准球面下方[0.4,0.6]米的抛物面方程，以步进值为 0.01 对其数据进行约束和优化的验证，其部分求得抛物面的解的评价指标情况如下表所示，完整数据见支撑材料：

表格 2　下方$[0.4,0.6]$内部分抛物面解情况

h	ρ	$\overline{\sum u_i}$	方差	n_d
-0.45	0.000103214	0.362	0.035	610
-0.46	0.000103208	0.3763	0.036	675
-0.47	0.000103202	0.3765	0.036	705
-0.48	0.000103196	0.3671	0.035	705
…	…	…	…	705
-0.58	0.000103135	0.2823	0.024	705
-0.59	0.000103129	0.2754	0.023	705
-0.6	0.000103123	0.2687	0.021	705

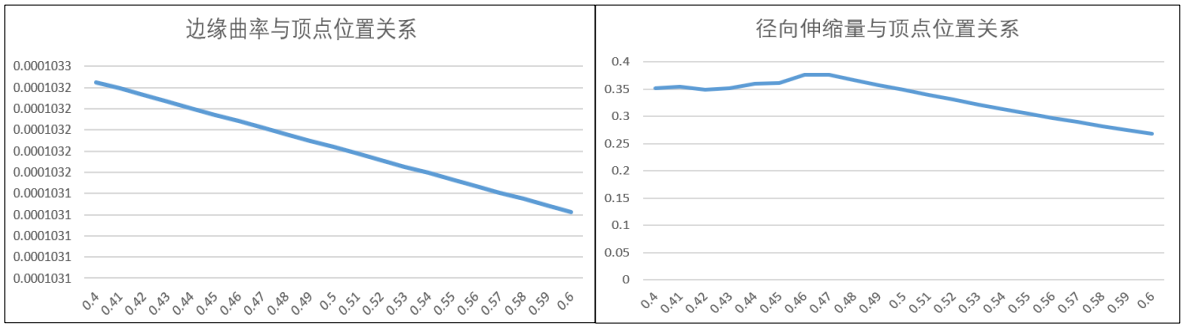

图 9　分析量变化趋势示意图

可以得出，数据约在$h=-0.47$处取得径向伸缩量和方差的极大值，表示在这附近产生的抛物面方程其平滑度最差，不适合作为理想方程，继续使抛物面顶点下降至可调范围的最大值 0.6 的过程中，解得抛物面方程的边缘曲率呈下降趋势，主索节点平均

伸缩量呈下降趋势，方差呈下降趋势。另外此时在照射口径内的主索节点数目达到极值，可认为其对应的抛物面面积最大，反射效果最好。综上，在$h=-6$的位置取得抛物面边缘处连接最平滑，促动器总伸缩量最小，且整体趋势最为平缓，有效面积最大的抛物面解，以此作为顶点产生的抛物面方程即认为是该天体位置下对应的最理想抛物面。

代入(2)式得抛物面的解析式如下：

$$x^2+y^2=561.6(z+300.6) \tag{6}$$

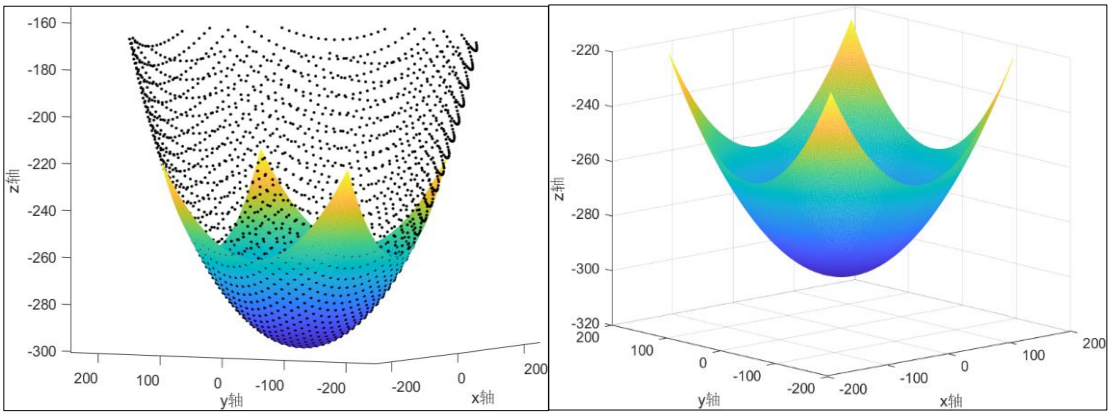

图 10　问题一理想抛物面示意图

5.2 问题二模型的建立与求解

5.2.1 模型的建立

根据问题二中的天体位置（$\alpha=36.795°$，$\beta=78.169°$），可解得过馈源舱所在位置（即焦点位置）且垂直于目标抛物面主光轴的平面方程为：

$$Ax+By+Cz+D_p=0 \tag{7}$$

$$\begin{cases} A=-cos\beta cos\alpha \\ B=-cos\beta sin\alpha \\ C=-sin\beta \end{cases}$$

其中 A，B，C 表示了该平面的方向向量，亦为过馈源舱和球心连线 CP 的直线（目标抛物面的主光轴）的方向向量，D_p 是平面在 p 点处的截距。

根据空间中曲面的性质，可建立目标抛物面的一般方程：

$$(x-x_p)^2+(y-y_p)^2+(z-z_p)^2=\frac{(Ax+By+Cz+D)^2}{A^2+B^2+C^2} \tag{8}$$

其中 A，B，C 与上述平面方程中的系数一致，$[x_p, y_p, z_p]$是焦点 P 的坐标。

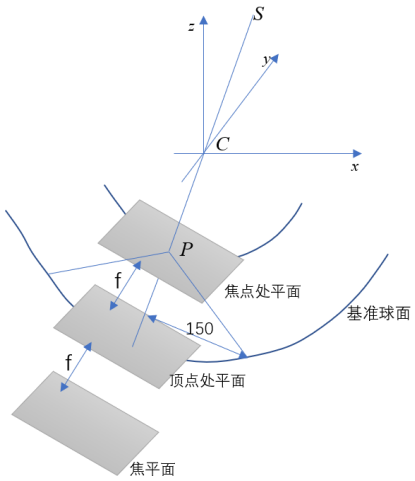

图 11　问题二模型示意图

由过焦点的平面方程可进一步建立目标抛物面的焦平面方程，以及过目标抛物线顶点，垂直于主光轴的平面方程，均与前式类似。

5.2.2 模型整体思路

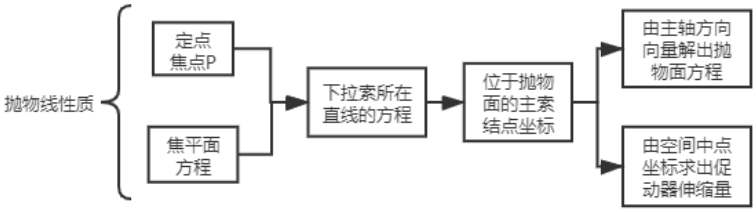

图 12　问题二模型思路

5.2.3 模型的求解

由于问题二中天体的角度对应的旋转抛物面不关于坐标轴对称，故采用空间内曲面的标准方程设解，根据主光轴方向向量和焦点坐标解出抛物面的通式，再根据下拉索延长线的坐标和方向向量给出各直线过圆心的参数方程，联立与空间曲面的交点，利用抛物面上的点距离焦平面和焦点的长度相等的性质，可解出所有在下拉索延长线过工作抛物面上的点的空间坐标。由此可进一步计算促动器伸缩量的均值和方差，曲率等数据。

对理想抛物面的优化方法与问题一中的一致，对顶点在其可变区间上的不同取值进行分析，用数值分析的方法找到最优解。主要考虑主索节点径向平均伸缩量和方差，主索节点数即有效面积，并满足在照明口径 D 上所有促动器在伸缩范围内。对数据处理后可知，类似问题一中的讨论，顶点在-0.4~0.6 范围内的解大部分不能满足要求，其余主要数据则根据优化目标选择，部分分析数据如下表所示，完整的分析数据见支撑材料：

表格 3　下方[0.47,0.6]内部分抛物面解情况

h	$\overline{\sum u_i}$	方差	n_d
-0.47	0.371814	0.036556	692
-0.48	0.363131	0.035498	692
-0.49	0.354264	0.034342	692
-0.50	0.34555	0.033155	692
…	…	…	
-0.58	0.281201	0.023527	692
-0.59	0.274015	0.022313	692
-0.6	0.26699	0.021139	692

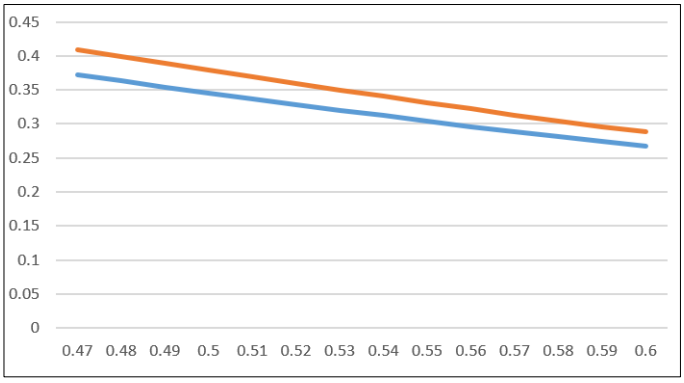

图 13　伸缩量均值和方差的变化趋势图

由数据和图线可得，h 向基准球面以下偏移至 0.47 时其各项指标达到最大值，之后到 0.6 之间的指标呈递减趋势，表明顶点偏下 0.6 米时所得的抛物面整体更平稳光滑，且边缘处连接光滑，认为是理想抛物面。

代入(8)式得抛物面的解析式如下：

$$(x+26.302)^2+(y+19.673)^2+(z+156.797)^2=(0.164x+0.123y+0.979z+441.78)^2$$

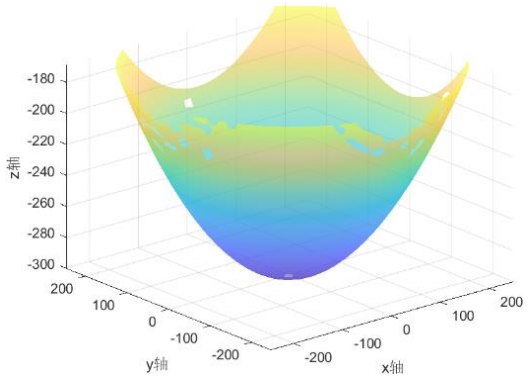

图 14　问题二理想抛物面示意图

调节相关促动器的伸缩量，使其尽量贴合抛物面。考虑将所有主索节点都移动至抛物面上，即在下拉索直线延长线上找到与理想抛物面的交点，通过将该坐标与基准球面上主索节点的原始坐标作差求解，设主索节点原坐标为$[x_1,y_1,z_1]$，对应在理想抛物面上的坐标为$[x_2,y_2,z_2]$，空间中两点距离公式$d=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}$可求得的即可得到所有相关促动器的伸缩量，以

趋向球面的方向为正。将最后解得的伸缩量，坐标等内容填入对应的文件中。

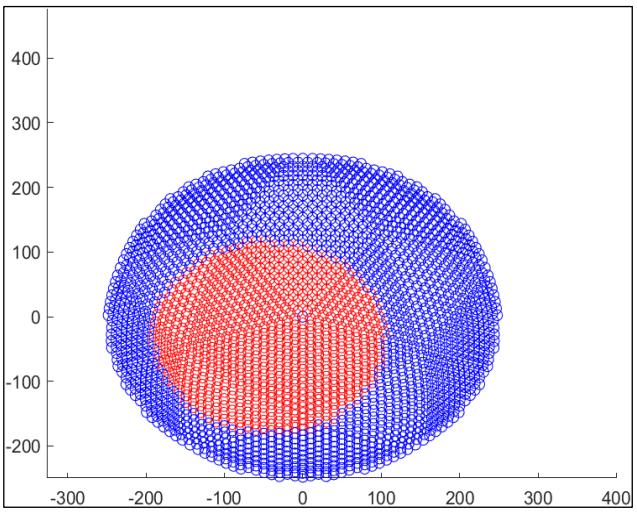

图 15　问题二中需伸缩的主索节点示意图

求出在该角度下有伸缩量的主索节点如上图所示。

表格 4　问题二理想抛物面的顶点坐标

X 坐标（米）	Y 坐标（米）	Z 坐标（米）
-49.352862	-36.913941	-294.21423

下表为部分主索节点求得的节点位置坐标和促动器伸缩量的结果，完整的坐标和伸缩量结果见支撑材料。

表格 5　问题二节点坐标和伸缩量的部分解

节点编号	X坐标(米)	Y坐标(米)	Z坐标(米)	伸缩量（米）
A0	0	0	-300.23109	0.16891413
B1	6.10250713	8.39972395	-299.96031	0.260044753
C1	9.8751741	-3.2085492	-299.99144	0.228896446
D1	0	-10.386964	-300.10362	0.116650182
E1	-9.880107	-3.2101556	-300.14138	0.078864396
A1	-6.104174	8.40201539	-300.04216	0.178149049
…	…	…	…	…

5.3 问题三模型的建立与求解

5.3.1 模型的建立

对问题三的反射信号建模，首先以工作抛物面范围内的反射面板三个顶点作为反射点，以三角形板的面法向量作为反射信号的法向量，可解得反射信号的空间函数表

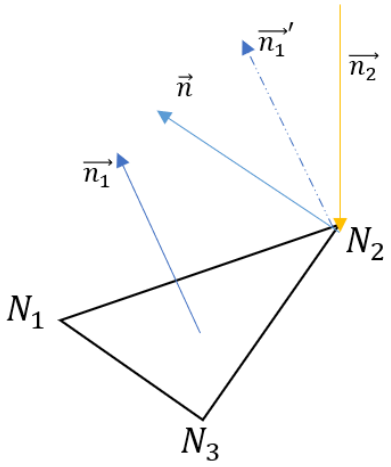

图 16　问题三模型示意图

达式。

设三角形板的法向量为$\overrightarrow{n_1} = (\alpha_1, \alpha_2, \alpha_3)$，入射信号的方向向量$\overrightarrow{n_2} = (\beta_1, \beta_2, \beta_3)$，由此确定反射信号的方向向量，设为$\overrightarrow{n_2} = (\gamma_1, \gamma_2, \gamma_3)$。

因为上述三个向量共面，且入射信号、反射信号与反射面板法向量的夹角相同，可得到：

$$\begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{vmatrix} = 0$$

$$\frac{\overrightarrow{n_1} \cdot \overrightarrow{n_2}}{|\overrightarrow{n_1}||\overrightarrow{n_2}|} + \frac{\overrightarrow{n_1} \cdot \vec{n}}{|\overrightarrow{n_1}||\vec{n}|} = 0$$

这样可以求解得到反射信号的空间解析表达式，对同一块反射面板的另外两个顶点同理。

再考虑馈源舱一米有效范围的表示，即在焦面上馈源舱所在位置的切平面上有一半径为 0.5 米，圆心为焦点的圆，反射信号在此圆内则视为被馈源舱有效接收，否则视为无效损失。焦点处切平面的一般方程可写为：

$$A(x - x_p) + B(y - y_p) + C(z - z_p) = 0$$

与反射信号表达式联立可解出在切平面上的交点。针对反射面板的三个顶点分别进行反射信号的函数模拟，则每个反射面板对应在切平面上是一个投影三角形，则将问题转化为平面上由反射面板映射的三角形与馈源舱有效区域组成的圆形位置关系的

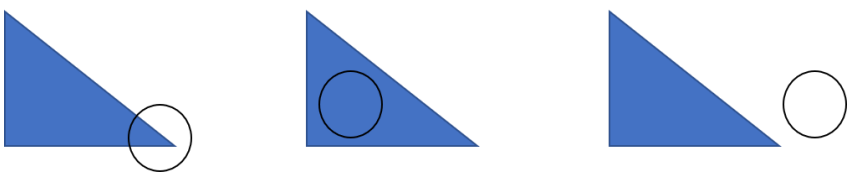

图 17　反射面板投影与馈源舱圆形区域位置关系

判别。为将问题离散化，同时满足信号在空间中的传播特征，可认为所有投影与圆形区域有交点的三角面板可以将信号反射至馈源舱内，没有交点就认为该反射面板的信号不能成功反射，求解可知道馈源舱的接收比。

如上图所示，靠左和居中的两幅图均认为信号反射成功，计算出未能成功反射信号的面板后再用面板的数量之比来计算馈源舱的接收比。

5.3.2 模型的求解

首先从主索节点的坐标中解出各反射面板的法向量，设信号入射的方程沿抛物面主光轴方向，在各个顶点处联立模型中的方程对反射光线的函数式求解。并与切平面方程解得反射面板在切平面上投影三角形的三个顶点坐标。用讨论三角形和圆形位置的思想，求解圆心与三角形的最短距离，判定其位置关系，统计成功反射信号到馈源舱的反射平面数量，与抛物面内的反射平面总数作比，即可求得馈源舱的接收比。采用第二问天体角度解得抛物面调节后馈源舱的接收比为78%。认为问题二模型的调整变形策略具有较高的拟合程度，得到的反射接收比较高。

再以基准球面的反射面板为准求解馈源舱的接收比，方法一致，经计算得到的反射成功的基准球面反射面板数目为 148 块，基准球面内的面板总数为 4300 块，馈源舱的接收比为 3.44%。可见球面的聚焦效果较差，所以要将其反射面板调整为理想抛物面形状，以提高信号的接收比。

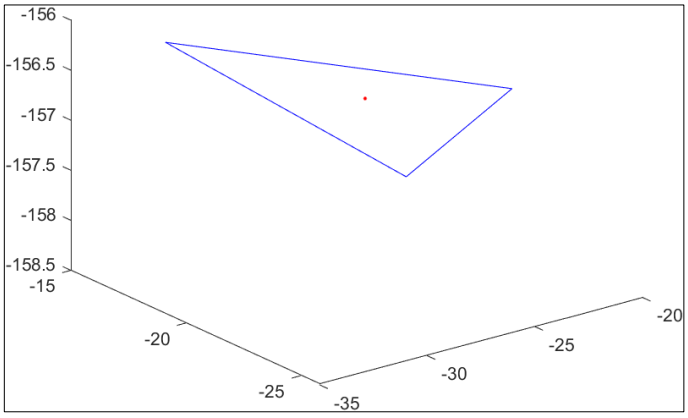

图 18　抛物面反射信号投影实例

六、 模型的分析与检验

6.1 模型完备性分析

对本文中使用到的解模型是否完备作分析，注意到题中提到：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，且变化幅度不超过 0.07%。这种微小变化可能是由于材料自身受牵拉后产生的轻微形变或结构设计中留有的机械余量导致的，考虑到每个主索节点都与相邻的六个节点通过反射面板相连，且在构成工作抛物面的主索节点调整过程中，与周围反射面板可能产生不同的相对角度，所以节点的空间位置会在 360 度的空间中产生变化。

为了便于分析，可忽略由万分之七的变化幅度在节点的径向、径向和纬向上的影响，假设其变化只集中在延反射面板边长的方向上。由于产生微小变化是可能性问题，故可通过计算机模拟这一变化的影响，考虑到变化幅度有最大值的上限，可以使用在±0.07%上正态分布的偏差量来对节点间的距离进行修正，即当某主索节点调节，其与

相邻节点的距离在原本的长度上乘以该正态分布的扰动量，即用蒙特卡洛方法模拟实际中可能的轻微形变，得到新的主索节点坐标。为验证模型的准确性和灵敏度，将考虑了扰动因素的数据代入前文模型求解最理想抛物面，并对结论作分析。

6.2 对模型添加扰动检验

当主索节点在径向移动时，其对相邻节点在沿反射面板各边方向有不超过 0.07%的扰动，即反射面板边长发生轻微变化。利用附件三同一面板对应主索节点的关系已建立将位置关系绑定的图模型，从目标抛物面的顶点处开始对其相邻边依次乘以随机扰动量处理，并在已变动的边长作标记。所有反射面板的边长都处理完毕后将新的主索节点坐标代入问题一的模型求解。以上思路的流程图如下：

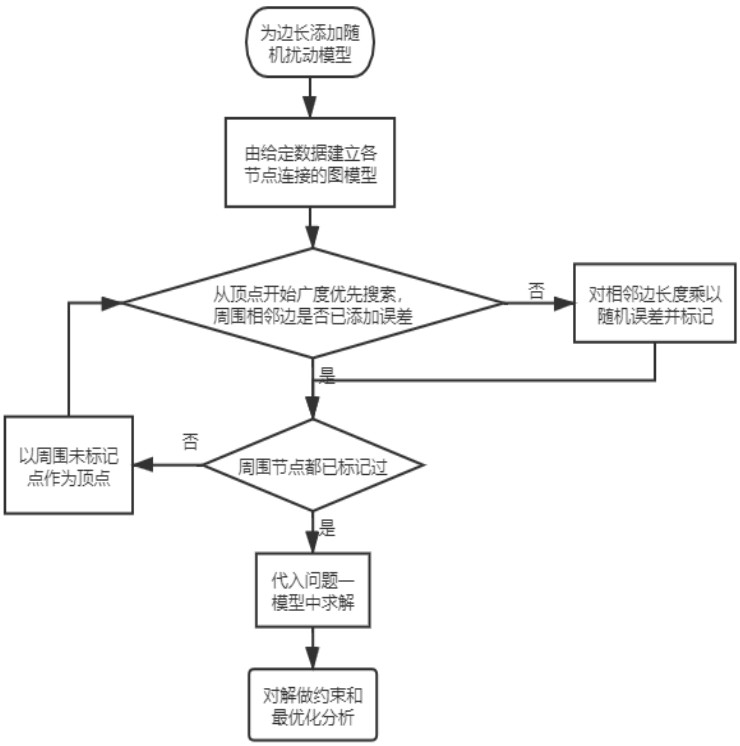

图 19　模型完备性检验流程图

经过添加扰动的主索节点模型在经过问题一模型验证发现，理想抛物面可行解的范围没有太大改变，优化指标的总体趋势在 h=-0.48 处取得最不理想的结果，之后各项指标逐渐上升，在 h=-0.6 时取得范围内最优的抛物线模型。模型解得的部分数据见下表：

表格 6　节点添加扰动后模型部分结果

h	主索节点平均径向伸缩量	n_d
-0.45	0.36204	610
-0.46	0.37632	675
-0.47	0.37646	705
-0.48	0.36708	705
…	…	…
-0.58	0.28234	705

-0.59	0.27538	705
-0.6	0.26865	705

6.3 检验结果评价

由此可得出，模型能在该微小形变扰动下得到正确结果，具有一定的完备性和稳定性。在实际工程中，若基准球面结构因为不可抗因素导致类似的干扰，本模型基本可以得到能够正确探测对应天体的旋转抛物面模型。

七、 模型评价与改进

7.1 模型的优点

a) 本模型考虑的优化指标较为完备，包括了整体的稳定性，边缘的过渡性以及有效区域最优等内容，对抛物面的解结果进行了较好的分析处理，得出理想抛物面解。

b) 建立与求解模型主要应用空间解析几何和数值分析求解的方法，增强模型的普适性和准确性。

7.2 模型的不足与改进

a) 问题二中的调整反射面板逼近理想抛物面，只考虑了对每个范围内的主索节点进行径向调整使其位于理想抛物面上，但由于反射面板是用平面三角形去拟合空间中的曲面，所以有出现在顶点处贴合而在面板其他部位偏差较大的情况，会影响第二位结果的拟合精度。这一问题应考虑应用动态面元拟合精度进行修正[5]。

b) 使用的算法模型不能兼顾处理数据的运行速度问题，在处理大量数据时需要耗费较长的时间，效率较低。这一问题需要进行编程思路的优化，降低处理复杂性。

八、 参考文献

[1]Rendong Nan and Di Li. The five-hundred-meter aperture spherical radio telescope (FAST) project[J]. IOP Conference Series: Materials Science and Engineering, 2013, 44(1)

[2]李明辉,朱丽春.FAST 瞬时抛物面变形策略优化分析[J].贵州大学学报(自然科学版),2012,29(06):24-28+43.

[3]杜敬利,段宝岩,保宏,訾斌.基于最小二乘方法的索网反射面形状精度调整[J].工程力学,2008(01):203-208.

[4]朱丽春.500 米口径球面射电望远镜(FAST)主动反射面整网变形控制[J].科研信息化技术与应用,2012,3(04):67-75.

[5]薛建兴,王启明,古学东,赵清,甘恒谦.500m 口径球面射电望远镜瞬时抛物面拟合精度的预估与改善[J].光学精密工程,2015,23(07):2051-2059.

附录

附录 1

介绍：支撑材料的文件列表

1.result.xlsx：按题目要求将问题二求解结果填入附件四表格中的文件

2.问题一二优化解的部分数据.xlsx：问题一、二中未在论文中完整列出的解各项指标的情况。

3.第 1 题最终运行结果.txt：问题一代码产生的全部结果

4.第 2 题最终运行结果.txt：问题二代码产生的全部结果

5.附录 2-8 的代码文件

附录 2

介绍：octave 代码用于求解问题一

```
    clear;clc;
data_p1=xlsread('附件 1');
R=300;
[r,c]=size(data_p1);
ans=[];
loc=-data_p1;
avg=[];
for h=0.4:0.01:0.6
    ans=[ans;[h,0,0,0]];
    count=0;
    sum=0;
   for i=1:r
       syms x y z;
       eq1=loc(i,2)*x-loc(i,1)*y;
       eq2=loc(i,3)*y-loc(i,2)*z;
       eq3=z+R+h-(x^2+y^2)/(4*h+1.864*R);
       [x,y,z]=solve(eq1,eq2,eq3,x,y,z);
       x=double(x);
       y=double(y);
       z=double(z);
       [r0,c0]=size(x);
       for j=1:r0
           flag=1;
           dis=sqrt((x(j,1)-data_p1(i,1))^2+(y(j,1)-
data_p1(i,2))^2+(z(j,1)-data_p1(i,3))^2);
           if sqrt(x(j,1)^2+y(j,1)^2)>150
               flag=0;
           endif
```

```
        if dis>0.6
            flag=0;
        endif
        if flag==1
            ans=[ans;[x(j,1),y(j,1),z(j,1),dis]];
            count=count+1;
            sum=sum+dis;
        endif
      endfor
    endfor
    ave0=sum/count;
    avg=[avg;[h,ave0]];

    endfor
```

附录 3

介绍：octave 代码用于求解广度优先搜索问题

```
    clc;clear;
global  len_edges all_edges
load('data_adj.mat')
node2idx = containers.Map(node_name,1:size(node_name,1));
adj = zeros(2226,2226);
for i=1:size(node_conn,1)
    temp1 = node2idx(node_conn(i,1));
    temp2 = node2idx(node_conn(i,2));
    temp3 = node2idx(node_conn(i,3));
    adj(temp1,temp2) = adj(temp1,temp2) + 1;
    adj(temp2,temp3) = adj(temp2,temp3) + 1;
    adj(temp1,temp3) = adj(temp1,temp3) + 1;
endfor
adj = (adj+adj')>0;
G = graph(adj);
all_edges = table2array(G.Edges);
rel_nodes = node_pos(all_edges(:,1),:) - node_pos(all_edges(:,2),:);
len_edges = sqrt(rel_nodes(:,1).^2 + rel_nodes(:,2).^2 +
rel_nodes(:,3).^2);
vflag=zeros(size(node_name,1),1);
for i=1:size(all_edges)
    if vflag(all_edges(i,2))==0
        a=-len_edges(i)*0.0007;
        b=len_edges(i)*0.0007;
        t=a+(b-a)*rand(1);
```

```
        A=node_pos(all_edges(i,1),1)-node_pos(all_edges(i,2),1);
        B=node_pos(all_edges(i,1),2)-node_pos(all_edges(i,2),2);
        C=node_pos(all_edges(i,1),3)-node_pos(all_edges(i,2),3);
        A1=A/sqrt(A^2+B^2+C^2);
        B1=B/sqrt(A^2+B^2+C^2);
        C1=C/sqrt(A^2+B^2+C^2);
      node_pos(all_edges(i,2),1)=node_pos(all_edges(i,2),1)+t*A1;
      node_pos(all_edges(i,2),2)=node_pos(all_edges(i,2),2)+t*B1;
      node_pos(all_edges(i,2),3)=node_pos(all_edges(i,2),3)+t*C1;
      vflag(all_edges(i,2))=1;
    endif
    endfor
```

附录 4

介绍：python 代码用于求解问题二

```
    import sympy
import code_1
import numpy

best_variation = -0.60
variance = float('inf')

alpha0 = 36.795
beta0 = 78.169
alpha = alpha0 / 180 * numpy.pi
beta = beta0 / 180 * numpy.pi
cp_length = (1 - 0.466) * 300
A = -numpy.cos(beta) * numpy.cos(alpha)
B = -numpy.cos(beta) * numpy.sin(alpha)
C = -numpy.sin(beta)
px = cp_length * A
py = cp_length * B
pz = cp_length * C
Dp = -(A * px + B * py + C * pz)

for i in range(244):
    variation = -0.60 + i / 200
    flag = 1
    temp_sum = 0
    ox = (300 - variation) * A
    oy = (300 - variation) * B
    oz = (300 - variation) * C
```

```
    Do = -(A * ox + B * oy + C * oz)
    D = 2 * Do - Dp

    offset_list = []
    temp_variance = 0

    for j in range(2226):
        name = code_1.node_namelist[j]
        node = code_1.main_cable_node[name]
        x = node.x
        y = node.y
        z = node.z
        direction = code_1.node_information[name].direction_1
        delta_x = px - x
        delta_y = py - y
        delta_z = pz - z
        distance =
numpy.linalg.norm(numpy.cross(numpy.array([delta_x, delta_y,
delta_z]), numpy.array([A, B, C])))
        if distance < 150:
            t = sympy.Symbol('t')
            X = x + t * direction[0]
            Y = y + t * direction[1]
            Z = z + t * direction[2]
            eq = [(X - px) ** 2 + (Y - py) ** 2 + (Z - pz) ** 2 - (A *
X + B * Y + C * Z + D) ** 2]
            ans = sympy.solve(eq, t)
            if len(ans) == 2:
                offset = min(abs(ans[0][0]), abs(ans[1][0]))
            else:
                offset = ans[t]

            if offset > 0.6:
                flag = 0
            else:
                offset_list.append(offset)

    temp_variance = numpy.var(offset_list)
    temp_sum = sum(offset_list)
    temp_average = numpy.mean(offset_list)
    print("variation = %f, temp_variance = %f, temp_sum = %f,
temp_average = %f, count = %d" % (
```

```
        variation, temp_variance, temp_sum, temp_average,
len(offset_list)))

    if flag and temp_variance < variance:
        best_variation = variation
        variance = temp_variance

print("best_variation = %f, variance = %f" % (best_variation,
variance))
    print("程序结束！")
```

附录 5

介绍：python 代码用于将问题二调整结果写入 result.xlsx 表格

```
    import openpyxl
import code_1
import numpy
import sympy

workbook = openpyxl.load_workbook(filename="附件 4.xlsx")
sheet_1 = workbook['理想抛物面顶点坐标']
sheet_2 = workbook['调整后主索节点编号及坐标']
sheet_3 = workbook['促动器顶端伸缩量']

variation = -0.60

alpha0 = 36.795
beta0 = 78.169
alpha = alpha0 / 180 * numpy.pi
beta = beta0 / 180 * numpy.pi
cp_length = (1 - 0.466) * 300
A = -numpy.cos(beta) * numpy.cos(alpha)
B = -numpy.cos(beta) * numpy.sin(alpha)
C = -numpy.sin(beta)
px = cp_length * A
py = cp_length * B
pz = cp_length * C
Dp = -(A * px + B * py + C * pz)
ox = (300 - variation) * A
oy = (300 - variation) * B
oz = (300 - variation) * C
Do = -(A * ox + B * oy + C * oz)
D = 2 * Do - Dp
```

```
# 填入理想抛物面的顶点坐标
sheet_1.cell(row=2, column=1).value = ox
sheet_1.cell(row=2, column=2).value = oy
sheet_1.cell(row=2, column=3).value = oz

move_node_name_list = []
offset_list = {}

for j in range(2226):
    name = code_1.node_namelist[j]
    node = code_1.main_cable_node[name]
    x = node.x
    y = node.y
    z = node.z
    direction = code_1.node_information[name].direction_1
    delta_x = px - x
    delta_y = py - y
    delta_z = pz - z
    distance = numpy.linalg.norm(numpy.cross(numpy.array([delta_x,
delta_y, delta_z]), numpy.array([A, B, C])))
    if distance < 150:
        t = sympy.Symbol('t')
        X = x + t * direction[0]
        Y = y + t * direction[1]
        Z = z + t * direction[2]
        eq = [(X - px) ** 2 + (Y - py) ** 2 + (Z - pz) ** 2 - (A * X
+ B * Y + C * Z + D) ** 2]
        ans = sympy.solve(eq, t)
        if len(ans) == 2:
            if abs(ans[0][0]) < abs(ans[1][0]):
                offset = ans[0][0]
            else:
                offset = ans[1][0]
        else:
            offset = ans[t]
        move_node_name_list.append(name)
        offset_list[name] = offset

line_num = 2
for k in move_node_name_list:
    node = code_1.main_cable_node[k]
```

```
    information = code_1.node_information[k]
    direction = information.direction_1
    x = node.x + direction[0] * offset_list[k]
    y = node.y + direction[1] * offset_list[k]
    z = node.z + direction[2] * offset_list[k]
    sheet_2['A%d' % line_num].value = k
    sheet_2['B%d' % line_num].value = float(x)
    sheet_2['C%d' % line_num].value = float(y)
    sheet_2['D%d' % line_num].value = float(z)
    sheet_3['A%d' % line_num].value = k
    sheet_3['B%d' % line_num].value = float(offset_list[k])
    line_num = line_num + 1

workbook.save(filename="附件 4.xlsx")
    workbook.close()
```

附录 6

介绍：octave 代码用于求解模型分析与检验部分

```
    clear;clc;
data_p2=xlsread('附件 1');
R=300;
[r,c]=size(data_p2);
ans=[];
loc=-data_p2;
avg=[];
for h=0.4:0.01:0.6
    disp(['h=',num2str(h)]);
    ans=[ans;[h,0,0,0]];
    count=0;
    sum=0;
   for i=1:r
       syms x y z;
       eq1=loc(i,2)*x-loc(i,1)*y;
       eq2=loc(i,3)*y-loc(i,2)*z;
       eq3=z+R+h-(x^2+y^2)/(4*h+1.864*R);
       [x,y,z]=solve(eq1,eq2,eq3,x,y,z);
       x=double(x);
       y=double(y);
       z=double(z);
       [r0,c0]=size(x);
       for j=1:r0
           flag=1;
```

```
        dis=sqrt((x(j,1)-data_p2(i,1))^2+(y(j,1)-
data_p2(i,2))^2+(z(j,1)-data_p2(i,3))^2);
        if sqrt(x(j,1)^2+y(j,1)^2)>150
            flag=0;
        endif
        if dis>0.6
            flag=0;
        endif
        if flag==1
            ans=[ans;[x(j,1),y(j,1),z(j,1),dis]];
            count=count+1;
            sum=sum+dis;
        endif
      endfor
    endfor
    ave0=sum/count;
    avg=[avg;[h,ave0]];

    endfor
```

附录 7

介绍：python 代码用于将附件 1，附件 2 的数据读入

```
    import openpyxl
import math

workbook_1 = openpyxl.load_workbook(filename="附件 1.xlsx")
workbook_2 = openpyxl.load_workbook(filename="附件 2.xlsx")

sheet_1 = workbook_1['附件 1']
sheet_2 = workbook_2['附件 2']


class Node(object):
    def __init__(self, out_x, out_y, out_z):
        self.x = out_x
        self.y = out_y
        self.z = out_z


class NodeInformation(object):
    def __init__(self, out_direction_1, out_direction_2,
out_pull_rope_length, out_length):
```

```
        self.direction_1 = out_direction_1
        self.direction_2 = out_direction_2
        self.pull_rope_length = out_pull_rope_length
        self.length = out_length


def calculate_distance(a1, a2):
    return math.sqrt((a1.x - a2.x) ** 2 + (a1.y - a2.y) ** 2 + (a1.z
- a2.z) ** 2)


def get_direction(a1, a2):
    k1 = a2.x - a1.x
    k2 = a2.y - a1.y
    k3 = a2.z - a1.z
    k = math.sqrt(k1 ** 2 + k2 ** 2 + k3 ** 2)
    k1 = k1 / k
    k2 = k2 / k
    k3 = k3 / k
    if k3 < 0:
        k1 = -k1
        k2 = -k2
        k3 = -k3
    return k1, k2, k3


main_cable_node = {}
pull_rope_low_node = {}
pull_rope_high_node = {}
node_information = {}
node_namelist = []

for i in range(2226):
    name = sheet_1['A%d' % (i + 2)].value
    x = sheet_1['B%d' % (i + 2)].value
    y = sheet_1['C%d' % (i + 2)].value
    z = sheet_1['D%d' % (i + 2)].value
    main_cable_node[name] = Node(x, y, z)
    node_namelist.append(name)

for i in range(2226):
    name = sheet_2['A%d' % (i + 2)].value
```

```
    low_x = sheet_2['B%d' % (i + 2)].value
    low_y = sheet_2['C%d' % (i + 2)].value
    low_z = sheet_2['D%d' % (i + 2)].value
    high_x = sheet_2['E%d' % (i + 2)].value
    high_y = sheet_2['F%d' % (i + 2)].value
    high_z = sheet_2['G%d' % (i + 2)].value
    pull_rope_low_node[name] = Node(low_x, low_y, low_z)
    pull_rope_high_node[name] = Node(high_x, high_y, high_z)

    pull_rope_length = calculate_distance(pull_rope_low_node[name],
pull_rope_high_node[name])
    length = calculate_distance(pull_rope_high_node[name],
main_cable_node[name])
    direction_1 = get_direction(pull_rope_low_node[name],
pull_rope_high_node[name])
    direction_2 = get_direction(main_cable_node[name],
pull_rope_high_node[name])
        node_information[name] = NodeInformation(direction_1,
direction_2, pull_rope_length, length)
```

附录 8

介绍：python 代码求解问题三

```
    import openpyxl
import code_1
import code_4
import numpy
import sympy

# 打开 excel 工作表'附件 3.xlsx'
workbook = openpyxl.load_workbook(filename='附件 3.xlsx')
sheet = workbook['附件 3']

# 计算初始顶点, alpha, beta, p 的数据
variation = -0.60
alpha0 = 36.795
beta0 = 78.169
alpha = alpha0 / 180 * numpy.pi
beta = beta0 / 180 * numpy.pi
cp_length = (1 - 0.466) * 300
A = -numpy.cos(beta) * numpy.cos(alpha)
B = -numpy.cos(beta) * numpy.sin(alpha)
C = -numpy.sin(beta)
```

```
px = cp_length * A
py = cp_length * B
pz = cp_length * C
p = [px, py, pz]


# 定义一个类：反射器
class Reflector(object):
    def __init__(self, first_node, second_node, third_node):
        self.first_node = first_node
        self.second_node = second_node
        self.third_node = third_node


reflectors_for_sphere = []
reflectors_for_paraboloid = []

# 读入附件 3 的数据
for i in range(4300):
    first = sheet['A%d' % (i + 2)].value
    second = sheet['B%d' % (i + 2)].value
    third = sheet['C%d' % (i + 2)].value
    reflectors_for_sphere.append(Reflector(first, second, third))

# 创建抛物面上的反射板构成的集合
for reflector in reflectors_for_sphere:
    r1 = reflector.first_node
    r2 = reflector.second_node
    r3 = reflector.third_node
    move_node_list = code_4.move_node_name_list
    if r1 in move_node_list and r2 in move_node_list and r3 in
move_node_list:
        reflectors_for_paraboloid.append(reflector)


# 定义函数：已知三个点坐标，求出平面法向量
def get_normal_vector(x1, y1, z1, x2, y2, z2, x3, y3, z3):
    a = (y1 - y2) * (z2 - z3) - (y2 - y3) * (z1 - z2)
    b = (z1 - z2) * (x2 - x3) - (x1 - x2) * (z2 - z3)
    c = (x1 - x2) * (y2 - y3) - (y1 - y2) * (x2 - x3)
    d = sympy.sqrt(a ** 2 + b ** 2 + c ** 2)
    a = a / d
```

```
    b = b / d
    c = c / d
    if c < 0:
        a = -a
        b = -b
        c = -c
    return [a, b, c]


# 定义函数：求两向量的点乘
def get_dot_product(n1, n2):
    return n1[0] * n2[0] + n1[1] * n2[1] + n1[2] * n2[2]


# 定义函数：求行列式的值
def get_determinant(n1, n2, n3):
    sum_1 = n1[0] * n2[1] * n3[2] + n1[1] * n2[2] * n3[0] + n1[2] *
n2[0] * n3[1]
    sum_2 = n1[2] * n2[1] * n3[0] + n1[1] * n2[0] * n3[2] + n1[0] *
n2[2] * n3[1]
    return sum_1 - sum_2


# 定义函数：求两点间距离
def get_distance(l1, l2):
    return sympy.sqrt((l1[0] - l2[0]) ** 2 + (l1[1] - l2[1]) ** 2 +
(l1[2] - l2[2]) ** 2)


# 定义函数：求向量模长
def get_modulus(n):
    return sympy.sqrt(n[0] * n[0] + n[1] * n[1] + n[2] * n[2])


# 定义函数：求直线与平面交点
def get_intersection(start, direction):
    g = sympy.Symbol('g')
    x = start[0] + direction[0] * g
    y = start[1] + direction[1] * g
    z = start[2] + direction[2] * g
    eq_1 = [A * (x - px) + B * (y - py) + C * (z - pz)]
    result = sympy.solve(eq_1, g)
```

```
    x = start[0] + direction[0] * result[g]
    y = start[1] + direction[1] * result[g]
    z = start[2] + direction[2] * result[g]
    return [x, y, z]


# 定义函数：求垂足
def get_drop_feet(point1, point2, point3):
    t0 = point2[0] - point1[0]
    t1 = point2[1] - point1[1]
    t2 = point2[2] - point1[2]
    s = sympy.Symbol('s')
    x = point1[0] + t0 * s
    y = point1[1] + t1 * s
    z = point1[2] + t2 * s
    eq = [(point3[0] - x) * t0 + (point3[1] - y) * t1 + (point3[2] -
z) * t2]
    ans = sympy.solve(eq, s)
    x = point1[0] + t0 * ans[s]
    y = point1[1] + t1 * ans[s]
    z = point1[2] + t2 * ans[s]
    return [x, y, z]


# 求基准反射球面的接收比
def get_rate_for_sphere():
    print("sphere:")
    total = len(reflectors_for_sphere)
    print("total = %d" % total)
    count = 0
    for r in reflectors_for_sphere:
        vertex_1 = code_1.main_cable_node[r.first_node]
        vertex_2 = code_1.main_cable_node[r.second_node]
        vertex_3 = code_1.main_cable_node[r.third_node]
        v1 = [vertex_1.x, vertex_1.y, vertex_1.z]
        v2 = [vertex_2.x, vertex_2.y, vertex_2.z]
        v3 = [vertex_3.x, vertex_3.y, vertex_3.z]
        n1 = get_normal_vector(vertex_1.x, vertex_1.y, vertex_1.z,
vertex_2.x, vertex_2.y, vertex_2.z, vertex_3.x,
                               vertex_3.y, vertex_3.z)
        n2 = [A, B, C]
        x0 = sympy.Symbol('x0')
```

```
        y0 = sympy.Symbol('y0')
        z0 = sympy.Symbol('z0')
        n = [x0, y0, z0]
        m1 = get_dot_product(n1, n2) / (get_modulus(n1) *
get_modulus(n2))
        m2 = get_dot_product(n1, n) / (get_modulus(n1) *
get_modulus(n))
        eq = [get_determinant(n1, n2, n), m1 + m2, x0 ** 2 + y0 ** 2
+ z0 ** 2 - 1]
        ans = sympy.solve(eq, [x0, y0, z0])
        if get_distance(n2, (-ans[0][0], -ans[0][1], -ans[0][2])) <
0.001:
            direction = ans[1]
        else:
            direction = ans[0]

        intersection_1 = get_intersection(v1, direction)
        intersection_2 = get_intersection(v2, direction)
        intersection_3 = get_intersection(v3, direction)
        drop_feet_1 = get_drop_feet(intersection_1, intersection_2,
p)
        drop_feet_2 = get_drop_feet(intersection_1, intersection_3,
p)
        drop_feet_3 = get_drop_feet(intersection_2, intersection_3,
p)
        distance_list = [get_distance(intersection_1, p),
get_distance(intersection_2, p),
                         (get_distance(intersection_3, p))]
        if (intersection_1[0] - drop_feet_1[0]) * (drop_feet_1[0] -
intersection_2[0]) > 0:
            distance_list.append(get_distance(drop_feet_1, p))
        if (intersection_1[0] - drop_feet_2[0]) * (drop_feet_2[0] -
intersection_3[0]) > 0:
            distance_list.append(get_distance(drop_feet_2, p))
        if (intersection_2[0] - drop_feet_3[0]) * (drop_feet_3[0] -
intersection_3[0]) > 0:
            distance_list.append(get_distance(drop_feet_3, p))
        min_distance = min(distance_list)

        if min_distance < 0.5:
            count = count + 1
    print(count / total)
```

```
# 求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
def get_rate_for_paraboloid():
    print("paraboloid:")
    total = len(reflectors_for_paraboloid)
    print("total = %d" % total)
    count = 0
    for r in reflectors_for_paraboloid:
        vertex_1 = code_1.main_cable_node[r.first_node]
        vertex_2 = code_1.main_cable_node[r.second_node]
        vertex_3 = code_1.main_cable_node[r.third_node]
        v1 = [vertex_1.x, vertex_1.y, vertex_1.z]
        v2 = [vertex_2.x, vertex_2.y, vertex_2.z]
        v3 = [vertex_3.x, vertex_3.y, vertex_3.z]
        n1 = get_normal_vector(vertex_1.x, vertex_1.y, vertex_1.z,
vertex_2.x, vertex_2.y, vertex_2.z, vertex_3.x,
                               vertex_3.y, vertex_3.z)
        n2 = [A, B, C]
        x0 = sympy.Symbol('x0')
        y0 = sympy.Symbol('y0')
        z0 = sympy.Symbol('z0')
        n = [x0, y0, z0]
        m1 = get_dot_product(n1, n2) / (get_modulus(n1) *
get_modulus(n2))
        m2 = get_dot_product(n1, n) / (get_modulus(n1) *
get_modulus(n))
        eq = [get_determinant(n1, n2, n), m1 + m2, x0 ** 2 + y0 ** 2
+ z0 ** 2 - 1]
        ans = sympy.solve(eq, [x0, y0, z0])
        if get_distance(n2, (-ans[0][0], -ans[0][1], -ans[0][2])) <
0.001:
            direction = ans[1]
        else:
            direction = ans[0]

        intersection_1 = get_intersection(v1, direction)
        intersection_2 = get_intersection(v2, direction)
        intersection_3 = get_intersection(v3, direction)
        drop_feet_1 = get_drop_feet(intersection_1, intersection_2,
p)
```

```
        drop_feet_2 = get_drop_feet(intersection_1, intersection_3,
p)
        drop_feet_3 = get_drop_feet(intersection_2, intersection_3,
p)
        distance_list = [get_distance(intersection_1, p),
get_distance(intersection_2, p),
                         (get_distance(intersection_3, p))]
        if (intersection_1[0] - drop_feet_1[0]) * (drop_feet_1[0] -
intersection_2[0]) > 0:
            distance_list.append(get_distance(drop_feet_1, p))
        if (intersection_1[0] - drop_feet_2[0]) * (drop_feet_2[0] -
intersection_3[0]) > 0:
            distance_list.append(get_distance(drop_feet_2, p))
        if (intersection_2[0] - drop_feet_3[0]) * (drop_feet_3[0] -
intersection_3[0]) > 0:
            distance_list.append(get_distance(drop_feet_3, p))
        min_distance = min(distance_list)

        if min_distance < 0.5:
            count = count + 1
    print(count / total)


get_rate_for_sphere()
    get_rate_for_paraboloid()
```