优化馈源舱信号吸收比的“FAST”主动反射面调节方案

摘要

主动反射面是“FAST”射电天文望远镜的关键装置，它由三维网索结构组成，每个网格单元为三角形的镜面，镜面形态由三角形顶点的主索节点控制。当天体处在不同的方位时，我们需要设计主动反射面的调节方案来获取高的馈源舱信号接受比。

对于给定的天体方位角，我们先设计一个理想抛物面。空间中的理想抛物面有三个自由度，我们通过分析开口方向的约束、焦点位置必定在馈源舱的处的约束、抛物面上任一点相对基准球面的径向偏移量有限的约束、抛物线最大口径为给定值的约束、主索节点间距离变化量有限的约束，最终将抛物面的自由度减小到一个，特征参数变化限定在很小的范围内，得到满足所有约束同时也满足馈源舱信号接受比的抛物面族。在这个过程中，我们采用了逐步增加约束条件减小参数空间的办法，在数值计算上将参数空间进行了离散化处理。我们最后还引入了促动器调节里程的成本函数来对其唯一的自由度进行优化，得到了最优化的抛物面方程。我们对该方程进行坐标变换即可得到任意天体方位角对应的理想抛物面。这种理想抛物面的馈源舱信号接收比为 100%.

对于工作抛物面形态的确定，我们为了兼顾满足所有约束以及馈源舱信号接受比最高这两个条件，一方面令工作抛物面上的所有主索节点都在抛物面上，以确保其满足约束条件，且约化工作抛物面的高维参数空间，另一方面采用释放无关约束的办法来增加自由度，以便于采用优化算法来对抛物面的形态进行调整。我们使用计算机程序，结合几何光学定律、坐标变换、矩阵分析、蒙特卡洛算法给出了馈源舱信号接受比的计算函数，以函数值作为优化性能指标来提供优化方向参考。这也是一个能计算任意方位天体以及任意形态反射面所对应的馈源舱信号接受比的函数。我们采用这种优化模型得出了本方案所能达到的最优工作抛物面，其接收比为 0.05238364729352367%，相对接收比为 0.035%。

我们最后基于我们的上面所用的方案进行了物理上的定性分析，提出了一种新的调节方案，即所有三角形镜面的重心都落在理想抛物面上，且镜面与理想抛物面相切。我们对这种方案的效率进行了初步验证，发现新方案确实可以大大提高馈源舱信号接受比，其相对接收比可达 37.48%。

关键字： 优化模型，蒙特卡洛算法，矩阵分析，近似分析，离散化处理

一、 问题重述

综述与目的 FAST 射电望远镜需要调节大型主动反射面的形状，使其形状接近接收信号性能优良的理想抛物面，以实现对宇宙深处的电磁波信号达到最大程度的吸收。大型主动反射面的主体结构包括反射电磁信号的 4300 个反射面板、将反射面板连结成大型主动反射面的主索网、主索网网格节点、与每个主索网节点耦合，实现对大型主动反射面调节的下拉索与促动器以及外围的支承结构。对大型主动反射面的调节主要是通过促动器对下拉索进行控制，进一步调节主索节点的位置实现的。

约束条件 在通过促动器调节主索节点高度时，存在下面的约束条件：
(1): 下拉索的伸缩距离有限，主索节点只能在其位于基准球面上的最初位置沿径向浮动 $-0.6m \quad 0.6m$
(2): 相邻主索节点的距离不能超过原先距离的 0.07%
(3): 在将大型主动反射面从基准状态调节到工作状态时，工作部分形状为口径 $300m$ 的近似抛物面，且要求焦点位于接收信号的馈源仓处

问题
(1): 在被测天体方位角为一特殊值:$\alpha = 0$，$\beta = 90$ 时，在上述约束条件下，确定此时调节目标——理想抛物面的相关参数
(2): 在被测天体方位角为一相对随机的值:$\alpha = 36.795$，$\beta = 78.169$ 时，基于上一问的经验与结论，计算实际调节时每一个主索节点的伸缩量，使得调节后大型主动反射面形状尽可能接近理想抛物面
(3): 根据上面问题的解答，计算在这一解答下，馈源仓的信号接收比，并于调节前做出比较

二、 问题分析

2.1 问题一的分析

这是一个给定约束条件，对三维空间中的旋转抛物面进行优化的问题。为了建立完备的模型，我们需要掌握三类情况：模型的自由度、模型的约束、模型的优化目标。下面我们逐一进行讨论。

图 1　问题一流程图

模型的优化目标　“FAST”作为一个科研装置，首要目标必然是保证科研产出效率，具体地说，就是要保证信号接受率尽可能地高。由旋转抛物面的性质可知，正对抛物面开口入射的平行光（即平行于抛物面对称轴入射的光）可以完美地汇聚到抛物面的焦点上。那么，在我们的理想抛物面模型中，这个优化目标可以量化为抛物面的焦点应当在馈源舱的圆盘上，且抛物面的开口应当正对入射光。除此之外，考虑到工程上的成本和装置疲惫性的问题，我们还可以把促动器的调节里程的绝对值之和最小作为优化目标之一。

模型的约束　天体在装置正上方时，为保证信号接受率的最优化，抛物线的开口方向必须正对天体，这是一个等式约束。且馈源舱处于抛物面的对称轴上，考虑到馈源舱的轨道限制，馈源舱的坐标是唯一确定的。我们的优化目标要求抛物面的焦点和馈源舱的坐标重合，这是一个等式约束。促动器的伸缩范围是有限的，即，抛物面的可行集为一个同心有限厚度球壳层，这是一个不等式约束。抛物面的口径为 300 m, 这要求抛物线的边缘落在直径为 300 m 的圆锥上，这是一个不等式约束。还有一个比较容易忽略的约束是，相邻主索节点的距离变化量是有限的。我们假设每个主索节点只有径向位移而节点

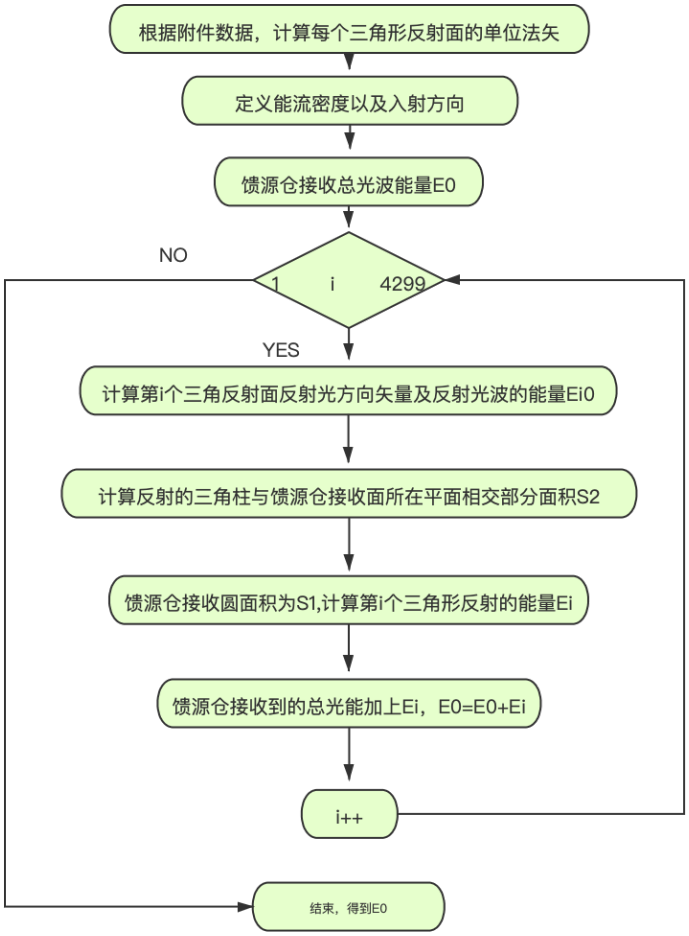

图 2　问题二流程图

间的角间距不变，这就要求相邻节点的径向位移对角向坐标的差分的绝对值有限。对应到连续的理想抛物面即为对偏导数的一个不等式约束。

模型的自由度　三维空间中的自由旋转抛物面显然有三个自由度，也即，三维座标架中旋转抛物面的方程由三个参数给定。由前述分析可知，该模型有两个等式约束和三个不等式约束，前者分别消去两个自由度，后者限定了自由参数的范围。因此，该模型的自由度只有一个，抛物面方程中只包含一个特征参数，且其范围由不等式约束给出。

我们可以在不等式约束限定的范围内搜索使模型最优的唯一特征参数。由于满足约束的理想抛物面总能完美聚焦在馈源舱所在点，我们接下来把促动器的调节里程的绝对值之和最小作为唯一的优化目标，即可确定抛物面的最优形态。

2.2 问题二的分析

理想抛物面与工作抛物面的关系 通过问题一的分析，我们可以得到理想抛物面。我们在求得理想抛物面的过程中，要求理想抛物面上的每一个点都要满足所有的约束，这就为我们后续找到工作抛物面提供了便利。也就是说，我们只需要令工作抛物面的所有主索节点都落在理想抛物面上，就可以得到满足所有约束的工作抛物面。然而，这样得到的工作抛物面并不一定可以是最接近我们的优化目标的。这是因为我们将连续的理想抛物面离散成工作抛物面后，光束不再能完美聚焦于馈源舱，可能会有部分光束不能射到馈源舱从而被浪费。

工作抛物面的优化目标 在工作抛物面无法理想聚光的情况下，我们必须首要考虑的优化目标就是使馈源舱的信号接受率达到最大。我们需要建立模型把不同形态的工作抛物面对应的馈源舱的信号接受率量化处理，以此来评价各个形态的优劣，以便于搜索到最佳的状态。我们考虑直接利用几何光学规律，即光沿直线传播以及不考虑损耗的反射定律，对于给定的光线入射方向和抛物面每个主索节点的位置，直接通过解析几何的数值计算得出射到馈源舱圆盘面的光束能量和入射抛物面的总光束能量之比。我们以此为优化性能指标，尝试找到该比值最大时的抛物面形态。

工作抛物面优化算法 通过前述分析我们知道，工作抛物面即要通过与理想抛物面的联系理想抛物面来保证满足约束，又要考虑以馈源舱信号接收率为优化性能指标去调节，我们需要尝试找到一个合理的优化算法来同时兼顾这两点。我们考虑对理想抛物面解放不必要的约束的办法，也即，我们在问题一的最后唯一确定理想抛物面的形态时，加入了工程成本方面的考量，这一约束将抛物面族的约束个数由 1 减为 0，但是对馈源舱信号接收率提高并没有帮助。因此我们在优化工作抛物面时这一约束暂时是不必要的。我们解放这一约束，这将给予我们的工作抛物面一个自由度，我们在这个自由度上调节工作抛物面，搜索得到馈源舱信号接收率最大的工作抛物面形态。

2.3 问题三的分析

问题三要求的后馈源舱的接收比我们在问题二的优化过程中将作为优化性能指标直接算出，该计算过程在数值计算上存在一定的技巧空间，但是思路上是简单的，此处不再重复分析。

问题流程图：如图 2和 1所示.

三、 模型的假设

- 主索节点只能沿径向位移。

- 每块镜面是刚性的、平整的，面积铺满以相邻的三个主索节点为顶点的三角形。
- 光沿直线传播，反射过程没有能量损失，反射率为 100%。

四、 符号说明

如表格 1所示。

五、 模型的建立与求解

5.1 模型的建立

5.1.1 开口竖直朝上的理想抛物面

先考虑简单的天体位于装置正上方的情形。

按照前述的两个等式约束，抛物面的开口竖直朝上，且焦点落在馈源舱轨道与抛物面对称轴的交点。由于理想抛物面是任意角度旋转对称的，我们不妨只需考虑抛物面的一个过对称轴的截面，所截部分为抛物线。下面我们的模型将对这条任意母抛物线分析，只要确定了抛物线的形态，整个抛物面也就自然随之确定。

我们建立以馈源舱所在点为坐标原点的空间直角坐标系，已知馈源仓为抛物线的焦点，故可得到抛物线方程为

$$y = \frac{1}{2p}x^2 - \frac{p}{2}. \tag{1}$$

以基准面球心为坐标原点，抛物线方程平移后为

$$y = \frac{1}{2p}x^2 - \frac{p}{2} - f, \tag{2}$$

其中 f 为馈源舱到球心的距离。

由此可见，该方程只有 p 一个参数，只要再确定抛物线上一点，就可以确定抛物线方程。建立如图 3所示的极坐标系，上述抛物线方程可以转换为

$$r = p \cdot \frac{-\cos\theta + \sqrt{1 + \frac{2f}{p}\sin^2\theta}}{\sin^2\theta}. \tag{3}$$

只要确定抛物线上任一点的坐标，我们就能解出参数 p，从而确定整条抛物线。我们考虑以抛物线边界点的位置来确定抛物线。由于抛物面的口径要求为 300 m，抛物线的边界点将落在距离对称轴 $d = 150\,m$ 的竖直线上。同时我们考虑到，每个促动器的伸缩量不超过 0.6 m，则连续化后，抛物线上的所有点包括边界点都将落在内径为 $R - 0.6\,m$，外径为 $R + 0.6\,m$ 的圆环内。我们取竖直线与圆环的交集，得到一有限长的线段，此即

表 1　模型中采用的符号的实际意义

符号	含义	数值	单位
R	基准面半径	300	m
d	抛物面口径值的一半	150	m
f	馈源仓到球心的距离	0.534R	m
i	主索点编号		
r_{i0}	基准面上主索点的初始径向坐标		m
θ_i	主索点的角向坐标		rad
η	相邻主索点距离变化的临界百分比	0.07%	
p	抛物线定型参数		
θ_1^*	半口径为 d 时的上临界角		rad
θ_2^*	半口径为 d 时的下临界角		rad
σ_i	入射光线照射到第 i 个反射镜面上的有效面积		m^2
S_i	第 i 个反射镜面面积，也用来表示这个三角面		m^2
S_i'	第 i 个反射镜面反射光线与馈源仓所在平面相交区域的面积，也用来表示这个三角面		m^2
S_0	馈源仓有效接收面积，也用来表示这个圆面	$\pi(\frac{1}{2})^2$	m^2
E_i	第 i 个反射面上单位时间接收到的光能		
E_{total}	馈源仓单位时间内接收到的光能		
E_0	300 单位时间内米口径反射面上接收到的光能		
$\hat{v_i}$	入射光线方向单位矢量		
$\hat{n_i}$	第 i 个反射镜面法向矢量		
$\hat{v_i'}$	第 i 个反射镜面反射光线方向的单位矢量		
λ	垂直于广播传播方向的单位面积上，单位时间内流过的光波的能量	100	
α	馈源仓接受比		

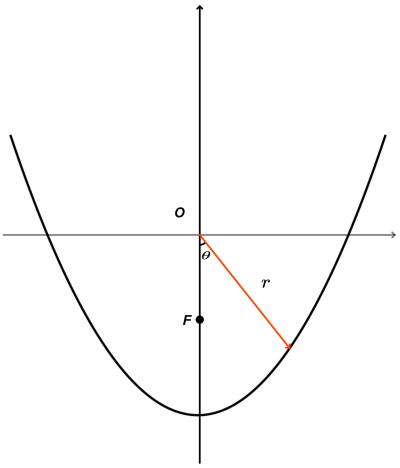

图 3　极坐标系和直角坐标系的对应关系示意图：以 y 轴负方向为极轴，逆时针为角向坐标的正方向。

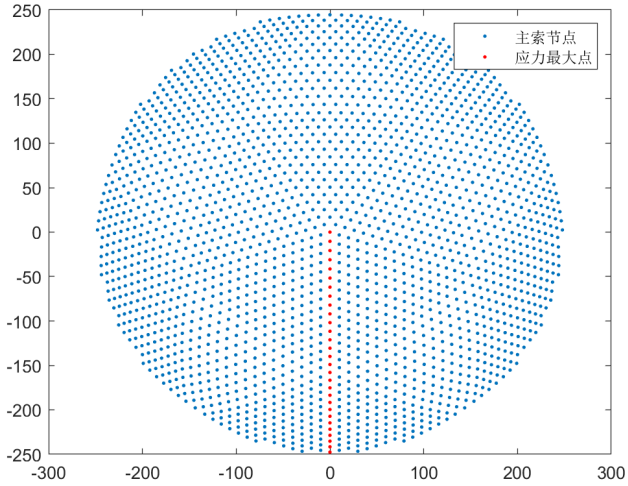

图 4　主索点应力分析图

抛物线的边界点的所有可能落点（图 5）。该线段的角向投影落在 $[\theta_1^*, \theta_2^*]$ 区间内，其中区间上下限的角向坐标由

$$\theta_1^* = \arcsin \frac{d}{R - 0.6} = 0.5248\ m, \tag{4}$$

$$\theta_2^* = \arcsin \frac{d}{R + 0.6} = 0.5224\ m \tag{5}$$

给出。接下来我们只考虑边界点的角向坐标 θ_0 的取值在区 $[\theta_1^*, \theta_2^*]$ 内，径向坐标与角向

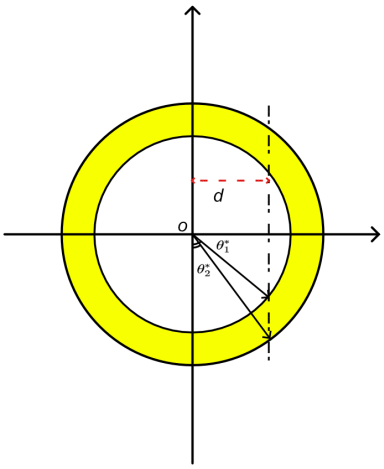

图 5　边界点范围的确定：图中竖直虚线与 y 轴的距离为 d，抛物线的边界点落在这条虚线上。黄色圆环为抛物线上所有点的可行域。圆环截虚线所得的线段即边界点的可行域。

坐标也是一一对应的，则可行的参数 p 也是一一对应的。我们将继续逐步加入约束条件来缩小参数 p 的可行范围。

我们刚刚只考虑了促动器的有限伸缩量对边界点的约束，现在我们把这个约束加在抛物线上的每一点。我们要求抛物线上的每一点都要落在图 5的黄色圆环内。又考虑到抛物线的连续性，出于简化数值计算的目的，我们可以对抛物线上的点进行离散化处理，只要这些离散点都落在可行域内，即这些点的径向坐标满足

$$R - 0.6 \leq r_i \leq R + 0.6, \tag{6}$$

我们就认为整条抛物线都处在可行域内。不难证明，当我们的均匀离散点数量足够多、足够密时，这样的考虑是合理的。这样一来，原来由参数 p 表出的抛物线族中的一些抛物线就会因为不符合约束而被剔除，我们通过这种逐步加强约束的方法来缩小参数范围，逼近最优的情况。

还有一个约束是容易被忽略的，那就是相邻主索节点之间的距离变化量不超过 0.07%。考虑到促动器只能沿径向移动，而由赛题 A 的附件 2 可知，下拉索的长度较短，且不能变形或伸缩，那么我们有理由假设在促动器的调节过程中，主索节点只能沿径向移动，也即，所有主索节点的角向坐标都是固定的。那么主索节点之间距离的变化量有限我们可以转化为不同主索节点径向位移的差值的绝对值有限。

由于实际的三角形单元边长 a 远远小于基准面球面半径，故考虑两个主索点间距

$R\Delta\theta = \Delta x$, 因此，相邻的主索节点之间的约束可以表示为

$$[(1+\eta^2)-1]x = \Delta l \tag{7}$$

故，

$$\sqrt{\eta^2+2\eta}R\Delta\theta = \Delta l \tag{8}$$

因此，

$$|\frac{\Delta l}{\Delta\theta}| = R\sqrt{\eta^2+2\eta} \tag{9}$$

$$\frac{r_{i+1}-r_i}{\theta_{i+1}-\theta_1} \leq R\sqrt{\eta^2+2\eta} \tag{10}$$

在我们考虑了以上所有约束后，抛物线族的可行域被我们限制到最小。由于抛物线的理想光学性质，在这个可行域内的所有抛物线都内完美聚焦于馈源舱，因此我们只考虑馈源舱的信号接受率为优化目标是不足以确定唯一的抛物面的，这个抛物线族确定的理想旋转抛物面族已经可以作为本问题的答案了。

但是我们不妨再考虑更多一些，出于工作成本和装置耐久度的考量，促动器的疲惫性常常也被看作主动反射面的性能指标 [3, 1, 2]。在保证馈源舱的信号接受率最高的情况下，我们可以尽量减少各个促动器的伸缩的总里程。对于连续的理想抛物面情形，这一指标可以解析化为抛物面与基准球面所夹的体积，又由对称性易知，这一体积正比于我们所研究截面的抛物线和基准圆弧所夹的面积。我们将这个面积 S 定义为成本函数

$$S = \int_{-\theta_0}^{\theta_0} |r(\theta) - R|\, R\mathrm{d}\theta, \tag{11}$$

我们可以在前面确定的最小可行域内搜索合适的抛物线，使成本函数 S 最小，即可确定唯一的参数 p，把我们的理想旋转抛物面唯一地确定下来。这一结果不仅可以保证科研产出效率的最大化，同时还可以保证工作成本的最小化。

5.1.2 利用理想抛物面确定工作抛物面

我们在求解理想抛物面的过程中，已经将所有的约束都加入到了我们的抛物面方程中，也即，我们的理想抛物面上的任意一点都满足所有的约束条件。这意味着，我们在考虑工作抛物面时，不需要逐一考虑数百个主索节点的自由度，和千余条棱边对系统的约束，我们只需要令每个主索节点都落在理想抛物面上，就可以让所有的主索节点都满足所有约束。我们已经假设所有主索节点只能沿径向移动，即每个主索节点的角向坐标都是不发生变化的。那么当我们给定一个理想抛物线方程时，可以由各个主索节点的角向坐标确定主索节点的径向坐标，工作抛物面也就完全确定了，即理想抛物面和工作抛物面时一一对应的。在这种情况下，工作抛物面的自由度等于理想抛物面的自由度，我们的模型由此可以得到大大简化，但又能不失严谨性地考虑到所有的约束。

但是，关于这种将工作抛物面和理想抛物面一一对应的方法我们还有一些问题需要讨论。注意到我们之前导出理想抛物面时我们将求解三维抛物面的问题等效为了求解二维截面上的抛物线的问题，对于问题一的情形，理想抛物面时任意角度旋转对称的，那么这种等效是完全合理的。但是对于任意角度旋转对称的抛物面而言，我们还没有讨论这种等效所得出来的数值的合理性。现在的问题在于，对于我们的工作抛物面情形，我们是否仍然可以证明：我们选取工作抛物面的某个截面，只要在这个截面上的抛物线满足约束条件（尤其是节点间距离变化量有限这一条件），就可以确保整个抛物面都满足约束条件？所幸前人的工作给予了我们理论上的支持，在钱宏亮先生的研究中 [3]，他指出“FAST”主动反射面现行的索网结构可以使应力在其中尽量均匀分布。但是我们在他给出的应力分布云图中依然可以看到，当工作抛物面均匀形变时，仍然有五条“棱边”是受力最大的，这意味这随着应变的增大，这五条“棱边”最先受到不等式约束的限制，而只要“棱边”上满足约束条件，其他的主索节点也都不会超出主索节点的限制。因此我们选取这条棱边所在的截面分析其约束，就仍可以等效于整个工作抛物面受到的约束。我们之前在理想抛物面中通过这种等效限制出的可行域依然可以应用在工作抛物面情形。

将工作抛物面和理想抛物面一一对应还存在另外一个问题，就是理想抛物面受的约束相对于工作抛物面而言过于严格。理想抛物面要求所有的点都必须满足约束条件，而工作抛物面只要求离散的点满足约束即可，这导致我们的理论可行域比实际可行域偏小，有可能会漏掉最优的情况，如何平衡约束强度和优化效果，是我们的优化算法需要考虑的问题。

5.1.3 工作抛物面的优化模型

我们的优化目标为馈源舱信号接收比最大。馈源舱信号接收比的算法我们在下一节中讨论。在此我们先明确优化思路。为了平衡约束强度和优化效果，我们考虑去除一部分不必要的约束，解锁一部分自由度，在这个自由度上做优化。

问题一中我们我们考虑了两个优化目标，一个是馈源舱信号接收比最大，一个是促动器调节里程的成本函数最低。前者将抛物面限制到了一个自由度内，后者在这个基础上将自由度减为 0，确定了唯一的最优抛物面。可见，我们可以解锁最后一个自由度，即我们的参数 p，它将在一个较窄的范围内变化，而不影响理想抛物面的馈源舱信号接收比。

开口竖直朝上的理想抛物线方程为

$$\rho' = p \cdot \frac{-\cos\phi' + \sqrt{1 + \frac{2f}{p}\sin^2\phi'}}{\sin^2\phi'}. \tag{12}$$

在非开口朝上情形，我们也可以通过坐标变换，在理想抛物面的“本征坐标系”中写出

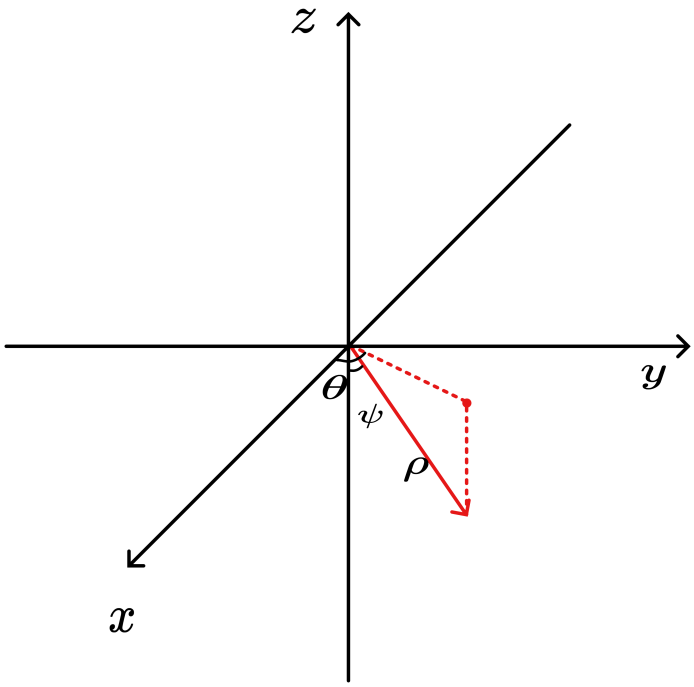

图 6　求解球坐标系

相同形式的方程, 我们以带撇的上标表示本征系中的坐标。当天体方位角给定 (α, β) 时, 理想抛物线顶点的坐标为

$$\rho_0 = -\frac{p}{2} - f \tag{13}$$

$$\phi_0 = \frac{\pi}{2} - \beta \tag{14}$$

$$\theta_0 = \pi + \alpha \tag{15}$$

由于主索节点只有径向位移而方位角不改变, 我们可以直接由原始数据求出各个主索节点的方位角集合 (α, β), 变换到理想抛物线的本征系中写为

$$\phi' = \phi - \phi_0 \tag{16}$$

$$\theta' = \theta - \theta_0 \tag{17}$$

我们将上式代入本征系的抛物面方程可以解得每个点的径向坐标的集合，记为 ρ'，再做坐标反变换得到 ρ。由此我们就通过天体方位角和唯一的参数 p 确定整个工作抛物面，需补充的是，参与形成工作抛物面可以由截基准球得口径为 d 的圆来筛定。

根据我们下一小节的建模，馈源舱信号接收比可以写为各点坐标集合以及天体方位角的函数

$$\eta_{eff} = \eta_{eff}(\alpha, \beta, \rho, \phi, \theta) \tag{18}$$

我们可以把函数中非独立的总量用参数 p 表出

$$\eta_{eff} = \eta_{eff}(\alpha, \beta, p) \tag{19}$$

我们在其他约束确定的范围内搜索 p，按馈源舱信号接收比的大小对其不同取值评以优劣，从而找到最优的参数 p。

5.1.4 馈源舱信号接收比的计算模型

我们采用原点在基准球面原点的空间直角座标架来描述主索节点在抛物面上的三角反射镜阵列。

根据第二个假设条件，建立如下的反射镜面模型：每三三相邻的主索节点确定一个平面三角形反射镜。光线在每一个平面反射镜表面的反射规律遵循平面镜反射规律。在单个三角形反射镜上入射的光和反射的光线都组成一个三棱柱。所有平面三角形反射镜组成反射镜阵列。

5.2 模型的求解

5.2.1 求解理想抛物面

我们逐步将之前建立的模型数值化处理。

由表达式（4)、(5）代入抛物线方程可以求出参数 p 的范围为

$$p \in [278.592314150362, 280.744474973557] \tag{20}$$

这样确定了一组由参数 p 描述的抛物线族，我们为了对其进行筛选，不妨对其离散取值，我们将边界点的角向坐标等间隔地取 50 个值，分别代入抛物线方程得出 50 个参数 p，也即 50 条抛物线，下面我们对其进行筛选。

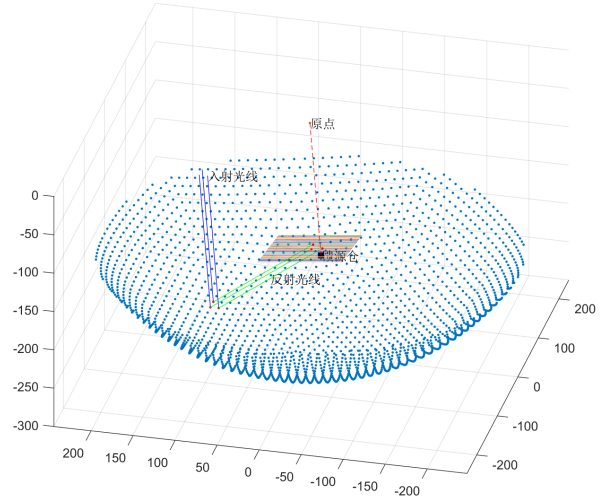

图 7　问题二光路图

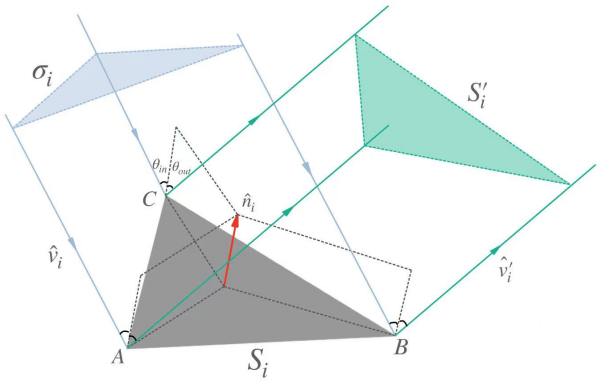

图 8　问题二光路图

我们再考虑约束（6）（10）剔除掉不符合约束条件的抛物线，得到满足所有约束的抛物线有 22 条，对应的最窄的参数 p 的范围为

$$p \in [279.819105058603, 280.744474973557] \tag{21}$$

这一结论我们在求解工作抛物面时也将要用到。

我们接下来考虑促动器的成本函数（11）最大来优化我们的参数。我们将上面得到的 22 个可行的参数逐一代入成本函数，选取函数值最低的参数 p，我们得到

$$p = 280.391417895406 \tag{22}$$

这种情况下的理想抛物面能同时满足科研效率和工程成本的优化要求。

此时我们可以算出抛物线顶点的直角坐标为

$$(0, 0, -300.395708947703) \tag{23}$$

实现以上计算的代码我们在附录中给出。

5.2.2 求解馈源仓接受比

根据接收比的定义

$$\alpha = \frac{E_{total}}{E_0} \tag{24}$$

需要计算馈源仓有效区域，即一个 1 米口径的圆面，上单位时间接收到的光能与口径 300 米的反射面单位时间接收到的光能的比值。

对于计算 E_0，不难得到下面的关系

$$E_0 = \lambda\pi(\frac{300}{2})^2 \tag{25}$$

对于计算 E_{total}，需要计算单位时间内每一个反射镜面反射到馈源仓有效面积上的能量，然后求和。由于每一个反射镜面反射的光柱与馈源仓所在平面的公共区域并不被馈源仓有效面积完全包括，所以每个反射镜面反射的光中，实际被馈源仓接受的光能与该反射镜面反射的总光能之比应该等于 S_i' 与 S_0 公共部分 $S_i' \cap S_0$ 与 S_i' 之比。

因此

$$E_{total} = \sum_{i=0}^{4299} \frac{S(S_i' \cap S_0)}{S_i'} E_i \tag{26}$$

对于每一个反射镜面单位时间反射的能量 E_i，不难得到下面的关系

$$E_i = \lambda\sigma_i \tag{27}$$

$$\sigma_i = S_i cos(\hat{v_i}, \hat{n_i}) \tag{28}$$

除了计算 S_i' 与 S_0 的公共面积以外，其它各量的计算是直截了当的。本小组使用“蒙特卡洛”方法计算 S_i' 与 S_0 的公共面积。具体实现方式如下。

与传统的蒙特卡洛方法的随机取点不同，本小组在 S_i' 内部均匀选取 N 个点，然后计算机遍历这些点，判断这些点是否又在 S_0 区域内部。满足条件的点总计有 m 个，那么便有如下关系近似成立。

$$\frac{S(S_i' \cap S_0)}{S_i'} = \frac{m}{N} \tag{29}$$

这样便提供了计算方法。

5.2.3 求最优工作抛物面

根据我们之前建立的模型，写成附录中对应的代码，我们解出的各主索节点坐标在附录中给出。

根据我们的算法,优化得到我们这种调节方案可以达到的最大信号接收比为0.00052383647293523

值得一提的是，上述接受比只是我们之前考虑的方案所能优化到的最大值。事实上，我们可以对原方案做一些改进，得到更优的方案如下：由于抛物面上的点是效率百分之百的点，因此，实际抛物面应该有尽可能多的点在理想抛物面上，尽量多的点在抛物面上即另三角形的中心与抛物面相切，此时该三角形有最多的点在抛物面上。我们基于此原则，可以得到最优的工作抛物面。我们对此进行了初步的计算，得到最大的馈源仓信号接收比为 0.00565948，相较于我们原定的方案提高了不少。但是需承认的是，这个值我们是在天体出于正上方的情况下算出的，我们没有足够的时间精力把新方案完善。

六、 模型的评价与推广

事实上，由于工作抛物面的三角形镜面面积较大，在反射过程中总有相当大部分的光会无法射中馈源舱。我们不妨估算一下该索网结构能够达到的相率的上限，即，我们用馈源舱面积除以单个镜面面积得到比值

$$\eta^* = \frac{S}{\pi r_0^2} \approx 0.0151 \tag{30}$$

这个值就是单个反射镜面最理想情况下能达到的信号有效反射比上限，对于我们的复杂多镜面情形，该上限只减不增。因此为了评价我们方案的效果，我们不妨确定一个相对接收比，即绝对接受比与接受比上限的比值。

6.1 灵敏性分析

我们对三角形重心贴合型抛物线通过改变参数 p 进行灵敏性分析。考虑最大的馈源仓信号接收比 $\eta^* \approx 0.0151$, 下面表格中展示的是相对接收比。

符号		意义	
p 取值	279.8	280.0	280.7
馈源仓信号接收比	0.0881	0.3748	0.2762

由此可见，p 参数取值为 280 时，馈源仓的信号接收比达到局部最优值。且当 p 的取值有较小的偏离，相对接受比的衰减是巨大的，这印证了这种新方案的合理性。

七、 总结

参考文献

[1] 丁磊, 郭正兴, and 罗斌. Fast 反射面支承结构的索网疲劳性能研究. In 第四届全国建筑结构技术交流会论文集 (上), 2013.

[2] 商文念. FAST 反射面支承结构优化研究. PhD thesis, 哈尔滨工业大学, 2007.

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

附录 A 求解理想抛物面–matlab 源程序

```
clc;
clear;

%% 通过边界确定p

dim = 50;
len = 1000;

R = 300;
F = 0.466*R;
CP = R - F;

theta = linspace(asin(150/(300-0.6)),asin(150/(300+0.6)),dim);
r       = 150 ./ sin(theta);

x0 = 150;
y0 = - r .* cos(theta);

p = -(y0+CP) + ( (y0+CP).^2 + x0^2 ).^0.5;

%% 抛物线中的所有点都在可行球层内

theta0 = linspace(asin(150/(300-0.6)),asin(150/(300+0.6)),dim);
dim=length(theta0);

f = CP;
R = 300;

theta = zeros(dim,len);
r       = zeros(dim,len);
for i = 1:dim
    theta(i,:) = linspace(-theta0(i), theta0(i), len);
%    ct          = cot(theta(i,:));
%    r(i,:)       = p(i) .* ( ct.^2 + 1 ).^0.5 .* ( ct + ( ct.^2 + 1 + 2*f/p(i) ).^0.5 ) ;
    r(i,:) = p(i) * ( -cos(theta(i,:)) + ( 1 + 2*f/p(i) .* sin(theta(i,:)).^2 ).^0.5 ) ./
        sin(theta(i,:)).^2 ;
end

er = abs( r - R );
max_er = zeros(dim,1);
for i = 1:dim
    max_er(i) = max( er(i,:) );
    if max_er(i) > 0.6
        p(i) = 0;
```

```
        r(i,:) = 0;
        theta(i,:) = 0;
    end
end

p(find(p==0))=[];
r(all(r==0,2),:)=[];
theta(all(theta==0,2),:)=[];

%% 抛物线的斜率约束

dim = length(p);

dr = diff(r')';
dt = diff(theta')';
partial=dr./dt;

eta = 0.0007;

max_p = zeros(dim,1);
for i = 1:dim
    max_p(i) = abs( max( partial(i,:) ) );
    if max_p(i) > R*sqrt(eta^2+2*eta)
        p(i) = 0;
        r(i,:) = 0;
        theta(i,:) = 0;
    end
end

p(find(p==0))=[];
r(all(r==0,2),:)=[];
theta(all(theta==0,2),:)=[];
dim = length(p);

%% 利用成本函数作优化

cost = zeros(dim, 1);
ds = zeros(dim,len-1);
for i = 1:dim
    for j = 1:len-1
        ds(i,j) = abs( (r(i,j)+r(i,j+1))/2 - R) * R .* dt(i,j);
    end
    cost(i) = sum(ds(i,:));
end

good_p = p(find(cost==min(cost)))
```

```
o = -good_p/2 - f
```

求解三角形中心贴合抛物面的理想抛物面–matlab 源程序

```
  clc;clear;
load("data.mat");
load("tri.mat");
load("musk.mat");
load("p.mat");
filename = ["1.mat", "2.mat", "3.mat", "4.mat", "5.mat", "6.csv", "7.csv", "8.csv", "9.csv",
    "10.csv", "11.csv", "12.csv", "13.csv", "14.csv", "15.csv", "16.csv", "17.csv", "18.csv",
    "19.csv", "20.csv", "21.csv", "22.mat" ];
%%有4300个三角形
%%p个coord
%%定义一个length(p)*3*3*length(tri)的
big=zeros(3,3,length(tri),length(p));
%%对每个三角形可以确定一个初始x1,y1,z1;x2,y2,z2;x3,y3,z3
mask=zeros(length(musk),3);%%x,y,z坐标累加
musk=string(musk);
tri=string(tri);
%坐标矩阵
coord=zeros(3,3,length(tri));%第i个三角形的每一行是一个点的坐标
%%
rt=15;
p=p(rt);

for i=1:length(tri)%%对每个三角形进行寻找，找到一个坐标矩阵
%%寻找tri中i行j列元素的索引，并找到其在musk中行，即data中的行
  for j=1:3
%       fun = @(x)strcmp(tri(i,j),char(x));
%       result = cellfun(fun,musk,'un',0);
%       [row,col] = find(~cellfun(@isempty,result));
%       coord(j,:,i)=data(row,:);
        row=find(musk==tri(i,j));
        coord(j,:,i)=data(row,:);
  end

end

%%确定每个三角形三个顶点最后的坐标
coord_new=zeros(3,3,length(tri));
%%每个三角形三个顶点初始的坐标
x_i0=zeros(1,3);
y_i0=zeros(1,3);
z_i0=zeros(1,3);

for k=1:length(tri)
   x_i0=coord(:,1,k);
   y_i0=coord(:,2,k);
```

```
    z_i0=coord(:,3,k);
    %%初始重心坐标
    x_10=mean(x_i0);
    y_10=mean(y_i0);
    z_10=mean(z_i0);
    %%%函数参数法向量为任意两条切线的方向向量的叉乘
    f=300*(1-0.466);
    %p=p(w);
    %%根据角度求解后来的重心坐标x_0,y_0,z_0
    syms t;
    eqn = (( x_10 * t )^2+( y_10 * t )^2)/(2*p) -p/2-f == ( z_10 * t );
    t_0 = max(double(solve(eqn, t)));
    x_0 = x_10 * t_0;
    y_0 = y_10 * t_0;
    z_0 = z_10 * t_0;
    %%确定抛物面在x_0,y_0,z_0点的法向量
    %%法向量为任意两条切线的方向向量的叉乘
     %%xy平面内的圆切线
    n1=[-y_0,x_0,0];
    %%zx平面
    theta=atan(y_0/x_0);
    A=[cos(theta),-sin(theta),0;sin(theta),cos(theta),0;0,0,1];
    u=[p,0,x_0];
    n2=A*u';
    %%确定法向量
    n=cross(n1,n2);
    %平面的方程
     t_i = zeros(3,1);
     x_i = x_i0;
     y_i = y_i0;
     z_i = z_i0;
     for i = 1:3
         syms t
         eqn = n(1)*(x_i0(i) * t-x_0)+n(2)*(y_i0(i) * t-y_0)+n(3)*(z_i0(i) * t-z_0) == 0;
         qq = max(double(solve(eqn, t)));
         t_i(i)=qq;
         x_i(i) = t_i(i) * x_i0(i);
         y_i(i) = t_i(i) * y_i0(i);
         z_i(i) = t_i(i) * z_i0(i);

     end
%      for i = 1:3
%        syms t
%        eqn = n(1)*(x_i0(i)/(1-t)-x_0)+n(2)*(y_i0(i) /(1-t)-y_0)+n(3)*(z_i0(i) /(1-t)-z_0) ==
    0;
%        qq = max(double(solve(eqn, t)));
%        t_i(i)=qq;
```

```
%         x_i(i) = t_i(i) * x_i0(i);
%         y_i(i) = t_i(i) * y_i0(i);
%         z_i(i) = t_i(i) * z_i0(i);
%
%     end


    coord_new(:,1,k)=x_i;
    coord_new(:,2,k)=y_i;
    coord_new(:,3,k)=z_i;

end
%save(filename(i),"coord_new");
%%由于每个点会有重复的，我们得找到每个点的所有坐标索引
%%点的个数为length（musk)

mask=zeros(length(musk),4);%%x,y,z坐标累加（sumx,sumy,sumz,sum)
for i=1:length(musk)
   %%遍历tri寻找musk(i),再将坐标值相加
   for j=1:size(tri,1)
       for k=1:size(tri,2)
%           [row,len]=find(tri(j,k)==musk(i));
            if tri(j,k)==musk(i)
               mask(i,4)=mask(i,4)+1;%%记录一下有多少个
               mask(i,1)=mask(i,1)+coord_new(k,1,j);%%3,3怎么存进去的，第j个三角形，k表示第几个点，j123表示x/y/z
               mask(i,2)=mask(i,2)+coord_new(k,2,j);
               mask(i,3)=mask(i,3)+coord_new(k,3,j);
            end
       end
   end

end
%%平均一下
mask_aver=mask(:,1:3)./mask(:,4);
%save(filename(rt),"mask_aver")
```

附录 B 计算馈源舱接受比程序–python 源代码

```
import numpy as np
import csv


def det(A):
   return float(A[0,0])*float(A[1,1]) - float(A[0,1])*float(A[1,0])
```

```
def area(A, B, C):
    x_ = det(np.array([[float(B[1]) - float(A[1]), float(B[2]) - float(A[2])],
                       [float(C[1]) - float(A[1]), float(C[2]) - float(A[2])]]))
    y_ = -det(np.array([[float(B[0]) - float(A[0]), float(B[2]) - float(A[2])],
                        [float(C[0]) - float(A[0]), float(C[2]) - float(A[2])]]))
    z_ = det(np.array([[float(B[0]) - float(A[0]), float(B[1]) - float(A[1])],
                       [float(C[0]) - float(A[0]), float(C[1]) - float(A[1])]]))
    return 0.5 * np.sqrt((x_) ** 2 + (y_) ** 2 + (z_) ** 2)


def dot_p(a, b):
    s = 0
    for i in range(len(a)):
        s += a[i]*b[i]
    return s


def norm(a):
    s = 0
    for i in range(len(a)):
        s += a[i]**2
    return np.sqrt(s)


def optimizer(file_name, alpha, beta):

    # 读取数据
    data1 = csv.reader(open(file_name))
    data_coor = np.array([0, 0, 0])
    for line in data1:
        data_coor = np.vstack((data_coor, line))
    data_coor = data_coor[1:, :]

    neib_p = csv.reader(open('./pares.csv'))
    data_neib = np.array([0, 0, 0])
    for line in neib_p:
        data_neib = np.vstack((data_neib, line))
    data_neib = data_neib[2:, :]

    name = csv.reader(open('./ori.csv'))
    names = np.array([0, 0, 0, 0])
    for line in name:
        names = np.vstack((names, line))
    names = names[2:, 0]
```

```
data_coor_d = {}
i = 0
for ele in data_coor:
    data_coor_d[names[i]] = ele
    i += 1

# 计算法矢
data_normal = {}
j = 0
for tri in data_neib:
    A = tri[0]
    B = tri[1]
    C = tri[2]
    A_coor = data_coor_d[A]
    B_coor = data_coor_d[B]
    C_coor = data_coor_d[C]
    n_x = det(np.array([[float(C_coor[1]) - float(A_coor[1]), float(C_coor[2]) -
        float(A_coor[2])],
                        [float(B_coor[1]) - float(A_coor[1]), float(B_coor[2]) -
                            float(A_coor[2])]]))
    n_y = -det(np.array([[float(C_coor[0]) - float(A_coor[0]), float(C_coor[2]) -
        float(A_coor[2])],
                         [float(B_coor[0]) - float(A_coor[0]), float(B_coor[2]) -
                             float(A_coor[2])]]))
    n_z = det(np.array([[float(C_coor[0]) - float(A_coor[0]), float(C_coor[1]) -
        float(A_coor[1])],
                        [float(B_coor[0]) - float(A_coor[0]), float(B_coor[1]) -
                            float(A_coor[1])]]))
    n = np.array([n_x, n_y, n_z]) / np.sqrt(n_x ** 2 + n_y ** 2 + n_z ** 2)
    if n_z <= 0:
        n = -n
    data_normal[j] = n
    j += 1


# 定义参数
#alpha = (0 / 180) * np.pi
#beta = (90 / 180) * np.pi
v = np.array([np.cos(beta) * np.cos(alpha), np.cos(beta) * np.sin(alpha), np.sin(beta)])
r_0 = -0.534 * 300.4 * v
Energy_per_unit_area = 100


# 主程序
total_energy_absorbed = 0
for i in data_normal:
    # 计算反射方向法矢
```

```
        n = data_normal[i]
        cos_th = dot_p(v, n) / (norm(v) * norm(n))
        v_prim = 2 * cos_th * n - v

        # 计算原来三角形接收到辐射能量
        tmp = data_neib[i]
        A = np.array(data_coor_d[tmp[0]]).astype(float)
        B = np.array(data_coor_d[tmp[1]]).astype(float)
        C = np.array(data_coor_d[tmp[2]]).astype(float)
        Area_Of_Triangle = area(A, B, C)
        # print(Area_Of_Triangle)
        Energy_this_triangle = Energy_per_unit_area * Area_Of_Triangle * cos_th

        # 计算反射光柱到馈源舱平面的新坐标
        r_0 = -0.534 * 300.4 * v
        A_prim = A - v_prim * ((dot_p((A - r_0), v)) / (dot_p(v, v_prim)))
        B_prim = B - v_prim * ((dot_p((B - r_0), v)) / (dot_p(v, v_prim)))
        C_prim = C - v_prim * ((dot_p((C - r_0), v)) / (dot_p(v, v_prim)))

        # 蒙特卡洛模拟 近似 计算馈源舱接受面与三角面的相交区域的面积
        O_ = np.array([(A_prim[0] + B_prim[0] + C_prim[0]) / 3, (A_prim[1] + B_prim[1] +
            C_prim[1]) / 3,
                       (A_prim[2] + B_prim[2] + C_prim[2]) / 3])
        R_ = max(np.sqrt((O_[0] - A_prim[0]) ** 2 + (O_[1] - A_prim[1]) ** 2 + (O_[2] -
            A_prim[2]) ** 2),
                 np.sqrt((O_[0] - B_prim[0]) ** 2 + (O_[1] - B_prim[1]) ** 2 + (O_[2] -
                     B_prim[2]) ** 2),
                 np.sqrt((O_[0] - C_prim[0]) ** 2 + (O_[1] - C_prim[1]) ** 2 + (O_[2] -
                     C_prim[2]) ** 2))
        if (np.sqrt((O_[0] - r_0[0]) ** 2 + (O_[1] - r_0[1]) ** 2 + (O_[2] - r_0[2]) ** 2)) >
            0.5 + R_:
            print('no intersection')
            continue
        else:
            # 蒙特卡洛
            print('i =', i)
            dict_tmp = {'A': list(A_prim), 'B': list(B_prim), 'C': list(C_prim)}
            dict_tmp = sorted(dict_tmp.items(), key=lambda item: item[1])
            Min = (dict_tmp[0])[1]
            Mid = (dict_tmp[1])[1]
            Max = (dict_tmp[2])[1]
            x_range = np.linspace(Min[0], Max[0], 100)
            delta = (Max[0] - Min[0]) / 100
            y_range = {}
            points_tmp = 0
            for xi in x_range:
                if xi <= Mid[0]:
```

```
                l1 = Min[1] + (xi - Min[0]) * (Mid[1] - Min[1]) / (Mid[0] - Min[0])
                l2 = Min[1] + (xi - Min[0]) * (Max[1] - Min[1]) / (Max[0] - Min[0])
                y_range[xi] = np.array(
                    [min(l1, l2) + delta * i for i in range(1000) if (min(l1, l2) + delta * i)
                        <= max(l1, l2)])
                points_tmp += len(y_range[xi])
            else:
                l1 = Max[1] + (xi - Max[0]) * (Mid[1] - Max[1]) / (Mid[0] - Max[0])
                l2 = Max[1] + (xi - Max[0]) * (Min[1] - Max[1]) / (Min[0] - Max[0])
                y_range[xi] = np.array(
                    [min(l1, l2) + delta * i for i in range(1000) if (min(l1, l2) + delta * i)
                        <= max(l1, l2)])
                points_tmp += len(y_range[xi])
        sum_ = 0
        for xi in x_range:
            for yi in y_range[xi]:
                zi = r_0[2] - (n[0] * (xi - r_0[0]) + n[1] * (yi - r_0[1])) / n[2]
                if np.sqrt((xi - r_0[0]) ** 2 + (yi - r_0[1]) ** 2 + (zi - r_0[2]) ** 2) <=
                    0.25:
                    sum_ += 1
        Energy_absorbed = Energy_this_triangle * (sum_) / (points_tmp)
        total_energy_absorbed += Energy_absorbed

    # 计算返回参数
    total_energy = Energy_per_unit_area * (np.pi * 150 ** 2) #* np.sin(beta)
    return total_energy_absorbed / total_energy
```