基于坐标变换和双步优化的定日镜场布局分析

摘要

定日镜作为太阳能光热发电站的关键组件, 在能源和环境领域具有广泛的应用背景. 本文基于在某区域内建设定日镜场的规划, 建立优化模型对定日镜场的建设给出优化方案, 进而对定日镜场的各项指标进行计算.

针对问题一, 依据相关公式计算定日镜场的**光学效率**与**输出热功率**. 首先计算定日镜的光学效率, 其与五个量相关: 其中镜面反射率可近似认为常数, 余弦效率与大气透射率由标准公式求得; 对于阴影遮挡效率, 本文首先进行**坐标变换**, 利用投影法得到阴影遮挡区域, 然后通过将镜面**网格化**, 统计网格中心位于阴影遮挡区域的占比, 得出该定日镜的阴影遮挡效率. 对于集热器截断效率, 利用坐标变换与网格化求解每一个网格的截断效率, 其平均值即为该定日镜的集热器截断效率. 然后, 通过公式计算法向直接辐射辐照度, 得到定日镜场总输出热功率. 计算得到的 **年平均光学效率为 0.5872, 年平均输出热功率为 31.5110MW**, **单位面积镜面年平均输出热功率为 0.5016kW/m^2**.

针对问题二, 建立交替式栏栅渐变布局模型与优化模型对定日镜场的位置布局与相关参数进行求解. 首先对问题一的定日镜输出功率分布结果进行分析, 可知: 吸收塔的位置应在镜场坐标系沿南北方向的直径上, 且尽可能地靠南. 本文以吸收塔为原点在地面建立极坐标系, 提出**交替式栏栅渐变布局**的方法进行布局, 并以单面镜面面积年平均输出热功率最大为目标函数, 构建定日镜场的优化模型, 并在一定约束条件下**剔除**低效率的布局点. 然后利用 **SLSQP 优化算法**求解优化模型得出定日镜场的参数设定值, 最后依据问题一的模型得出定日镜场的各项光学效率与输出热功率. 求解得到的**吸收塔位置坐标为** $(0, -150)$**, 定日镜尺寸为** 6.1 **m** $\times 6.1$**m, 安装高度为** 4.2**m, 定日镜总面数为** 3322 **面, 定日镜总面积为** 123611.62**m^2, 年平均光学效率为 0.4808**, **年平均输出热功率为 60.7236MW**, **单位面积镜面年平均输出热功率为 0.4945kW/m^2**.

针对问题三, **改进定日镜场布局模型**求解定日镜场相关参数. 首先对问题二中的布局方法作出改进, 体现出环之间距离的渐变性, 获得更高的利用效率, 提出**环域块域复合式布局**方法对定日镜场的位置进行布局. 最后根据问题二中的优化模型与优化算法得出需设定的参数值, 再利用问题一的模型计算定日镜场的各项光学效率与输出热效率. 求解得到的**吸收塔位置坐标为** $(0, -150)$**, 定日镜总面数为** 3175 **面, 定日镜总面积为** 109893.75**m^2, 年平均光学效率为 0.4732**, **年平均输出热功率为 52.9MW**, **单位面积镜面年平均输出热功率为 0.4817kW/m^2**.

关键词: 网格化　交替式栏栅渐变布局　**SLSQP** 优化算法　环域块域复合式布局

一、 问题重述

1.1 问题背景

随着全球对可再生能源的需求不断增加, 太阳能被认为是一种清洁、可持续的能量来源. 然而, 太阳能光伏电池技术受限于天气条件与能源存储问题, 而太阳能光热发电技术则为解决这类问题提供了一种有前景的途径.

定日镜利用反射镜将太阳光聚集在集热器上, 将收集的太阳能转化为热能, 最后转化为电能供人们使用. 定日镜的形状为平面矩形, 可以通过纵向转轴和水平转轴实现控制反射角度和方向, 确保太阳光能够准确地聚焦在集热器上, 获得尽可能高的能量收集效率. 本文依据计划在某区域内建设定日镜场的规划, 对定日镜场的建设给出优化方案, 及对建设的定日镜场的各项指标进行计算.

1.2 具体问题重述

根据已知背景信息, 建立数学模型解决以下问题:

问题一: 根据设定的所有定日镜的尺寸为 6m×6m, 安装高度为 4m, 且吸收塔建于圆形定日镜场的中心处, 并依据附件给出的所有定日镜中心的位置, 计算某一年中每月 21 日的各项平均光学效率与输出功率, 及年平均光学效率与年平均输出功率. 将计算结果填入表格中.

问题二: 根据设定的额定功率为 60 MW, 所有定日镜的尺寸和安装高度相同的要求, 设计定日镜场的参数, 包括吸收塔的位置坐标、定日镜尺寸、安装高度、定日镜数量和定日镜位置. 设计目标是在满足额定功率的条件下, 尽量增大单位镜面面积的年平均输出热功率. 将设计结果填入表格中, 并将吸收塔的位置坐标、定日镜尺寸、安装高度和位置坐标按照规定的格式保存到 result2.xlsx 文件中.

问题三: 根据设定的额定功率为 60 MW, 定日镜尺寸和安装高度可以不同的要求, 重新设计定日镜场的各个参数. 目标是在满足额定功率的条件下, 尽量增大单位镜面面积的年平均输出热功率. 将设计结果填入表格中, 并将吸收塔的位置坐标、各定日镜尺寸、安装高度和位置坐标按照规定的格式保存到 result3.xlsx 文件中.

二、 模型假设

为了构建更加精确的模型对问题进行求解, 本文基于实际情况作出如下合理假设:

(1) 定日镜的宽始终与地面平行;

(2) 太阳光只有在被集热器接受时视为锥形光, 其余情况皆视为平行光;

(3) 假设研究的该年份为平年，二月只有 28 天;

(4) 在对定日镜场进行布局时, 中心位于圆形区域边缘的定日镜镜面允许超出圆形区域.

三、 符号约定

符号	意义	单位
ω	太阳时角	度 (°)
ST	当地时间	小时 (h)
δ	太阳正午角	度 (°)
D	从春分节气开始计数的天数	天
φ	当地纬度	度 (°)
α_s	太阳高度角	度 (°)
γ_s	太阳方位角	度 (°)
E_{field}	定日镜场总输出热功率	MW
DNI	法向直接辐射辐照度	W/m^2
N	定日镜数目	面
H	海拔高度	m
η	定日镜镜面光学效率	/
η_c	余弦效率	/
η_a	大气透射率	/
η_s	阴影遮挡效率	/
η_t	集热器截断效率	/
η_r	定日镜镜面折射率	/
Q, T	坐标变换矩阵	/
w_s	单位镜面面积年平均输出热功率	kW/m^2
h	定日镜安装高度	m
p	定日镜镜面宽	m
q	定日镜镜面高	m
r	极径	m
θ	极角	度 (°)

注: 表中未标出的符号以文中首次出现时的说明为准.

四、 问题一模型的建立与求解

4.1 问题一分析

问题一要求根据附件中所给的定日镜坐标对定日镜场的光学效率与输出热功率进行求解. 定日镜场的总输出热功率与太阳光的法向直接辐射辐照度、每一面定日镜的镜面面积及其光学效率相关. 首先根据公式计算得到太阳高度角, 其次根据当地海拔高度确定相关的参数, 最后依据公式得到各定日镜的法向直接辐射辐照度.

求解各个定日镜的光学效率与五个量有关: 余弦效率、大气透射率、阴影遮挡效率、集热器截断效率及镜面折射率. 镜面折射率为常数, 余弦效率与大气透射率由公式易得出, 而阴影遮挡效率、集热器截断效率考虑使用其它方法进行求解.

(1) 对于阴影遮挡效率的求解, 首先通过矩阵变换, 将原镜场坐标系变换为以每一定日镜中心为原点的坐标系, 这样处理极大地简便了后续运算. 其次将目标镜面网格化, 以太阳光的入射中心光线的方向对目标镜面相邻的镜面进行投影, 统计位于阴影区域的网格中心数占比, 即可求得阴影遮挡效率.

(2) 对于集热器截断效率的求解同样采用坐标变换与网格化, 在变换后的球坐标系下对参数进行遍历, 统计能够被集热器接受的光线数占比, 即可求得截断效率. 最后即可求解得到光学效率与输出热功率.

问题一思路分析如图 1 所示:

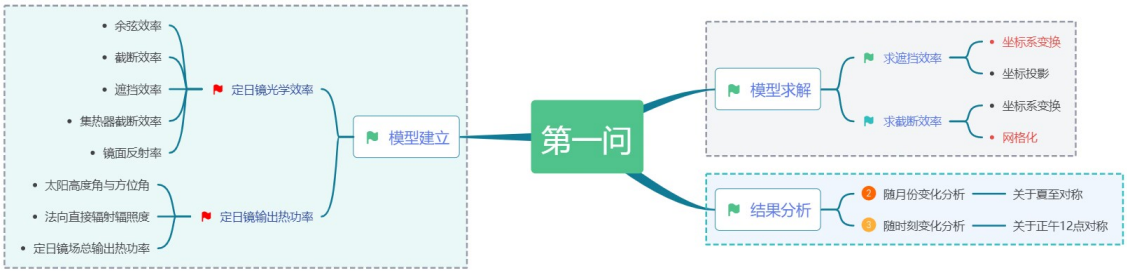


图 **1**　问题一思路分析图

4.2 问题一模型的建立

4.2.1 定日镜光学效率

单面定日镜的光学效率[1] 与多个量相关, 分别有太阳光的余弦效率、大气透射率、阴影遮挡效率、集热器截断效率及镜面反射率.

1. 余弦效率

定日镜的余弦效率取决于太阳入射角, 入射角越小, 余弦效率越高, 有效面积也越大. 反之, 入射角越大, 余弦效率越低, 有效面积也越小. 因此, 确定高余弦效率的定日镜分布区域对于系统的发电效率至关重要.

当照射定日镜时, 太阳入射角 β_s 为太阳光线与定日镜表面法线的夹角, 则余弦效率可由公式计算为:

$$\eta_c = cos\beta_s. \tag{1}$$

2. 大气透射率

本文中的大气透射率是指太阳光在空气中传播时成功穿过的比例. 在太阳光成功从定日镜表面反射, 至成功到达集热器被吸收的过程中, 其大气透射率为:

$$\eta_a = 0.99321 - 0.0001176 d_{HR} + 1.97 \times 10^{-8} \times d_{HR}^2, \tag{2}$$

其中,d_{HR} 为定日镜中心至集热器中心的距离. 设定日镜中心坐标为 (x_i, y_i, z_i), 集热器中心的坐标为 (x_0, y_0, z_0), 则:

$$d_{HR} = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2}.$$

3. 阴影遮挡效率

阴影遮挡[2] 会严重影响定日镜对太阳光的吸收. 太阳光在入射至某一定日镜面的路径上及反射至集热器中心的路径上时, 可能会遭遇到周围其它定日镜的遮挡, 从而造成损失.

对于某一定日镜, 若其光线被遮挡的比例为 η_b, 则阴影遮挡效率 η_s 的计算公式为:

$$\eta_s = 1 - \eta_b. \tag{3}$$

4. 集热器截断效率

太阳光在定日镜上成功反射后, 在反射路径上可能还会存在阴影遮挡造成能量损失, 另外, 未发生阴影遮挡的反射光线并不一定能够完全被集热器接受, 而是与集热器的位置产生一定的偏差, 从而再次造成部分能量损失[3].

假设镜面全反射的能量为 E_1, 反射路径上阴影遮挡损失的能量为 E_2, 成功被集热器接受的能量为 E_3, 则集热器的截断效率为:

$$\eta_t = \frac{E_3}{E_1 - E_2}. \tag{4}$$

5. 单一定日镜光学效率

定日镜的镜面反射率 η_r 可近似认为常数. 那么依据所得的余弦效率、大气透射率、阴影遮挡效率、集热器截断效率及镜面反射率公式, 即可得到每一面定日镜的光学效率

为:

$$\eta = \eta_s \eta_c \eta_a \eta_t \eta_r. \tag{5}$$

4.2.2 定日镜场输出热功率

1. 太阳高度角与方位角

太阳高度角[4], 是指太阳照射某地的光线方向与该地地平线之间的夹角. 可以通过计算该地的太阳时角与太阳正午角来得到太阳高度角.

对于某一地点, 首先计算其太阳时角为:

$$\omega = \frac{\pi}{12}(ST - 12), \tag{6}$$

其中 ST 表示当地时间 (24 小时制). 另外, 该地的太阳正午角 δ 满足公式:

$$sin\delta = sin\frac{2\pi D}{365} sin(\frac{2\pi}{360} 23.45), \tag{7}$$

其中 D 定义为从春分节气开始计算的天数, 规定春分为第 0 天.

若当地纬度为 φ, 那么再根据太阳时角与太阳正午角可以计算太阳高度角 α_s, 其满足的公式为:

$$sin\alpha_s = cos\delta cos\varphi cos\omega + sin\delta sin\varphi, \tag{8}$$

对于太阳方位角 γ_s, 其满足公式:

$$cos\gamma_s = \frac{sin\delta - sin\alpha_s sin\varphi}{cos\alpha_s cos\varphi}. \tag{9}$$

2. 法向直接辐射辐照度

法向直接辐射辐照度, 是指太阳光线垂直于平面的辐射能量, 即地球上某一垂直于太阳光线方向上的单位面积上, 每秒钟接收到的太阳能辐射的能量[5].

其近似计算公式为:

$$\text{DNI} = G_0[a + be^{(-\frac{c}{sin\alpha_s})}], \tag{10}$$

其中 G_0 表示太阳常数 (单位:W/m^2),a,b,c 均为参数. 各参数取值及计算公式如表 1 所示:

其中 H 为当地海拔高度 (单位:km).

3. 定日镜场总输出热功率

对于拥有 N 面定日镜的定日镜场, 其总输出热功率 E_{field} 为:

$$E_{field} = \text{DNI} \cdot \sum_{i}^{N} A_i \eta_i, \tag{11}$$

表 **1**　参数取值及计算公式

参数	取值或计算公式
G_0	1366
a	$0.4237-0.0082(6-H)^2$
b	$0.5055+0.00595(6.5-H)^2$
c	$0..2711+0.01858(2.5-H)^2$

其中 A_i 为第 i 面定日镜对太阳光的采光面积 (单位:m^2),η_i 为第 i 面定日镜的光学效率.

4.3 问题一模型的求解

4.3.1 坐标变换与投影求解阴影遮挡效率

原镜场坐标系为以圆形建设区域圆心为坐标原点, 正东方向作为 x 轴正向, 正北方向作为 y 轴正向, 垂直地面向上为 z 轴正向, 为了便于对每一定日镜的阴影遮挡效率与集热器截断效率进行计算求解, 本文将坐标变换为以定日镜中心为原点, x 轴、y 轴均位于定日镜所在平面,z 轴正向变为垂直镜面向上的新坐标系. 其变换示意图如图 2 所示:

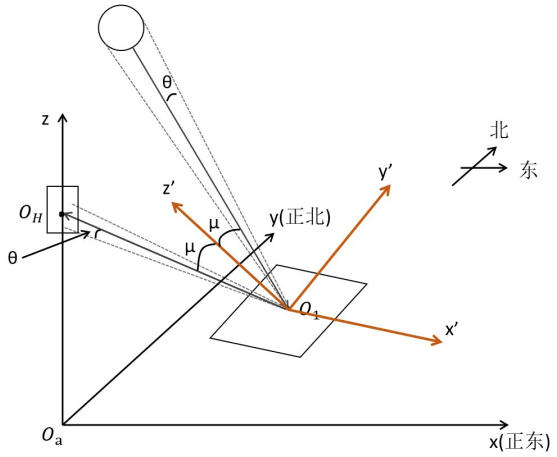


图 **2**　坐标变换示意图

具体变换过程如下文所示.

设原镜场坐标系为 O_a, 变换后的坐标系为 O_1. 由太阳高度角与太阳方位角可以得出太阳入射的中心光线方向向量为:

$$\vec{G}=(cos\alpha_s cos\gamma_s, cos\alpha_s sin\gamma_s, sin\alpha_s),$$

其反射光线与 z 轴的交点为 O_H, 则 O_1 坐标系中的 z' 轴的方向向量为:

$$\overrightarrow{z_1} = \frac{\overrightarrow{O_1O_H} - \vec{G}}{\left|\overrightarrow{O_1O_H} - \vec{G}\right|}.$$

给出矩阵 Q:

$$Q = (\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{z_1}),$$

其中 $\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{z_1}$ 为坐标系 O_1 的坐标轴在坐标系 O_a 中的向量坐标. 对于坐标系 O_a 中的任意一点 $P(x, y, z)$, 其变换为坐标系 O_1 下的坐标 $P_1(x_1, y_1, z_1)$ 为:

$$P_1 = QP.$$

对于某一平面镜, 设其四个顶点为 A,B,C,D, 以 A(a_1, a_2, a_3) 为例, 其坐标变换后得到在坐标系 O_1 下的坐标为 A$'(a_1', a_2', a_3')$. 太阳光中心光线方向向量 $\vec{G} = (g_1, g_2, g_3)$ 坐标变换后为 $\overrightarrow{G'} = (g_1', g_2', g_3')$.

投影过程, 由如下几个步骤进行:

step1 以入射的太阳光中心线方向为基准, 选取目标定日镜旁相邻的定日镜, 照射它的四个顶点.

step2 延长照射光线使照射光线与目标定日镜所在的平面有交点, 连接四个交点, 构成一个新的四边形.

step3 找出该四边形与目标平面镜的重叠区域, 即为所求解的阴影遮挡部分.

依据入射光线作出反射光线, 照此方法同样可求解出反射时的阴影遮挡部分.

投影示意图如图 3 所示:

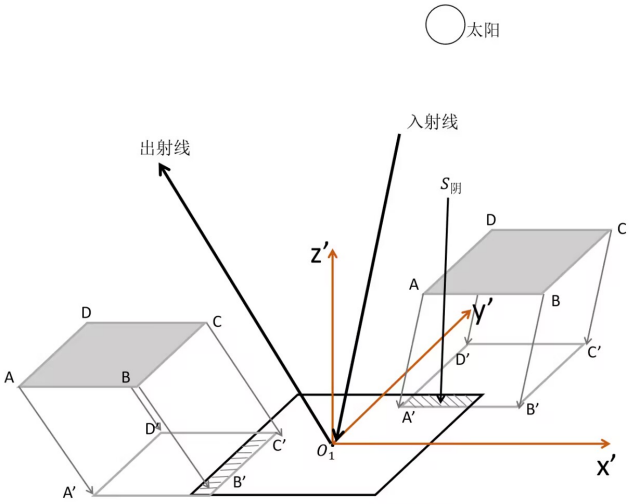


图 **3**　投影示意图

以 $\mathrm{A}'(a_1', a_2', a_3')$ 为例, 计算其投影点如下:

$$\begin{cases} \dfrac{a_1'' - a_1'}{g_1'} = \dfrac{a_2'' - a_2'}{g_2'} = \dfrac{a_3'' - a_3'}{g_3'}, \\ a_3'' = 0, \end{cases}$$

其中 $\mathrm{A}''(a_1'', a_2'', a_3'')$ 为其投影点坐标.

将定日镜划分为 $n \times n$ 个小正方形网格, 对网格中心是否位于阴影遮挡区域内进行判定. 若有 m 个网格中心位于阴影遮挡区域内, 则阴影遮挡效率为:

$$\eta_s = 1 - \frac{m}{n^2}. \tag{12}$$

4.3.2 坐标变换与网格化求解集热器截断效率

对于成功发生反射且未被阴影部分遮挡的太阳光线, 将其视为圆锥形光束, 反射点为圆锥顶点. 为了便于计算求解, 本文对原镜场坐标系再次进行变换, 将其变换为球坐标系, 称为光锥坐标系.

以球心作为坐标原点, 假设球坐标系内存在三条两两垂直的直角坐标轴 x 轴、y 轴、z 轴, 其中 z 轴正向为反射光线方向, 设 x 轴、y 轴的方向向量在原镜场坐标系下的坐标为 $\vec{x} = (x_1, x_2, 0)$,$\vec{y} = (y_1, y_2, y_3)$, 反射光线的方向向量为 $\vec{n}$, 则有:

$$\begin{cases} \vec{n} \cdot \vec{x} = 0, \\ \vec{n} \cdot \vec{y} = 0, \\ \vec{x} \cdot \vec{y} = 0, \end{cases}$$

并令 $x_1 = y_3 = 1$, 可以求解得到一组 $\vec{x}, \vec{y}, \vec{n}$, 对其单位化使其均变为单位向量, 则得到坐标变换的转移矩阵为:

$$T = (\vec{x}, \vec{y}, \vec{n})^T.$$

定义光锥射线在光锥坐标系下的坐标为:

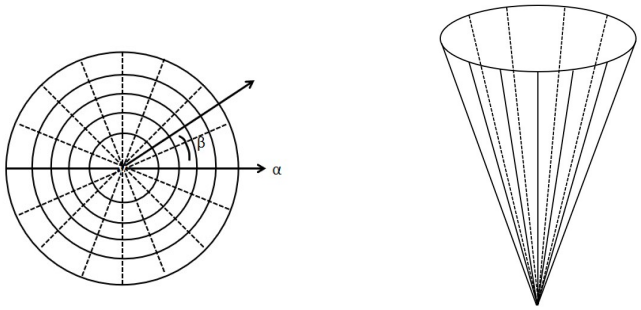
$$\vec{\tau} = (1, \alpha, \beta) = (sin\alpha cos\beta, sin\alpha sin\beta, cos\alpha),$$

其中,$\alpha \in (0, 4.65)$, 单位 (mrad) ; $\beta \in [0, 2\pi)$, 单位 (rad) 则坐标变换后的光锥射线在镜场坐标系下的向量坐标为:

$$\overrightarrow{\tau} = T \cdot \tau = (t_1, t_2, t_3).$$

光锥坐标系与光锥射线如图 4 所示:

将定日镜网格化为 $n \times n$ 个小正方形网格后, 选定其中一个网格, 执行如下步骤:



(a) 光锥坐标系　　(b) 光锥射线

图 **4**　光锥坐标系与光锥射线示意图

step1 对光锥坐标系下的光锥射线中的参数 α 与 β 在其约束范围内开始遍历, 其中 α 以 0.1mrad 为步长,β 以 $\frac{\pi}{180}$ 为步长遍历;

step2 遍历的同时将光锥坐标系下的光锥射线变换为镜场坐标系下的射线;

step3 如果在该镜场坐标系下光锥射线与吸收塔上的集热器有交集, 则说明该条光线被成功接收, 为有效光线; 反之则说明该光线损失. 网格化示意图如图 5 所示:

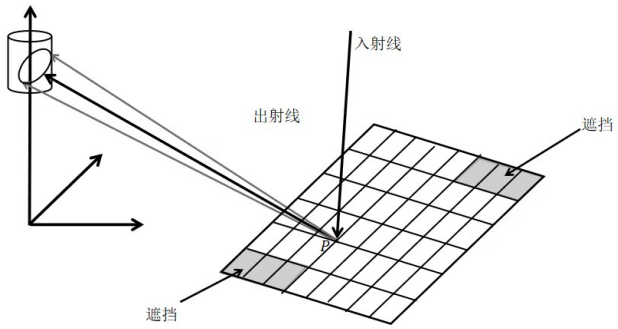


图 **5**　网格划分示意图

step4 重复 step1∼step3, 直至遍历结束, 得到该网格总有效光线数.

共遍历 $360 \times 46 = 16560$ 条光线, 记总有效光线数为 k, 则该网格的集热器截断效率可以用下式计算:

$$\eta_{ti} = \frac{k}{16560}, \tag{13}$$

其中 η_{ti} 为网格 i 的集热器截断效率.

step5 重新选定网格, 重复 step1∼step4, 直至计算出所有网格的集热器截断效率.

最后即可计算出整面定日镜的集热器截断效率为:

$$\eta_t = \frac{\sum_i^N \eta_{ti}}{30^2}. \tag{14}$$

4.3.3 模型求解结果

求解所需要的已知条件及部分设定的参数如下表 2 所示:

表 **2**　已知及设定参数

参数名称	符号	取值及单位
定日镜尺寸	/	$6m \times 6m$
安装高度	/	$4m$
当地纬度	φ	$39.4°$
太阳常数	G_0	$1366\mathrm{W}/m^2$
海拔高度	H	$3km$
镜面反射率	η_r	0.92
网格划分数量	$n \times n$	30^2
定日镜数目	N	1745

以当地时间每月 21 日的 9:00、10:30、12:00、13:30、15:00 为时间节点, 对定日镜场中的所有定日镜的各项平均参数指标进行计算. 计算出的定日镜场总输出热功率由如下公式可以求得单位面积镜面平均输出热功率:

$$E_{ind} = \frac{E_{field}}{1745 \times 36}. \tag{15}$$

计算得到的结果如表 3、表 4 所示:

4.4 问题一结果分析

1. 各月份各项平均效率变化分析

根据求解结果绘制出各项效率随日期变化的曲线如图 6 所示:

对图像结果进行分析可知:

(1) 遮挡效率、截断效率、余弦效率、总光学效率以及输出热功率五个量随月份变化呈现出对称状态, 其它月份的数据以夏至为对称轴, 并在 1 月 21 日到夏至时各项光学效率以不同程度递增.

(2) 夏至到冬至过程中遮挡效率、余弦效率以及总光学效率以不同程度递减, 其中, 遮挡效率变化最小, 余弦效率与总光学效率的变化趋势基本一致.

(3) 截断效率基本趋于 1, 并且在 1 月 21 日到夏至递减, 夏至到冬至递增.

2. 夏至日的输出热功率分布分析

已经分析了平均的各项光学效率变化, 而某一天中定日场中的输出热功率分布未知. 因此, 以夏至这一天为例, 根据模型求解出的夏至 9:00、10:30、12:00、13:30、15:00 的输

表 **3**　问题一每月 **21** 日平均光学效率及输出功率表

日期	平均 光学效率	平均 余弦效率	平均阴影 遮挡效率	平均 截断效率	单位面积镜面平均 输出热功率 (kW/m^2)
1 月 21 日	0.5431	0.7199	0.8500	0.9990	0.4234
2 月 21 日	0.5754	0.7404	0.8770	0.9978	0.4786
3 月 21 日	0.5966	0.7611	0.8860	0.9962	0.5190
4 月 21 日	0.6135	0.7793	0.8901	0.9960	0.5492
5 月 21 日	0.6226	0.7893	0.8912	0.9967	0.5645
6 月 21 日	0.6252	0.7924	0.8911	0.9971	0.5688
7 月 21 日	0.6225	0.7892	0.8912	0.9967	0.5644
8 月 21 日	0.6126	0.7786	0.8896	0.9960	0.5479
9 月 21 日	0.5962	0.7601	0.8866	0.9964	0.5177
10 月 21 日	0.5720	0.7378	0.8748	0.9980	0.4726
11 月 21 日	0.5414	0.7182	0.8491	0.9990	0.4192
12 月 21 日	0.5253	0.7111	0.8319	0.9992	0.3940

表 **4**　问题一年平均光学效率及输出功率表

年平均 光学效率	年平均 余弦效率	年平均阴影 遮挡效率	年平均 截断效率	年平均输出 热功率 (MW)	单位面积镜面年平均 输出热功率 (kW/m^2)
0.5872	0.7565	0.8757	0.9973	31.5110	0.5016

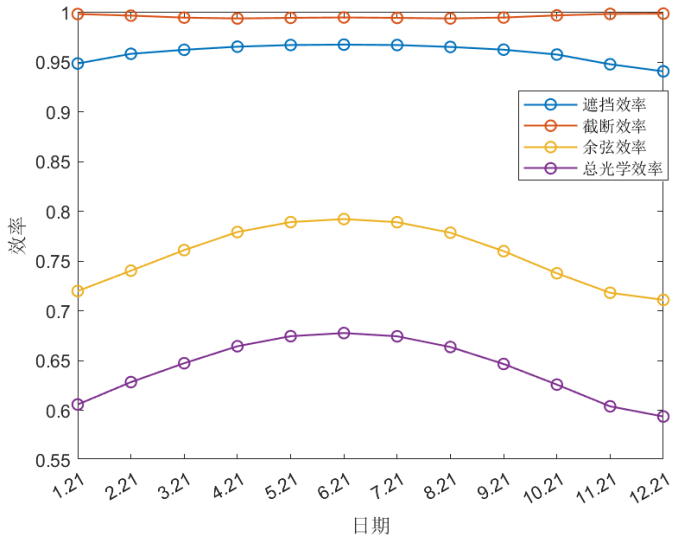
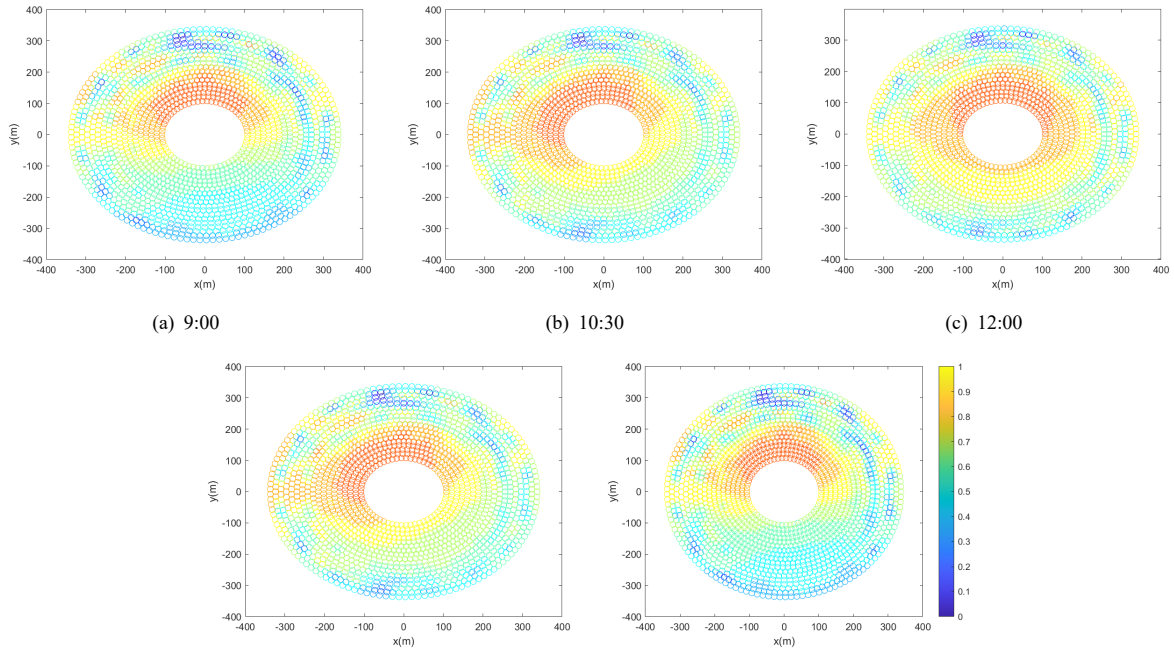


图 **6**　各项效率变化图



(a) 9:00 (b) 10:30 (c) 12:00

(d) 13:30 (e) 15:00

图 **7** 夏至各时刻定日镜输出热功率分布图

出热功率, 绘制出其分布图像如图 7 所示:

分析结果可知, 数据与定日镜输出热功率关于 12:00 对称, 即 9:00 的数据与 15:00 的数据高度相似,10:30 与 13:30 的数据高度相似. 并呈现输出热功率随 9:00 到 12:00 递增,12:00 到 15:00 递减.

分析定日镜输出热功率的分布可知, 高功率的定日镜大多分布在中心圆的北边, 北边的功率总体比南边的功率大, 而且能明显得出南边部分的定日镜在 12:00 的功率大于其它时刻的功率. 此功率分布的分析也为后续优化提供了布局分域与分块上的参考.

五、 问题二模型的建立与求解

5.1 问题二分析

问题二要求在系列约束条件下, 对定日镜场的布局及部分相关参数进行设定, 使得单位镜面面积年平均输出热功率尽可能大, 根据设定的结果求解定日镜场的光学效率与输出热功率. 本文首先提出交替式栏栅渐变布局方法, 以吸收塔为原点建立极坐标系对定日镜场进行布局, 其次在约束条件下以单位镜面面积年平均输出热功率最大为目标函数建立优化模型, 并在一定的约束条件下剔除低效率的布局点. 然后使用 SLSQP 算法对优化模型进行求解, 即可得出定日镜场的参数设定值. 将求解得到的设定参数代入问题一中的模型, 即可得到定日镜场的各项光学效率与输出热功率.

问题二思路分析如图 8 所示:

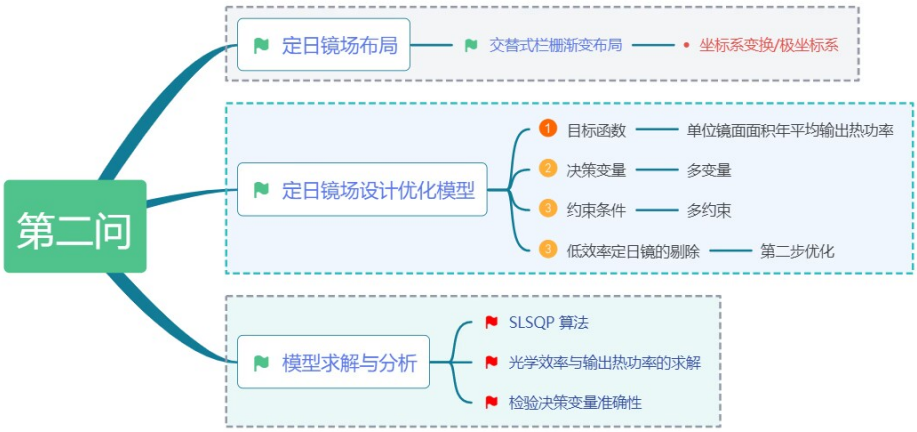


图 **8**　问题二思路分析图

5.2 问题二模型的建立

5.2.1 定日镜场交替式栏栅渐变布局

对问题一的求解结果进行分析, 可以发现输出热效率高的定日镜大部分分布在吸收塔的北部, 这是因为该定日镜场位于北半球, 一年中吸收塔大部分时间均位于太阳与北部定日镜的中间部分, 更容易接受反射光线. 基于此本文对定日镜场进行交替式布局[6], 如图 9 所示:

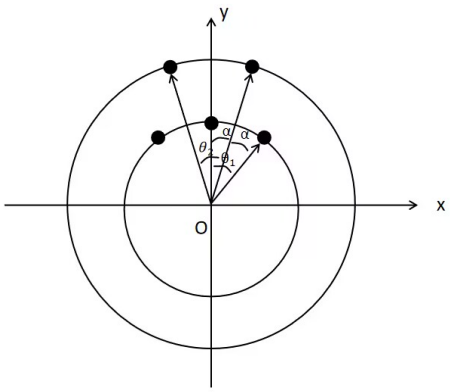


图 **9**　交替式布局图

以吸收塔为原点在地面建立极坐标系, 正东方向的极坐标角度为 0, 任一定日镜与吸收塔的水平距离为 r, 且假定当定日镜与吸收塔的距离处于同一个 r_i 时, 相邻两面定日镜中心与吸收塔连线所成的夹角均为 $\Delta\theta$.

计划建设定日镜场的圆形区域半径为 350m, 设定该吸收塔位于该圆形区域中过圆心且沿南北方向的直径上, 可以使其光线接受更加均匀、对称, 设吸收塔在镜场坐标系下的坐标为 $(x, y, 0)$, 则

$$x = 0, y \in [-350, 350].$$

该吸收塔为极坐标系原点, 那么对于第一环定日镜, 其与吸收塔的距离为 r_0, 同处于该环的相邻定日镜中心与吸收塔的连线所成的夹角为 θ_0, 后续每环增加 $\Delta\theta$, 相邻环之间的距离均为 Δr. 对极坐标的极径与极角进行遍历, 则各定日镜的极坐标可表示为:

$$\begin{cases} \left(r_0+n\Delta r, \frac{\pi}{2} \pm k(\theta_0+n\Delta\theta)\right), n\text{为偶数}, \\ \left(r_0+n\Delta r, \frac{\pi}{2} \pm \frac{1+k}{2}(\theta_0+n\Delta\theta)\right), n\text{为奇数}. \end{cases} \tag{16}$$

其中,n 为布局环数,k 为遍历序数, 遍历以吸收塔的轨迹直径为对称轴左右对称.

需要注意的是, 同一环在利用角度进行遍历时布局时, 会存在超出圆形区域的布局点, 因此在对极径与极角进行遍历时, 同时判断该布局点是否位于圆形区域内. 设该点为 $A_{n,k}$, 圆形区域为集合 C_s, 若:

$$A_{n,k} \in C_s,$$

则该点位于圆形区域内, 为有效布局点.

5.2.2 定日镜场设计的优化模型

需要在限定条件下求解单位镜面面积最大年平均输出热功率, 本文建立多目标优化模型进行求解.

设定日镜的尺寸宽高分别为 p, q, 安装高度为 h, 定日镜面数为 N, 定日镜场额定年平均输出热功率为 E_{field}, 总镜面面积为 $A_{总}$, 单位镜面面积年平均输出热功率为 w_s.

1. 目标函数

单位镜面面积年平均输出热功率计算表达式为:

$$w_s = \frac{E_{field}}{A_{总}} = \frac{\mathrm{DNI}\sum_i^N \eta_i p_i q_i}{\sum_i^N p_i q_i},$$

在定日镜场额定年平均输出热功率达到 60MW 的情况下, 要使单位镜面面积年平均输出热功率最大, 则目标函数为:

$$f = \max w_s.$$

2. 决策变量

本文优化模型的决策变量为吸收塔的横纵坐标、定日镜的宽高及安装高度, 另外还包括相邻两环定日镜之间的距离 Δr_i, 相邻两面定日镜中心与吸收塔连线所成的夹角均为 $\Delta\theta_i$, 即为:

$$\{x, y, p, q, h, \Delta r_i, \Delta\theta_i\}.$$

3. 约束条件

在额定功率不低于 60MW 的前提下, 定日镜镜面边长应该保持在 2m 至 8m 之间, 安

装高度应在 2m 至 6m 之间, 并且安装高度应该保证定日镜在旋转时不会与地面发生触碰. 除此之外, 相邻的两面定日镜底座中心应该比镜面的宽度多 5m 以上.

则约束条件为:

$$\begin{cases} E_{field} \geq 60\text{MW}, \\ 2m \leq p \leq 8m, \\ 2m \leq q \leq 8m, \\ 2m \leq h \leq 6m, \\ \dfrac{q}{2} < h, \\ 2r_i sin\dfrac{\Delta\theta_i}{2} \geq p + 5, \\ |r_{i+1} - r_i| \geq p + 5. \end{cases} \tag{17}$$

4. 模型综述

由此建立优化模型为:

$$\begin{cases} \text{目标函数} : f = \max w_s, \\ \text{决策变量} : \{x, y, p, q, h, \Delta r_i, \Delta\theta_i\}, \\ \text{约束条件} : \text{式 (17)} \end{cases} \tag{18}$$

5.2.3 低效率定日镜的剔除

在定日镜的布局中会存在输出热功率较低的布局点, 从而影响对最优目标的寻找. 因此本文对输出效率低的布局点进行剔除.

定义定日镜场为 Ω:

step1 对于定日镜 S_i, 判断其是否满足剔除条件:

$$\begin{cases} w(S_i) = \min w(\Omega), \\ S_i \in \Omega, \\ E_{field} - w(S_i) \geq 60\text{MW}. \end{cases} \tag{19}$$

step2 若其满足剔除条件, 则将 S_i 剔除:

$$\Omega = \Omega - S_i. \tag{20}$$

重复执行 step1, 直至不满足剔除条件, 结束剔除操作.

5.3 问题二模型的求解

5.3.1 SLSQP 算法求解优化模型

SLSQP 算法是一种用于求解非线性约束优化问题的数值算法。它的基本步骤如下：

step1 初始化: 选择初始变量值 y_0, p_0, q_0, h_0;

step2 确定可行性: 检查变量值是否满足所有约束条件;

step3 构建目标函数和约束条件的梯度: 计算目标函数 $f = \max w_s$ 和约束条件对变量的梯度;

step4 求解线性方程系统: 使用牛顿方法或拟牛顿方法求解关于变量和拉格朗日乘子的线性方程系统;

step5 更新变量: 根据线性方程系统的解更新变量值;

step6 检查停止准则: 比较当前变量值与之前的变量值, 如果满足停止准则则停止迭代, 否则返回 step2;

step7 输出最优解：返回最优变量值;

通过迭代以上步骤,SLSQP 算法可以寻找本文中优化问题的最优解, 已知 $r_0 = 100$m, 且设定 $\Delta r = p + 5$, 则其最优参数如表 5 所示:

表 **5**　问题二设计参数表

吸收塔 位置坐标	定日镜尺寸 (底 × 高)	定日镜 安装高度 (m)	定日镜 总面数	定日镜 总面积 (m^2)
$(0, -150\text{m})$	6.1×6.1	4.2	3322	123611.62

5.3.2 光学效率与输出热功率的求解

根据求解出的参数, 利用问题一中的模型求解定日镜场的光学效率与输出热功率如表 6 所示:

5.4 问题二结果分析与检验

5.4.1 结果分析

并在最佳决策变量条件下, 由吸收塔纵坐标 y、渐变半径差、定日镜宽 p 以及定日镜间距约束, 共同确定了定日镜场中每一面定日镜的坐标, 进一步计算可求出每一面定日镜的单位面积镜面平均输出热功率, 从而描绘出定日镜场的坐标分布以及平均输出热功率的分布情况. 将数据可视化, 如图 10 所示:

表 **6**　问题二每月 **21** 日平均光学效率及输出功率表

日期	平均 光学效率	平均 余弦效率	平均阴影 遮挡效率	平均 截断效率	单位面积镜面平均 输出热功率 (kW/m^2)
1 月 21 日	0.4289	0.6488	0.7765	1	0.4031
2 月 21 日	0.4694	0.6662	0.8143	1	0.4699
3 月 21 日	0.4993	0.6857	0.8307	1	0.5219
4 月 21 日	0.5101	0.6920	0.8321	1	0.5483
5 月 21 日	0.5208	0.7088	0.8266	1	0.5669
6 月 21 日	0.5240	0.7148	0.8240	1	0.5724
7 月 21 日	0.5249	0.7179	0.8240	1	0.5712
8 月 21 日	0.5156	0.7046	0.8284	1	0.5537
9 月 21 日	0.4894	0.6655	0.8368	1	0.5104
10 月 21 日	0.4648	0.6639	0.8104	1	0.4622
11 月 21 日	0.4139	0.6193	0.7865	1	0.3857
12 月 21 日	0.4080	0.6418	0.7538	1	0.3688

表 **7**　问题二年平均光学效率及输出功率表

年平均 光学效率	年平均 余弦效率	年平均阴影 遮挡效率	年平均 截断效率	年平均输出 热功率 (MW)	单位面积镜面年平均 输出热功率 (kW/m^2)
0.4808	0.6775	0.8120	1	60.7236	0.4945

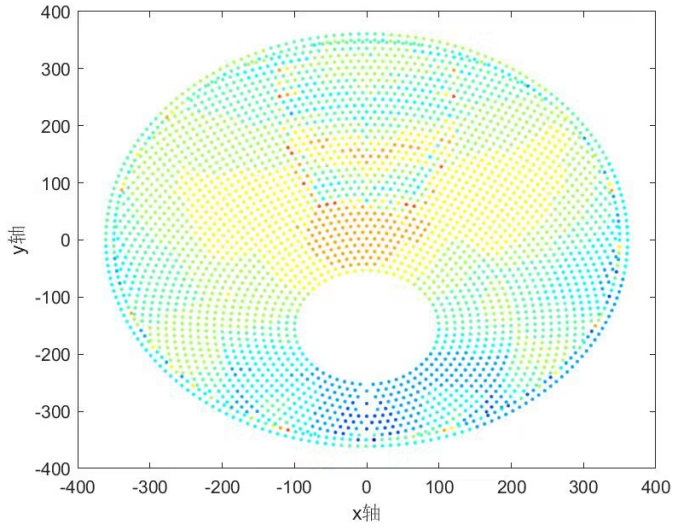
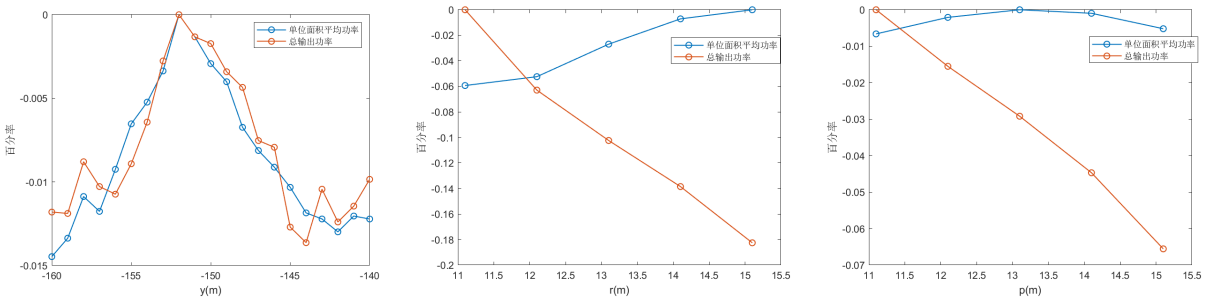


图 **10**　定日镜及其功率分布图



(a) 吸收塔纵坐标 y　　(b) 渐变半径差 Δr　　(c) 定日镜宽 a

图 **11**　功率随决策变量变化曲线

分析计算结果可知, 吸收塔坐落于镜场中心的正南方向 150 米的位置, 吸收塔北边相应位置上定日镜的输出热功率大于正南方向上的输出热功率. 此外, 输出热功率以放射型向北方延展并逐渐减小.

5.4.2 结果检验

由于智能算法求解优化算法时受初始点的影响较大, 易陷入局部最优, 因此在决策变量的最优解附近选取一较大范围, 计算决策变量在不同的数值下功率的变化.

(1) 吸收塔纵坐标 y

由图 11(a) 可知最优点在-153, 此处将吸收塔纵坐标 y 从-160m 以 1m 为步长遍历到-140m, 得到单位面积平均功率和总输出功率相对最优解误差随 y 变化的趋势, 可知, 最优解确为最高点, 从-160m 到-153m 之间, 两个功率均逐步上升到最大值, 而-153m 之后便逐渐减小.

(2) 渐变半径差 Δr

观察图 11(b), 有一约束条件是 >a+5, 因此从 a+5 到 a+10 以 1 为步长遍历, 最优解中 a 为 6.1, 即：11.1 为的左边界. 在增大的过程中, 单位面积平均功率递增, 但除去 11.1 外的四个点均不符合总功率大于 60MW 的条件, 因此在约束条件的约束下,11.1 确实为最优解.

(3) 定日镜宽 a

由图 11(c), 可知最优点在 6m 附近, 此处将定日镜宽 a 从 5.8m 以 0.1m 为步长遍历到 6.2m, 得到单位面积平均功率和总输出功率相对最优解误差随 y 变化的趋势, 可知, 单位面积平均功率是目标函数, 其最高点即为最优点, 从 5.8m 到 6m 之间, 单位面积平均功率功率逐步上升到最大值, 而 6m 之后便逐渐减小; 总输出功率作为约束条件, 其中 5.8m 到 6m 的总输出功率大于 60MW, 因此符合优化过程.

六、 问题三模型的建立与求解

6.1 问题三分析

在问题二的基础上, 问题三设定各个定日镜的尺寸与安装高度不尽相同. 首先对问题二中的定日镜布局方式进行改进, 采用环域块域复合式布局方法对定日镜场进行布局, 其次利用问题二中的优化模型与优化算法对定日镜场需设定的参数进行求解, 将求解得到的参数代入问题一中的模型, 计算定日镜场的各项光学效率与输出热功率.

问题二思路分析如图 12 所:

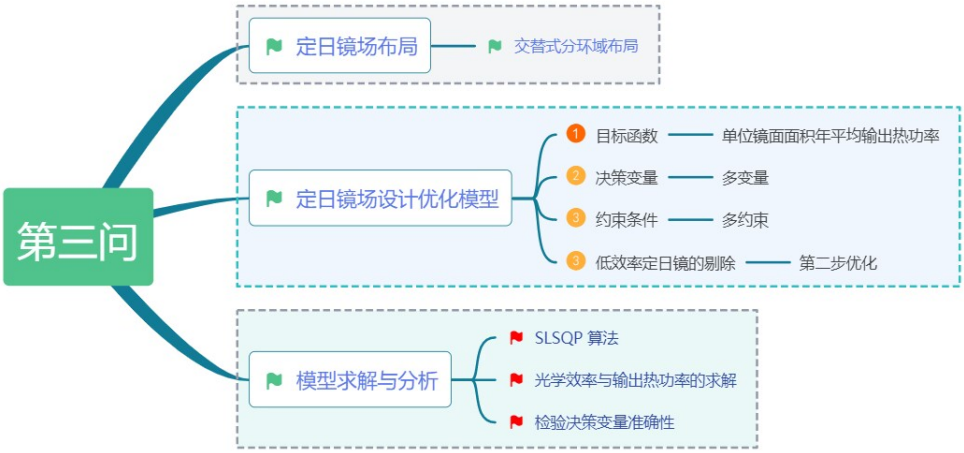


图 **12**　问题三思路分析图

6.2 定日镜场布局模型的改进

与问题二不同的是, 问题三各个定日镜的尺寸与安装高度不尽相同. 本文设定同一环内的各个定日镜尺寸与安装高度相同, 各定日镜中心在极坐标系中的极径与极角的设定均与问题二中的布局相同.

6.2.1 交替式栏栅渐变布局的改进

对问题二中的交替式栏栅渐变布局的改进, 相邻环之间的距离仍相同, 为 Δr, 但相邻环中的镜面尺寸与安装高度服从渐变式变化. 设定第一环的镜面宽高及安装高度为:

$$\{p_1, q_1, h_1\},$$

假设渐变系数为 ζ_1, ζ_2, 则第 n 环的镜面宽高及安装高度为:

$$\{\zeta_1^{n-1} p_1, \zeta_1^{n-1} q_1, \zeta_2^{n-1} h_1\},$$

6.2.2 交替式分环域布局

对问题二中的交替式分环域布局进行改进: 假设距离吸收塔最远的环与最近的环沿任一极径方向的线段长度为 L_{max}, 则将该线段三等分使得圆环被分为三个环域, 依据与吸收塔的距离由近到远分别记为 U_1, U_2, U_3.

假定处于同一环域 $U_i(i = 1, 2, 3)$ 内的定日镜尺寸与安装高度均相同, 尺寸为 $p_i \times q_i$, 安装高度为 h_i.

6.2.3 环域块域复合式布局

结合以上交替式栏栅渐变布局与交替式分环域布局方式, 提出环域块域复合式布局方法.

假设有 $3N_0$ 个环, 首先对以吸收塔为中心的区域中的 $3N_0$ 个环数进行环域划分, 分为三部分, 每部分包含 N_0 个环, 其次利用栏栅渐变式布局的方法得到三部分的镜面宽高及安装高度为:

$$\begin{cases} \text{环域:} U_1 : \{p_1, q_1, h_1\}, \\ \text{环域:} U_2 : \{\zeta_1 p_1, \zeta_1 q_1, \zeta_2 h_1\}, \\ \text{环域:} U_3 : \left\{\zeta_1^2 p_1, \zeta_1^2 q_1, \zeta_2^2 h_1\right\}. \end{cases}$$

由前文结果经验可知北部定日镜输出热功率较南部高, 因此最后以吸收塔为圆心在北部划分 120° 的圆心角, 将其划分为南北两个区块, 实现了块域划分.

6.3 问题三模型的求解

6.3.1 设定参数的求解

成功布局后, 利用问题二的优化模型对定日镜场的参数设定进行优化, 不同的是问题三中增加了变量 ζ_1 , ζ_2. 然后利用 SLSQP 算法求解需设定的参数, 求解结果如表 8 所示:

表 **8**　问题三设计参数表

吸收塔 位置坐标	定日镜尺寸 (底 × 高)	定日镜 安装高度 (m)	定日镜 总面数	定日镜 总面积 (m^2)
(0,-150)	/	/	3175	109893.75

6.3.2 光学效率与输出热功率求解

求解出定日镜场的相关参数后, 将其代入问题一中的模型, 求解定日镜场的各项光学效率与输出热功率如表 9、表 10 所示:

表 **9**　问题三每月 **21** 日平均光学效率及输出功率表

日期	平均 光学效率	平均 余弦效率	平均阴影 遮挡效率	平均 截断效率	单位面积镜面平均 输出热功率 (kW/m^2)
1 月 21 日	0.4372	0.7986	0.6432	1	0.4111
2 月 21 日	0.4752	0.8336	0.6603	1	0.4759
3 月 21 日	0.5030	0.8476	0.6795	1	0.5258
4 月 21 日	0.5119	0.8476	0.6850	1	0.5502
5 月 21 日	0.5230	0.8420	0.7021	1	0.5693
6 月 21 日	0.5263	0.8393	0.7083	1	0.5749
7 月 21 日	0.5187	0.8444	0.6983	1	0.5571
8 月 21 日	0.4913	0.8531	0.6582	1	0.5123
9 月 21 日	0.4709	0.8302	0.6580	1	0.4685
10 月 21 日	0.4184	0.8056	0.6121	1	0.3899
11 月 21 日	0.3856	0.7965	0.6042	1	0.3873
12 月 21 日	0.4169	0.7765	0.6364	1	0.3772

表 **10**　问题三年平均光学效率及输出功率表

年平均 光学效率	年平均 余弦效率	年平均阴影 遮挡效率	年平均 截断效率	年平均输出 热效率 (MW)	单位面积镜面年平均 输出热功率 (kW/m^2)
0.4732	0.6573	0.8263	1	52.9	0.4817

6.4 问题三结果分析

问题三求解得到的定日镜位置与功率分布如图 13 所示:

观察图像, 其红色部分为功率分布第一级, 黄色部分为功率分布第二级, 蓝色部分为功率分布第三级, 紫色部分为功率分布第四级, 整体仍呈现出北部效率高于南部.

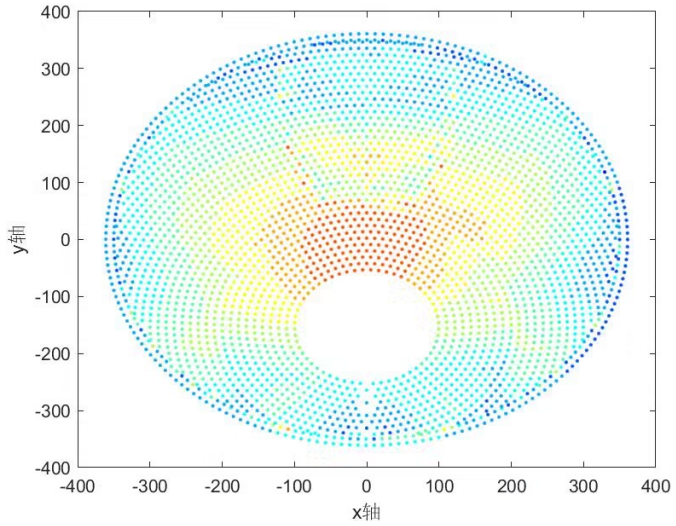


图 **13**　定日镜位置与功率分布图

七、 模型评价

7.1 模型的优点

(1) 问题一中模型求解采用坐标变换, 以研究对象中心为原点建立坐标系, 使求解过程得到了极大的简化.

(2) 本文问题二中的优化模型具有较多地变量与约束条件, 而 SLSQP 优化算法应用于本文具有大量变量和约束条件的问题时, 体现出高效性、全局优化能力及灵活的约束处理能力.

(3) 问题三将问题二中两种布局方式加以结合, 提出环域块域复合式布局方法对定日镜场进行布局.

7.2 模型的缺点

(1) 算法求解结果的时间复杂度较高, 导致任务执行时间延长，影响实时性和效率.

(2)SLSQP 优化算法容易陷入局部最优, 布局具有局限性, 不能同时考虑余弦效率、阴影遮挡效率及集热器截断效率, 使单位面积年平均输出功率输出最大.

参考文献

[1] 张平等，太阳能塔式光热镜场光学效率计算方法 [J]，技术与市场，2021，28(6):5-8.

[2] 张茂龙, 卫慧敏, 杜小泽等. 塔式太阳能镜场阴影与遮挡效率的改进算法 [J]. 太阳能学报,2016,37(08):1998-2003.

[3] 周春, 赵晓凯. 线聚焦菲涅尔太阳能集热器阵列集热性能理论与实验研究 [J]. 科技展望,2014(18):135-136.

[4] 蔡志杰，太阳影子定位 [J]，数学建模及其应用，2015，4(4):25-33

[5] 张墨耕, 韩兆辉, 顾鹏程等. 塔式光热技术在综合能源系统中的应用 [J]. 新疆石油天然气,2022,18(02):71-77.

[6] 高博, 刘建兴, 孙浩等. 基于自适应引力搜索算法的定日镜场优化布置 [J]. 太阳能学报,2022,43(10):119-125.

附录 A 支撑材料清单

1.result2.xlsx

2.result3.xlsx

附录 B 源程序

B.1 问题一源程序

```
import numpy as np
import math
import pandas as pd
from scipy.spatial import cKDTree
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# 附件读取
df = pd.read_excel('附件.xlsx')
x = [] # 定日镜的x轴坐标
y = [] # 定日镜的y轴坐标
xy = []
if 'x坐标 (m)' in df.columns and 'y坐标 (m)' in df.columns:
x = df['x坐标 (m)'].tolist()
y = df['y坐标 (m)'].tolist()

for h in range(len(x)):
xy.append((x[h], y[h]))
# print(x)

h_Heliostat = 4 # 定日镜高度（m）
h_Absorption_tower = 80 # 吸收塔高度（m）
h_Collector = 8 # 集热器高度（m）
r_Collector = 3.5 # 集热器半径（m）
H = 3 # 海拔高度(km)
g0 = 1.366 # 太阳常数（kW/m^2）
a = 0.4237 - 0.00821 * ((6 - H) ** 2)
b = 0.5055 - 0.00595 * ((6.5 - H) ** 2)
c = 0.2711 - 0.01858 * ((2.5 - H) ** 2)
A = [] # 采光镜的面积

# st = 12 # 当地时间
st1 = [9, 10.5, 12]
a1 = 39.4 * math.pi / 180 # 当地纬度(faai)
# a2 = 0 # 太阳赤纬角(deerta)
# d = -60 # 以春分作为第 0 天起算的天数
```

```
day = [-59, -28, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275]


def Sun_altitude_angle_and_sun_azimuth():
# 太阳高度角
w = math.pi * (st - 12) / 12 # 太阳时角
sin_a2 = math.sin(2 * math.pi * d / 365) * math.sin(2 * math.pi * 23.45 / 360)
a2 = math.asin(sin_a2)
sin_a_s = math.cos(a2) * math.cos(a1) * math.cos(w) + math.sin(a2) * math.sin(a1) #
    太阳高度角a_s
a_s = math.asin(sin_a_s)
cos_r_s = (math.sin(a2) - math.sin(a_s) * math.sin(a1)) / (math.cos(a_s) * math.cos(a1)) #
    太阳方位角
if cos_r_s > 1 or cos_r_s < -1:
r_s = math.pi / 2
else:
r_s = math.acos(cos_r_s)

print(a_s * 180 / math.pi)
print(r_s)
return a_s, r_s


def Calculation_of_output_thermal_power(a_s, yt_sb, yt_trunc, yt_cos): # 法向直接辐射辐照度
yita = [] # 定日镜的光学效率
e_field = 0 # 定日镜场的输出热功率初始化
dni = g0 * (a + b * math.exp(-c / math.sin(a_s)))
for i in range(len(x)):
d_hr = math.sqrt(x[i] ** 2 + y[i] ** 2 + 76 ** 2) # 镜面中心到集热器中心的距离
# yt_sb = 1 - 0 # 0为阴影遮挡损失
# yt_cos = 1 - 0 # 0为余弦损失
# yt_trunc = o / (p - q)
yt_at = 0.99321 - 0.0001176 * d_hr + 1.97 * (10 ** (-8)) * d_hr ** 2
yt_ref = 0.92
yt = yt_sb * yt_cos * yt_at * yt_trunc * yt_ref
yita.append(yt)
for i in range(len(x)):
e_field += dni * 36 * yita[i]
return e_field / len(yita), sum(yita) / len(yita)


def Coordinate_system_transformation(a_s, r_s, i): #
    坐标系变换,a_s为太阳高度角, r_s为太阳方位角
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
s_collector = (0, 0, 80) # 集热器中心点的坐标
i_reflected_light = np.array([-x[i], -y[i], 76]) # 每一块采光镜的反射的光线的向量
```

```
vi_daylighting_mirrors = (i_s / np.linalg.norm(i_s) + i_reflected_light / np.linalg.norm(
i_reflected_light)) / 2 # 采光镜的法线方向（即z轴的法向量）
z_reflected_light = vi_daylighting_mirrors / np.linalg.norm(vi_daylighting_mirrors) #
    归一化法向量以求得采光镜平面下的z轴
x_reflected_light = np.array(
[1, -z_reflected_light[0] / z_reflected_light[1], 0]) # 求得在采光镜平面下的x轴
y_reflected_light = np.cross(z_reflected_light, x_reflected_light) #
    使用叉积法则确定采光镜平面的y轴
# print(z_reflected_light, '0')
return z_reflected_light, x_reflected_light / np.linalg.norm(x_reflected_light),
    y_reflected_light / np.linalg.norm(
y_reflected_light)


def Coordinate_system_transformation2(a_s, r_s, point): #
    坐标系变换,a_s为太阳高度角，r_s为太阳方位角
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
s_collector = (0, 0, 80) # 集热器中心点的坐标
i_reflected_light = np.array([-point[0], -point[1], 76]) # 每一块采光镜的反射的光线的向量
vi_daylighting_mirrors = (i_s / np.linalg.norm(i_s) + i_reflected_light / np.linalg.norm(
i_reflected_light)) / 2 # 采光镜的法线方向（即z轴的法向量）
z_reflected_light = vi_daylighting_mirrors / np.linalg.norm(vi_daylighting_mirrors) #
    归一化法向量以求得采光镜平面下的z轴
x_reflected_light = np.array(
[1, -z_reflected_light[0] / z_reflected_light[1], 0]) # 求得在采光镜平面下的x轴
y_reflected_light = np.cross(z_reflected_light, x_reflected_light) #
    使用叉积法则确定采光镜平面的y轴
# print(z_reflected_light, '0')
return z_reflected_light, x_reflected_light / np.linalg.norm(x_reflected_light),
    y_reflected_light / np.linalg.norm(
y_reflected_light)


def Shadow_block_occlusion_calculation(a_s, r_s): # 阴影遮挡和反射遮挡面积计算
point = [(3, 3, 0), (3, -3, 0), (-3, 3, 0), (-3, -3, 0)]
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
p_jieduan = [] # 截断效率
p_shadow = [] # 阴影效率
p_yuxian = [] # 余弦效率
for i in range(len(x)):
matrix = np.zeros((30, 30))
i_reflected_light = np.array([-x[i], -y[i], 76]) # 每一块采光镜的反射的光线的向量并单位化
i_reflected_light_1 = i_reflected_light / np.linalg.norm(i_reflected_light) # 单位化的结果
z_a, x_a, y_a = Coordinate_system_transformation(a_s, r_s, i) # 首先得到a采光镜的xyz坐标
j_reflected_light_a = (x[i], y[i], 4) # a采光镜的位置坐标
```

```
cos_yuxian = np.dot(i_reflected_light_1, z_a)
yuxian = math.acos(cos_yuxian)
p_yuxian.append(math.cos(yuxian))
# print(Coordinate_system_transformation(a_s, r_s, i))
#  print(i)
# 创建二维KD树
tree = cKDTree(xy)

# 要查找附近的点的目标点
target_point = (x[i], y[i])

# 搜索最近的k个点
k = 4 # 你可以根据需要修改k的值
nearest_point_indices = tree.query(target_point, k=k)[1]
nearest_points = [xy[j] for j in nearest_point_indices]
for pp in range(len(nearest_points)):
# for points in nearest_points:
if nearest_points[pp] != target_point:
points = nearest_points[pp]
xy_shadow = []
xy_block = []
# if x[j] == x[i] and y[j] == y[i]:

for p in point:
j_reflected_light_b = (points[0], points[1], 4) # b采光镜的位置坐标
z_b, x_b, y_b = Coordinate_system_transformation2(a_s, r_s, points) #
    首先得到b(干扰镜）采光镜的xyz坐标
T_b = np.array([ # b镜的转换矩阵
[x_b[0], y_b[0], z_b[0]],
[x_b[1], y_b[1], z_b[1]],
[x_b[2], y_b[2], z_b[2]]
])
T_a = np.array([ # a镜的转换矩阵
[x_a[0], y_a[0], z_a[0]],
[x_a[1], y_a[1], z_a[1]],
[x_a[2], y_a[2], z_a[2]]
])
i_s_h = np.dot(T_a.T, i_s)
i_reflected_light_1_h = np.dot(T_a.T, i_reflected_light_1)
H_b = p # 需要进行转换的坐标
H_b1 = np.dot(T_b, H_b) + j_reflected_light_b # 先转换到地面坐标系上
H_b2 = np.dot(T_a.T, H_b1 - j_reflected_light_a) # 转换到a镜坐标系中的坐标
x_shadow_b = -(i_s_h[0] * H_b2[2]) / i_s_h[2] + H_b2[0]
y_shadow_b = -(i_s_h[1] * H_b2[2]) / i_s_h[2] + H_b2[1] # 得到变换完成的阴影遮挡的坐标
point1 = (x_shadow_b, y_shadow_b)
xy_shadow.append(point1)
```

```
# print(H_b1)

x_block_b = -(i_reflected_light_1_h[0] * H_b2[2]) / i_reflected_light_1_h[2] + H_b2[0]
y_block_b = -(i_reflected_light_1_h[1] * H_b2[2]) / i_reflected_light_1_h[2] + H_b2[
1] # 得到变换完成的阴影遮挡的坐标
xy_block.append((x_block_b, y_block_b))

x_x = -3.1

# print(xy_shadow)
# print(point1)
for ii in range(30):
x_x += 0.2
y_y = -3.1
for jj in range(30):
y_y += 0.2

result1 = point_inside_quadrilateral((x_x, y_y), xy_shadow)
result2 = point_inside_quadrilateral((x_x, y_y), xy_block)
'''
if result1:
matrix[ii][jj] = 1
elif result2:
matrix[ii][jj] = 2
'''
if result1 or result2:
matrix[ii][jj] = 1
n_n = np.array([1, -x_a[0] / x_a[1], -(y_a[0] - x_a[0] * y_a[1] / x_a[1]) / y_a[2]])
n_n_h = n_n / np.linalg.norm(n_n)
T_c = np.array([ # 锥形光的转换矩阵
[x_a[0], y_a[0], n_n_h[0]],
[x_a[1], y_a[1], n_n_h[1]],
[x_a[2], y_a[2], n_n_h[2]]
])
# matrix2 = np.zeros((100, 100))

if matrix[ii][jj] == 0:
H_t_1 = np.dot(T_a, (x_x, y_y, 0)) + j_reflected_light_a # 网格点在地面上的坐标
matrix[ii][jj] = 3 + has_intersection_with_cylinder(H_t_1, i_reflected_light_1,
np.array([0, 0, 76]),
3.5, 8)
# print(H_t_1, (x_x, y_y, 0), matrix[ii][jj])
p_shadow.append((np.count_nonzero(matrix == 1) + np.count_nonzero(matrix == 2)) / 900)
p_jieduan.append(
np.count_nonzero(matrix == 4) / (900 - (np.count_nonzero(matrix == 1) +
    np.count_nonzero(matrix == 2))))
# print(len(p_shadow))
```

```
# print(np.count_nonzero(matrix == 1), i)
# print(matrix)
return sum(p_jieduan) / len(p_jieduan), 1 - (sum(p_shadow) / len(p_shadow)), sum(p_yuxian) /
    len(p_yuxian)


'''
def has_intersection_with_cylinder(line_start, line_direction, cylinder_center,
    cylinder_radius, cylinder_height):
# 计算直线与圆柱面的交点
a = np.dot(line_direction, line_direction)
b = 2 * np.dot(line_direction, line_start - cylinder_center)
c = np.dot(line_start - cylinder_center, line_start - cylinder_center) - (cylinder_radius **
    2)

# 计算判别式
discriminant = b ** 2 - 4 * a * c

if discriminant >= 0:
t1 = (-b + np.sqrt(discriminant)) / (2 * a)
t2 = (-b - np.sqrt(discriminant)) / (2 * a)

# 计算交点的高度坐标
z1 = line_start[2] + t1 * line_direction[2]
z2 = line_start[2] + t2 * line_direction[2]

# 检查交点是否在圆柱体的高度范围内
if (cylinder_center[2] <= z1 <= cylinder_center[2] + cylinder_height) or \
(cylinder_center[2] <= z2 <= cylinder_center[2] + cylinder_height):
return 1

return 0
'''


def has_intersection_with_cylinder(line_start, line_direction, cylinder_center,
    cylinder_radius, cylinder_height):
# 计算直线与圆柱面的交点
a = line_direction[0] ** 2 + line_direction[1] ** 2
b = 2 * line_direction[0] * (line_direction[2] * line_start[0] - line_direction[0] *
    line_start[2]) + 2 * \
line_direction[1] * (line_direction[2] * line_start[1] - line_direction[1] * line_start[2])
c = (line_direction[2] * line_start[0] - line_direction[0] * line_start[2]) ** 2 + (
line_direction[2] * line_start[1] - line_direction[1] * line_start[2]) ** 2 - (
3.5 * line_direction[2]) ** 2
```

```
# 计算判别式
discriminant = b ** 2 - 4 * a * c

if discriminant >= 0:
z1 = (-b + np.sqrt(discriminant)) / (2 * a)
z2 = (-b - np.sqrt(discriminant)) / (2 * a)

# 检查交点是否在圆柱体的高度范围内
if (cylinder_center[2] <= z1 <= cylinder_center[2] + cylinder_height) or \
(cylinder_center[2] <= z2 <= cylinder_center[2] + cylinder_height):
return 1

return 0


'''
def point_inside_quadrilateral(point, vertices): # 判断一个点是否在一个四边形内部
# point是要判断的点的坐标，vertices是四边形的顶点坐标列表
x, y = point
n = len(vertices)
inside = False

p1x, p1y = vertices[0]
for i in range(n + 1):
p2x, p2y = vertices[i % n]
if y > min(p1y, p2y):
if y <= max(p1y, p2y):
if x <= max(p1x, p2x):
if p1y != p2y:
xinters = (y - p1y) * (p2x - p1x) / (p2y - p1y) + p1x
if p1x == p2x or x <= xinters:
inside = not inside
p1x, p1y = p2x, p2y

return inside
'''


def point_inside_quadrilateral(point, vertices): # 判断一个点是否在一个四边形内部
x, y = point
# print(vertices)
# print(vertices[2][0])
if ((vertices[2][0] - vertices[3][0]) * (y - vertices[2][1]) / (vertices[2][1] -
    vertices[3][1]) + vertices[2][
0]) <= x <= ((vertices[0][0] - vertices[1][0]) * (y - vertices[0][1]) / (vertices[0][1] -
    vertices[1][1]) +
vertices[0][
```

```
0]) and ((vertices[1][1] - vertices[3][1]) * (x - vertices[1][0]) / (
vertices[1][0] - vertices[3][0]) +
vertices[1][1]) <= y <= ((vertices[0][1] - vertices[2][1]) * (x - vertices[0][0]) / (
vertices[0][0] - vertices[2][0]) + vertices[0][1]):
return 1
else:
return 0


def Calculation_of_truncation_efficiency(): # 集热器截断效率的计算
matrix = np.zeros((100, 100)) # 创建统计数组
i = -1
j = -1
for x in range(-3, 3, 100):
i += 1
for y in range(-3, 3, 100):
j += 1

return 0


if __name__ == '__main__':
for d in day:
p_shadow_1 = []
p_jieduan_1 = []
p_yuxian_1 = []
yita_1 = []
E_1 = []
for st in st1:
a_s, r_s = Sun_altitude_angle_and_sun_azimuth()
p_jieduan, p_shadow, p_yuxian = Shadow_block_occlusion_calculation(a_s, r_s)
E, yita = Calculation_of_output_thermal_power(a_s, p_shadow, p_jieduan, p_yuxian)
print(d)
print('阴影遮挡', p_shadow)
print('截断效率', p_jieduan)
print('余弦效率', p_yuxian)
print('总效率', yita)
print('平均功率', E)
p_shadow_1.append(p_shadow)
p_jieduan_1.append(p_jieduan)
p_yuxian_1.append(p_yuxian)
yita_1.append(yita)
E_1.append(E)
print('平均阴影遮挡', (2 * p_shadow_1[0] + 2 * p_shadow_1[1] + p_shadow_1[2]) / 5)
print('平均截断效率', (2 * p_jieduan_1[0] + 2 * p_jieduan_1[1] + p_jieduan_1[2]) / 5)
print('平均余弦效率', (2 * p_yuxian_1[0] + 2 * p_yuxian_1[1] + p_yuxian_1[2]) / 5)
print('平均总效率', (2*yita_1[0]+2*yita_1[1]+yita_1[2]) / 5)
```

```
print('平均平均功率', (2*E_1[0]+2*E_1[1]+E_1[2]) / 5)
```

B.2 问题二源程序

```
import numpy as np
import math
from scipy.spatial import cKDTree
import pandas as pd
import openpyxl
import random
import matplotlib.pyplot as plt
from deap import base, creator, tools

h_Heliostat = 4 # 定日镜高度（m）
h_Absorption_tower = 80 # 吸收塔高度（m）
h_Collector = 8 # 集热器高度（m）
r_Collector = 3.5 # 集热器半径（m）
H = 3 # 海拔高度(km)
g0 = 1.366 # 太阳常数（kW/m^2）
a = 0.4237 - 0.00821 * ((6 - H) ** 2)
b = 0.5055 - 0.00595 * ((6.5 - H) ** 2)
c = 0.2711 - 0.01858 * ((2.5 - H) ** 2)
A = [] # 采光镜的面积

# st = 12 # 当地时间
st1 = [9, 10.5, 12]
a1 = 39.4 * math.pi / 180 # 当地纬度(faai)

# a2 = 0 # 太阳赤纬角(deerta)
# d = 0 # 以春分作为第 0 天起算的天数


day = [-59, -28, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275]


def Sun_altitude_angle_and_sun_azimuth(st, d):
# 太阳高度角
w = math.pi * (st - 12) / 12 # 太阳时角
sin_a2 = math.sin(2 * math.pi * d / 365) * math.sin(2 * math.pi * 23.45 / 360)
a2 = math.asin(sin_a2)
sin_a_s = math.cos(a2) * math.cos(a1) * math.cos(w) + math.sin(a2) * math.sin(a1) #
    太阳高度角a_s
a_s = math.asin(sin_a_s)
cos_r_s = (math.sin(a2) - math.sin(a_s) * math.sin(a1)) / (math.cos(a_s) * math.cos(a1))
```

```
    # 太阳方位角
if cos_r_s > 1 or cos_r_s < -1:
r_s = math.pi / 2
else:
r_s = math.acos(cos_r_s)

# print(a_s * 180 / math.pi)
# print(r_s)
return a_s, r_s


def Calculation_of_output_thermal_power(i, points, a_s, yt_sb, yt_trunc, yt_cos, y, a1,
    b1, h): # 法向直接辐射辐照度,a1,b1为镜子的长款
aerf = 1.2
dni = g0 * aerf * (a + b * math.exp(-c / math.sin(a_s)))
d_hr = math.sqrt(points[i][0] ** 2 + (points[i][1] - y) ** 2 + (80 - h) ** 2) #
    镜面中心到集热器中心的距离
yt_at = 0.99321 - 0.0001176 * d_hr + 1.97 * (10 ** (-8)) * d_hr ** 2
yt_ref = 0.92
yt = yt_sb * yt_cos * yt_at * yt_trunc * yt_ref # 定日镜的光学效率
e_field = dni * a1 * b1 * yt # 定日镜的输出热功率
return e_field, yt


def Coordinate_system_transformation2(a_s, r_s, point, y, h): #
    坐标系变换,a_s为太阳高度角，r_s为太阳方位角
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
s_collector = (0, 0, 80) # 集热器中心点的坐标
i_reflected_light = np.array([-point[0], y - point[1], 80 - h]) #
    每一块采光镜的反射的光线的向量
vi_daylighting_mirrors = (i_s / np.linalg.norm(i_s) + i_reflected_light / np.linalg.norm(
i_reflected_light)) / 2 # 采光镜的法线方向（即z轴的法向量）
z_reflected_light = vi_daylighting_mirrors / np.linalg.norm(vi_daylighting_mirrors) #
    归一化法向量以求得采光镜平面下的z轴
x_reflected_light = np.array(
[1, -z_reflected_light[0] / z_reflected_light[1], 0]) # 求得在采光镜平面下的x轴
y_reflected_light = np.cross(z_reflected_light, x_reflected_light) #
    使用叉积法则确定采光镜平面的y轴
# print(z_reflected_light, '0')
return z_reflected_light, x_reflected_light / np.linalg.norm(x_reflected_light),
    y_reflected_light / np.linalg.norm(
y_reflected_light)


def mirror_layout(y, a): #
    整体场地布局，y为集热器纵坐标,dita_r为两排镜子之间的间隔,sita_1为初始间隔角度，dita_sita为第n圈的缩小角,a
```

```
point_Collector = (0, y, 80) # 集热器的坐标
dita_r = a + 5
r1 = 100 # 吸收塔100m内不设置镜子
points_mirror_coordinates = [] # 用于储存镜子的坐标
for ll in range(200):
points_mirror_coordinates.append((350 * math.cos(0.0314 * ll), 350 * math.sin(0.0314 *
    ll)))
for lll in range(200):
points_mirror_coordinates.append((361 * math.cos(0.0314 * lll), 361 * math.sin(0.0314 *
    lll)))
N3 = int((250 + y) / dita_r)
for n3 in range(N3):
if n3 % 2 == 1:
points_mirror_coordinates.append((0, (y - r1 - n3 * dita_r)))
for n1 in range(1 + int((350 - y - r1) / dita_r)): # n1为布置的周数
r = r1 + n1 * dita_r
sita = (a + 5) / r
sita_11 = 0
if judge_circle_intersection(0, 0, 350, 0, y, r) == 1: # 判断是否有交点
sita_all = 2 * math.pi
x1 = 0
sita_10 = math.pi
for nn in range(0, int(math.pi / sita)):
if n1 % 2 == 1:
point1 = (r * math.sin(nn * sita), (r * math.cos(nn * sita) + y))
point2 = (-r * math.sin(nn * sita), (r * math.cos(nn * sita) + y))
else:
point1 = (r * math.sin((0.5 + nn) * sita), (r * math.cos((0.5 + nn) * sita) + y))
point2 = (-r * math.sin((0.5 + nn) * sita), (r * math.cos((0.5 + nn) * sita) + y))
points_mirror_coordinates.append(point1)
points_mirror_coordinates.append(point2)

else:
x2, y2, x1, y1 = find_circle_intersection(0, 0, 350, 0, y, r) # 求解大圆和布置小圆的交点
sin_sita = abs(x2 - x1) / (2 * math.sqrt(x1 ** 2 + (y1 - y) ** 2)) # sita的一半的正弦值
sita_0 = 2 * math.asin(sin_sita)
if y1 > y:
sita_all = sita_0
sita_11 = math.asin(-x1 / r)
else:
sita_all = 2 * math.pi - sita_0
sita_11 = math.pi - math.asin(-x1 / r) # 左交点与y轴的夹角
N2 = int(sita_11 / sita) # 每一周需要布置的镜子的个数
for n2 in range(N2):
if n1 % 2 == 1:
point1 = (r * math.sin(n2 * sita), (r * math.cos(n2 * sita) + y))
point2 = (-r * math.sin(n2 * sita), (r * math.cos(n2 * sita) + y))
```

```
    else:
    point1 = (r * math.sin((0.5 + n2) * sita), (r * math.cos((0.5 + n2) * sita) + y))
    point2 = (-r * math.sin((0.5 + n2) * sita), (r * math.cos((0.5 + n2) * sita) + y))
    points_mirror_coordinates.append(point1)
    points_mirror_coordinates.append(point2)
    if n1 % 2 == 1:
    points_mirror_coordinates.append((0, -(r + y)))
    # print(int((350 - y - r1) / dita_r))
    return points_mirror_coordinates


def judge_circle_intersection(x1, y1, r1, x2, y2, r2): # 求解两个圆的交点
# 计算两个圆心之间的距离
d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# 判断是否存在交点
if d > r1 + r2 or d < abs(r1 - r2):
return 1 # 无交点
else:
return 0


def find_circle_intersection(x1, y1, r1, x2, y2, r2): # 求解两个圆的交点
# 计算两个圆心之间的距离
d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# 计算交点坐标
P_x = (x2 - x1) / d
P_y = (y2 - y1) / d

a = (r1 ** 2 - r2 ** 2 + d ** 2) / (2 * d)
h = math.sqrt(r1 ** 2 - a ** 2)

x3 = x1 + a * P_x - h * P_y
y3 = y1 + a * P_y + h * P_x
x4 = x1 + a * P_x + h * P_y
y4 = y1 + a * P_y - h * P_x

return x3, y3, x4, y4


def point_inside_quadrilateral(point, vertices): # 判断一个点是否在一个四边形内部
x, y = point
# print(vertices)
# print(vertices[2][0])
if ((vertices[2][0] - vertices[3][0]) * (y - vertices[2][1]) / (vertices[2][1] -
    vertices[3][1]) + vertices[2][
```

```
0]) <= x <= ((vertices[0][0] - vertices[1][0]) * (y - vertices[0][1]) / (vertices[0][1] -
    vertices[1][1]) +
vertices[0][
0]) and ((vertices[1][1] - vertices[3][1]) * (x - vertices[1][0]) / (
vertices[1][0] - vertices[3][0]) +
vertices[1][1]) <= y <= ((vertices[0][1] - vertices[2][1]) * (x - vertices[0][0]) / (
vertices[0][0] - vertices[2][0]) + vertices[0][1]):
return 1
else:
return 0


def has_intersection_with_cylinder(line_start, line_direction, cylinder_center,
    cylinder_radius, cylinder_height):
# 计算直线与圆柱面的交点
a = line_direction[0] ** 2 + line_direction[1] ** 2
b = 2 * line_direction[0] * (line_direction[2] * line_start[0] - line_direction[0] *
    line_start[2]) + 2 * \
line_direction[1] * (line_direction[2] * line_start[1] - line_direction[1] *
    line_start[2])
c = (line_direction[2] * line_start[0] - line_direction[0] * line_start[2]) ** 2 + (
line_direction[2] * line_start[1] - line_direction[1] * line_start[2]) ** 2 - (
3.5 * line_direction[2]) ** 2

# 计算判别式
discriminant = b ** 2 - 4 * a * c

if discriminant >= 0:
z1 = (-b + np.sqrt(discriminant)) / (2 * a)
z2 = (-b - np.sqrt(discriminant)) / (2 * a)

# 检查交点是否在圆柱体的高度范围内
if (cylinder_center[2] <= z1 <= cylinder_center[2] + cylinder_height) or \
(cylinder_center[2] <= z2 <= cylinder_center[2] + cylinder_height):
return 1

return 0


def Average_annual_power_output_per_mirror(r_s, a_s, point_s, a1, b1, y, h): #
    用于计算每个镜子的年平均输出功率
point = [(a1 / 2, b1 / 2, 0), (a1 / 2, -b1 / 2, 0), (-a1 / 2, b1 / 2, 0), (-a1 / 2, -b1 /
    2, 0)]
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
p_jieduan = [] # 截断效率
p_shadow = [] # 阴影效率
```

```
p_yuxian = [] # 余弦效率
for i in range(len(point_s)):
matrix = np.zeros((30, 30))
i_reflected_light = np.array([-point_s[i][0], y - point_s[i][1], 80 - h]) #
    每一块采光镜的反射的光线的向量并单位化
i_reflected_light_1 = i_reflected_light / np.linalg.norm(i_reflected_light) # 单位化的结果
z_a, x_a, y_a = Coordinate_system_transformation2(a_s, r_s, point_s[i], y, h) #
    首先得到a采光镜的xyz坐标
j_reflected_light_a = (point_s[i][0], point_s[i][1], h) # a采光镜的位置坐标
cos_yuxian = np.dot(i_reflected_light_1, z_a)
yuxian = math.acos(cos_yuxian)
p_yuxian.append(math.cos(yuxian))
# print(Coordinate_system_transformation(a_s, r_s, i))
#  print(i)
# 创建二维KD树
tree = cKDTree(point_s)

# 要查找附近的点的目标点
target_point = (point_s[i][0], point_s[i][1])

# 搜索最近的k个点
if len(point_s) >= 4:
k = 4 # 你可以根据需要修改k的值
else:
k = len(point_s)
nearest_point_indices = tree.query(target_point, k=k)[1]
nearest_points = [point_s[j] for j in nearest_point_indices]
for pp in range(len(nearest_points)):
# for points in nearest_points:
if nearest_points[pp] != target_point:
points = nearest_points[pp]
xy_shadow = []
xy_block = []
# if x[j] == x[i] and y[j] == y[i]:

for p in point:
j_reflected_light_b = (points[0], points[1], h) # b采光镜的位置坐标
z_b, x_b, y_b = Coordinate_system_transformation2(a_s, r_s, points, y, h) #
    首先得到b(干扰镜）采光镜的xyz坐标
T_b = np.array([ # b镜的转换矩阵
[x_b[0], y_b[0], z_b[0]],
[x_b[1], y_b[1], z_b[1]],
[x_b[2], y_b[2], z_b[2]]
])
T_a = np.array([ # a镜的转换矩阵
[x_a[0], y_a[0], z_a[0]],
[x_a[1], y_a[1], z_a[1]],
```

```
    [x_a[2], y_a[2], z_a[2]]
    ])
    i_s_h = np.dot(T_a.T, i_s)
    i_reflected_light_1_h = np.dot(T_a.T, i_reflected_light_1)
    H_b = p # 需要进行转换的坐标
    H_b1 = np.dot(T_b, H_b) + j_reflected_light_b # 先转换到地面坐标系上
    H_b2 = np.dot(T_a.T, H_b1 - j_reflected_light_a) # 转换到a镜坐标系中的坐标
    x_shadow_b = -(i_s_h[0] * H_b2[2]) / i_s_h[2] + H_b2[0]
    y_shadow_b = -(i_s_h[1] * H_b2[2]) / i_s_h[2] + H_b2[1] # 得到变换完成的阴影遮挡的坐标
    point1 = (x_shadow_b, y_shadow_b)
    xy_shadow.append(point1)

    # print(H_b1)

    x_block_b = -(i_reflected_light_1_h[0] * H_b2[2]) / i_reflected_light_1_h[2] + H_b2[0]
    y_block_b = -(i_reflected_light_1_h[1] * H_b2[2]) / i_reflected_light_1_h[2] + H_b2[
    1] # 得到变换完成的阴影遮挡的坐标
    xy_block.append((x_block_b, y_block_b))

    x_x = -a1 / 2 - a1 / 60

    # print(xy_shadow)
    # print(point1)
    for ii in range(30):
    x_x += a1 / 30
    y_y = -b1 / 2 - b1 / 60
    for jj in range(30):
    y_y += b1 / 30

    result1 = point_inside_quadrilateral((x_x, y_y), xy_shadow)
    result2 = point_inside_quadrilateral((x_x, y_y), xy_block)
    '''
    if result1:
    matrix[ii][jj] = 1
    elif result2:
    matrix[ii][jj] = 2
    '''
    if result1 or result2:
    matrix[ii][jj] = 1
    n_n = np.array([1, -x_a[0] / x_a[1], -(y_a[0] - x_a[0] * y_a[1] / x_a[1]) / y_a[2]])
    n_n_h = n_n / np.linalg.norm(n_n)
    T_c = np.array([ # 锥形光的转换矩阵
    [x_a[0], y_a[0], n_n_h[0]],
    [x_a[1], y_a[1], n_n_h[1]],
    [x_a[2], y_a[2], n_n_h[2]]
    ])
    # matrix2 = np.zeros((100, 100))
```

```
if matrix[ii][jj] == 0:
H_t_1 = np.dot(T_a, (x_x, y_y, 0)) + j_reflected_light_a # 网格点在地面上的坐标
matrix[ii][jj] = 3 + has_intersection_with_cylinder(H_t_1, i_reflected_light_1,
np.array([0, y, 76]),
3.5, 8)
# print(H_t_1, (x_x, y_y, 0), matrix[ii][jj])
p_shadow.append((np.count_nonzero(matrix == 1) + np.count_nonzero(matrix == 2)) / 900)
if 900 - (np.count_nonzero(matrix == 1) + np.count_nonzero(matrix == 2)) == 0:
p_jieduan.append(0)
else:
p_jieduan.append(
np.count_nonzero(matrix == 4) / (900 - (np.count_nonzero(matrix == 1) +
    np.count_nonzero(matrix == 2))))
return p_jieduan, p_shadow, p_yuxian


def constraint_function(solution): # 构建约束
t = 1
for n1 in range(int((350 - solution[0] - 100) / solution[1])): # n1为布置的周数
r = 100 + n1 * solution[1] # 镜子与吸收塔的距离
sita = solution[2] - n1 * solution[3] # 注意sita的约束
if 2 * r * math.sin(sita / 2) <= solution[4] + 5 and solution[1] <= solution[4] + 5 and
    solution[5] / 2 > \
solution[6] and (solution[4] < 2 or solution[4] > 8) and (solution[5] < 2 or solution[5]
    > 8) and (
solution[6] < 2 or solution[6] > 6):
t = 0
if t == 0:
return False
else:
return True


def evaluate(individual): # 最大单位面积功率
y, a2, b2, h, st, d = individual
e_field = []
yt = []
a_s, r_s = Sun_altitude_angle_and_sun_azimuth(st, d)
points = mirror_layout(y, a2)
p_jieduan, p_shadow, p_yuxian = Average_annual_power_output_per_mirror(r_s, a_s, points,
    a2, b2, y, h)
for i in range(len(points)):
e_field1, yt1 = Calculation_of_output_thermal_power(i, points, a_s, 1 - p_shadow[i], 1 -
    p_jieduan[i],
p_yuxian[i], y,
a2,
```

```
    b2, h)
    e_field.append(e_field1)
    yt.append(yt1)
    return sum(e_field) / (len(points) * a2 * b2), sum(p_yuxian) / len(p_yuxian),
        sum(p_shadow) / len(p_shadow), sum(
    p_jieduan) / len(p_jieduan), sum(yt) / len(yt), e_field


    if __name__ == '__main__':
    '''
    st1 = [9, 10.5, 12]
    day = [31, 61, 92, 122, 153, 184, 214, 245, 275]



    # 提取x和y坐标
    x = [point[0] for point in points]
    y = [point[1] for point in points]

    # 创建一个散点图
    plt.scatter(x, y, label='Points', color='blue', marker='o')

    # 添加标题和坐标轴标签
    plt.title('Scatter Plot of Points')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')

    # 显示图例
    plt.legend()

    # 显示图形
    plt.show()

    # 示例点的坐标列表
    points = mirror_layout(-150, 6)
    print(len(points))
    # print(points)
    individual = np.array([-150, 6.1, 6.1, 4.2])
    a = evaluate(individual)
    print(a)
    '''
    # print(a * 6.1 * 6.1 * len(points))
    points = mirror_layout(-153, 6.1)

    # 创建一个包含点坐标的 pandas 数据帧
    df = pd.DataFrame(points, columns=['X', 'Y'])
```

```
# 将数据帧写入Excel文件
df.to_excel('result2.xlsx', index=False)
print(points)
x = []
y = []

d = 92
yt = []
p_field1 = []
p_jieduan1 = []
p_shadow1 = []
p_yuxian1 = []
p_yt1 = []
st = 12
a_s, r_s = Sun_altitude_angle_and_sun_azimuth(st, d)
p_field, p_jieduan, p_shadow, p_yuxian, p_yt, e_field = evaluate(
individual=np.array([-150, 6.1, 6.1, 4.2, st, d]))
p_field1.append(p_field)
p_jieduan1.append(p_jieduan)
p_shadow1.append(p_shadow)
p_yuxian1.append(p_yuxian)
p_yt1.append(p_yt)
print(e_field)
'''
yt.append((2 * p_field1[0] + 2 * p_field1[1] + p_field1[2]) / 5)
yt.append((2 * p_jieduan1[0] + 2 * p_jieduan1[1] + p_jieduan1[2]) / 5)
yt.append((2 * p_shadow1[0] + 2 * p_shadow1[1] + p_shadow1[2]) / 5)
yt.append((2 * p_yuxian1[0] + 2 * p_yuxian1[1] + p_yuxian1[2]) / 5)
yt.append((2 * p_yt1[0] + 2 * p_yt1[1] + p_yt1[2]) / 5)
'''
p
```

B.3 问题三源程序

```
mport numpy as np
import math
from scipy.spatial import cKDTree
import matplotlib.pyplot as plt
import random
from scipy.optimize import minimize
from deap import base, creator, tools

h_Heliostat = 4 # 定日镜高度（m）
h_Absorption_tower = 80 # 吸收塔高度（m）
h_Collector = 8 # 集热器高度（m）
```

```
r_Collector = 3.5 # 集热器半径（m）
H = 3 # 海拔高度(km)
g0 = 1.366 # 太阳常数（kW/m^2）
a = 0.4237 - 0.00821 * ((6 - H) ** 2)
b = 0.5055 - 0.00595 * ((6.5 - H) ** 2)
c = 0.2711 - 0.01858 * ((2.5 - H) ** 2)
A = [] # 采光镜的面积

st = 10.5 # 当地时间
st1 = [9, 10.5, 12]
a1 = 39.4 * math.pi / 180 # 当地纬度(faai)
# a2 = 0 # 太阳赤纬角(deerta)
d = 0 # 以春分作为第 0 天起算的天数

day = [ 245, 275]


def Sun_altitude_angle_and_sun_azimuth(st, d):
# 太阳高度角
w = math.pi * (st - 12) / 12 # 太阳时角
sin_a2 = math.sin(2 * math.pi * d / 365) * math.sin(2 * math.pi * 23.45 / 360)
a2 = math.asin(sin_a2)
sin_a_s = math.cos(a2) * math.cos(a1) * math.cos(w) + math.sin(a2) * math.sin(a1) #
    太阳高度角a_s
a_s = math.asin(sin_a_s)
cos_r_s = (math.sin(a2) - math.sin(a_s) * math.sin(a1)) / (math.cos(a_s) * math.cos(a1))
    # 太阳方位角
if cos_r_s > 1 or cos_r_s < -1:
r_s = math.pi / 2
else:
r_s = math.acos(cos_r_s)

# print(a_s * 180 / math.pi)
# print(r_s)
return a_s, r_s


def Calculation_of_output_thermal_power(i, points, a_s, yt_sb, yt_trunc, yt_cos, y, a1,
    b1, h): # 法向直接辐射辐照度,a1,b1为镜子的长款
aerf = 1.2
dni = g0 * aerf * (a + b * math.exp(-c / math.sin(a_s)))
d_hr = math.sqrt(points[i][0] ** 2 + (points[i][1] - y) ** 2 + (80 - h) ** 2) #
    镜面中心到集热器中心的距离
yt_at = 0.99321 - 0.0001176 * d_hr + 1.97 * (10 ** (-8)) * d_hr ** 2
yt_ref = 0.92
yt = yt_sb * yt_cos * yt_at * yt_trunc * yt_ref # 定日镜的光学效率
e_field = dni * a1 * b1 * yt # 定日镜的输出热功率
```

```
    return e_field, yt


def Coordinate_system_transformation2(a_s, r_s, point, y, h): #
    坐标系变换,a_s为太阳高度角，r_s为太阳方位角
i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
    math.sin(a_s)]) # 太阳光线向量
s_collector = (0, 0, 80) # 集热器中心点的坐标
i_reflected_light = np.array([-point[0], y - point[1], 80 - h]) #
    每一块采光镜的反射的光线的向量
vi_daylighting_mirrors = (i_s / np.linalg.norm(i_s) + i_reflected_light / np.linalg.norm(
i_reflected_light)) / 2 # 采光镜的法线方向（即z轴的法向量）
z_reflected_light = vi_daylighting_mirrors / np.linalg.norm(vi_daylighting_mirrors) #
    归一化法向量以求得采光镜平面下的z轴
x_reflected_light = np.array(
[1, -z_reflected_light[0] / z_reflected_light[1], 0]) # 求得在采光镜平面下的x轴
y_reflected_light = np.cross(z_reflected_light, x_reflected_light) #
    使用叉积法则确定采光镜平面的y轴
# print(z_reflected_light, '0')
return z_reflected_light, x_reflected_light / np.linalg.norm(x_reflected_light),
    y_reflected_light / np.linalg.norm(
y_reflected_light)


def mirror_layout(y, a): #
    整体场地布局，y为集热器纵坐标,dita_r为两排镜子之间的间隔,sita_1为初始间隔角度，dita_sita为第n圈的缩小角,a
point_Collector = (0, y, 80) # 集热器的坐标
dita_r = a + 5
r1 = 100 # 吸收塔100m内不设置镜子
points_mirror_coordinates = [] # 用于储存镜子的坐标
for ll in range(200):
points_mirror_coordinates.append((350 * math.cos(0.0314 * ll), 350 * math.sin(0.0314 *
    ll)))
for lll in range(200):
points_mirror_coordinates.append((361 * math.cos(0.0314 * lll), 361 * math.sin(0.0314 *
    lll)))
N3 = int((250 + y) / dita_r)
for n3 in range(N3):
if n3 % 2 == 1:
points_mirror_coordinates.append((0, y - r1 - n3 * dita_r))
for n1 in range(1 + int((350 - y - r1) / dita_r)): # n1为布置的周数
r = r1 + n1 * dita_r
sita = (a + 5) / r
sita_11 = 0
if judge_circle_intersection(0, 0, 350, 0, y, r) == 1: # 判断是否有交点
sita_all = 2 * math.pi
x1 = 0
```

```
sita_10 = math.pi
for nn in range(0, int(math.pi / sita)):
if n1 % 2 == 1:
point1 = (r * math.sin(nn * sita), r * math.cos(nn * sita) + y)
point2 = (-r * math.sin(nn * sita), r * math.cos(nn * sita) + y)
else:
point1 = (r * math.sin((0.5 + nn) * sita), r * math.cos((0.5 + nn) * sita) + y)
point2 = (-r * math.sin((0.5 + nn) * sita), r * math.cos((0.5 + nn) * sita) + y)
points_mirror_coordinates.append(point1)
points_mirror_coordinates.append(point2)

else:
x2, y2, x1, y1 = find_circle_intersection(0, 0, 350, 0, y, r) # 求解大圆和布置小圆的交点
sin_sita = abs(x2 - x1) / (2 * math.sqrt(x1 ** 2 + (y1 - y) ** 2)) # sita的一半的正弦值
sita_0 = 2 * math.asin(sin_sita)
if y1 > y:
sita_all = sita_0
sita_11 = math.asin(-x1 / r)
else:
sita_all = 2 * math.pi - sita_0
sita_11 = math.pi - math.asin(-x1 / r) # 左交点与y轴的夹角
N2 = int(sita_11 / sita) # 每一周需要布置的镜子的个数
for n2 in range(N2):
if n1 % 2 == 1:
point1 = (r * math.sin(n2 * sita), r * math.cos(n2 * sita) + y)
point2 = (-r * math.sin(n2 * sita), r * math.cos(n2 * sita) + y)
else:
point1 = (r * math.sin((0.5 + n2) * sita), r * math.cos((0.5 + n2) * sita) + y)
point2 = (-r * math.sin((0.5 + n2) * sita), r * math.cos((0.5 + n2) * sita) + y)
points_mirror_coordinates.append(point1)
points_mirror_coordinates.append(point2)
if n1 % 2 == 1:
points_mirror_coordinates.append((0, r + y))
# print(int((350 - y - r1) / dita_r))
return points_mirror_coordinates


def judge_circle_intersection(x1, y1, r1, x2, y2, r2): # 求解两个圆的交点
# 计算两个圆心之间的距离
d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# 判断是否存在交点
if d > r1 + r2 or d < abs(r1 - r2):
return 1 # 无交点
else:
return 0
```

```
def find_circle_intersection(x1, y1, r1, x2, y2, r2): # 求解两个圆的交点
# 计算两个圆心之间的距离
d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# 计算交点坐标
P_x = (x2 - x1) / d
P_y = (y2 - y1) / d

a = (r1 ** 2 - r2 ** 2 + d ** 2) / (2 * d)
h = math.sqrt(r1 ** 2 - a ** 2)

x3 = x1 + a * P_x - h * P_y
y3 = y1 + a * P_y + h * P_x
x4 = x1 + a * P_x + h * P_y
y4 = y1 + a * P_y - h * P_x

return x3, y3, x4, y4


def point_inside_quadrilateral(point, vertices): # 判断一个点是否在一个四边形内部
x, y = point
# print(vertices)
# print(vertices[2][0])
if ((vertices[2][0] - vertices[3][0]) * (y - vertices[2][1]) / (vertices[2][1] -
    vertices[3][1]) + vertices[2][
0]) <= x <= ((vertices[0][0] - vertices[1][0]) * (y - vertices[0][1]) / (vertices[0][1] -
    vertices[1][1]) +
vertices[0][
0]) and ((vertices[1][1] - vertices[3][1]) * (x - vertices[1][0]) / (
vertices[1][0] - vertices[3][0]) +
vertices[1][1]) <= y <= ((vertices[0][1] - vertices[2][1]) * (x - vertices[0][0]) / (
vertices[0][0] - vertices[2][0]) + vertices[0][1]):
return 1
else:
return 0


def has_intersection_with_cylinder(line_start, line_direction, cylinder_center,
    cylinder_radius, cylinder_height):
# 计算直线与圆柱面的交点
a = line_direction[0] ** 2 + line_direction[1] ** 2
b = 2 * line_direction[0] * (line_direction[2] * line_start[0] - line_direction[0] *
    line_start[2]) + 2 * \
line_direction[1] * (line_direction[2] * line_start[1] - line_direction[1] *
    line_start[2])
c = (line_direction[2] * line_start[0] - line_direction[0] * line_start[2]) ** 2 + (
```

```
    line_direction[2] * line_start[1] - line_direction[1] * line_start[2]) ** 2 - (
    3.5 * line_direction[2]) ** 2

    # 计算判别式
    discriminant = b ** 2 - 4 * a * c

    if discriminant >= 0:
    z1 = (-b + np.sqrt(discriminant)) / (2 * a)
    z2 = (-b - np.sqrt(discriminant)) / (2 * a)

    # 检查交点是否在圆柱体的高度范围内
    if (cylinder_center[2] <= z1 <= cylinder_center[2] + cylinder_height) or \
    (cylinder_center[2] <= z2 <= cylinder_center[2] + cylinder_height):
    return 1

    return 0


    def Average_annual_power_output_per_mirror(r_s, a_s, point_s, a1, b1, y, h): #
        用于计算每个镜子的年平均输出功率
    point = [(a1 / 2, b1 / 2, 0), (a1 / 2, -b1 / 2, 0), (-a1 / 2, b1 / 2, 0), (-a1 / 2, -b1 /
        2, 0)]
    i_s = np.array([math.cos(a_s) * math.cos(r_s), math.cos(a_s) * math.sin(r_s),
        math.sin(a_s)]) # 太阳光线向量
    p_jieduan = [] # 截断效率
    p_shadow = [] # 阴影效率
    p_yuxian = [] # 余弦效率
    for i in range(len(point_s)):
    matrix = np.zeros((30, 30))
    i_reflected_light = np.array([-point_s[i][0], y - point_s[i][1], 80 - h]) #
        每一块采光镜的反射的光线的向量并单位化
    i_reflected_light_1 = i_reflected_light / np.linalg.norm(i_reflected_light) # 单位化的结果
    z_a, x_a, y_a = Coordinate_system_transformation2(a_s, r_s, point_s[i], y, h) #
        首先得到a采光镜的xyz坐标
    j_reflected_light_a = (point_s[i][0], point_s[i][1], h) # a采光镜的位置坐标
    cos_yuxian = np.dot(i_reflected_light_1, z_a)
    yuxian = math.acos(cos_yuxian)
    p_yuxian.append(math.cos(yuxian))
    # print(Coordinate_system_transformation(a_s, r_s, i))
    #  print(i)
    # 创建二维KD树
    tree = cKDTree(point_s)

    # 要查找附近的点的目标点
    target_point = (point_s[i][0], point_s[i][1])

    # 搜索最近的k个点
```

```
if len(point_s) >= 4:
k = 4 # 你可以根据需要修改k的值
else:
k = len(point_s)
nearest_point_indices = tree.query(target_point, k=k)[1]
nearest_points = [point_s[j] for j in nearest_point_indices]
for pp in range(len(nearest_points)):
# for points in nearest_points:
if nearest_points[pp] != target_point:
points = nearest_points[pp]
xy_shadow = []
xy_block = []
# if x[j] == x[i] and y[j] == y[i]:

for p in point:
j_reflected_light_b = (points[0], points[1], h) # b采光镜的位置坐标
z_b, x_b, y_b = Coordinate_system_transformation2(a_s, r_s, points, y, h) #
    首先得到b(干扰镜）采光镜的xyz坐标
T_b = np.array([ # b镜的转换矩阵
[x_b[0], y_b[0], z_b[0]],
[x_b[1], y_b[1], z_b[1]],
[x_b[2], y_b[2], z_b[2]]
])
T_a = np.array([ # a镜的转换矩阵
[x_a[0], y_a[0], z_a[0]],
[x_a[1], y_a[1], z_a[1]],
[x_a[2], y_a[2], z_a[2]]
])
i_s_h = np.dot(T_a.T, i_s)
i_reflected_light_1_h = np.dot(T_a.T, i_reflected_light_1)
H_b = p # 需要进行转换的坐标
H_b1 = np.dot(T_b, H_b) + j_reflected_light_b # 先转换到地面坐标系上
H_b2 = np.dot(T_a.T, H_b1 - j_reflected_light_a) # 转换到a镜坐标系
```