基于蒙特卡洛和粒子群遗传算法的定日镜场的优化设计

摘要

塔式太阳能光热发电是一种低碳环保的新型清洁能源技术，通过定日镜将太阳光反射汇聚到吸收塔顶端的集热器，加热其中的导热介质，从而实现太阳能到热能到电能的转化输出。定日镜场对太阳光的反射汇聚是一个复杂的过程，受到定日镜尺寸、位置、数目等多种因素的影响，并伴随着各种光学效率的损失。实现定日镜场的优化设计，对于构建以新能源为主体的新型电力系统，实现“碳达峰”“碳中和”目标具有重要意义。

针对问题一：我们构建了太阳方位确定模型，通过计算不同时刻的太阳方位角和太阳高度角可以确定太阳入射光线的方向向量，从而根据定日镜追踪太阳位置的原理确定定日镜的镜面法向量。为了计算定日镜的阴影遮挡效率与集热器截断效率，我们采用光线追踪法（均匀采样的蒙特卡洛法）进行建模，并构建了相应的判别器来判断选取得随机光线是否会被阴影遮挡/集热器截断。用未被遮挡/截断的光线数目除以总光线数目可得阴影遮挡效率和集热器截断效率，综合运算后得到各定日镜的光学效率。再结合题目附录中的公式求得一年中不同时刻的功率数据（12*5=60 组），先将每日不同时间的数据取平均，再将每年不同日期的数据取平均，得到所求的年平均功率数据如表 3, 表 4 所示。

针对问题二：考虑到参数数目过多会导致运算量过大，我们先采用了 Campo 方法布置初始镜场简化参数，再使用 PSO-GA（粒子群遗传算法）进行优化。考虑到镜子与塔的距离限制，我们将 Campo 初始镜场设置为 4 个紧密嵌套的同心圆环区域。所有区域内的定日镜尺寸、高度均相同，同一区域各行上的定日镜数量相同，定日镜之间最密集排布。继而使用遗传算法与粒子群优化相结合的 PSO-GA 算法进行优化，优化目标为：单位镜面面积年平均输出热功率尽量大；优化限制为：年平均输出热功率大于等于 $60MW$，$100m \leq$ 镜与塔的距离 $\leq 350m$。对于 PSO-GA 优化后的结果，按照光学效率从低到高的顺序分批剔除镜点，直至镜场的年平均输出功率大于等于 $60MW$ ，且继续剔除一个镜点会使镜场年平均输出功率小于 $60MW$ 为止。剔除处理后得到的定日镜和集热塔坐标储存在附件 result2.xlsx 中，计算整理得优化镜场参数如表 6, 表 7, 表 8 所示。

针对问题三：相较于问题二，问题三引入了更多可以优化的参数，但同样适用问题二的 Campo 初始镜场布置-PSO-GA 优化-剔除低效镜点模型。将初始镜场设置为 4 个紧密嵌套的同心圆环区域，同一区域内的定日镜尺寸、高度相同，不同区域内的定日镜尺寸、高度可以不同，同一区域各行上的定日镜数量相同，定日镜之间最密集排布。对简化后的镜场参数采用 PSO-GA 混合算法进行优化并分批分批剔除镜点，优化目标、优化限制以及低效镜点提出原则与问题二相同。处理后得到的定日镜和集热塔尺寸及坐标储存在附件 result3.xlsx 中，计算整理得优化镜场参数如表 10, 表 11, 表 12 所示。

最后，我们对问题 1 中结论数据进行了多项式插值，观察得到的“日期-时间-参量”曲面图。曲面的变化较为平缓，说明光线追迹的蒙特卡洛算法灵敏度较低，模型稳定性很好。

关键词：光线追迹；蒙特卡洛；Campo 布置；PSO-GA 算法

1 问题重述

1.1 问题背景

塔式太阳能光热发电是一种低碳环保的新型清洁能源技术，对于构建以新能源为主体的新型电力系统，实现“碳达峰”“碳中和”目标具有重要意义。定日镜是塔式电站收集太阳能的基本组件，其底座由纵向转轴和水平转轴组成，平面反射镜安装在水平转轴上（图 1）。纵向转轴的轴线与地面垂直，控制反射镜的方位角；水平转轴的轴线与地面平行，控制反射镜的俯仰角。塔式电站利用大量的定日镜组成阵列，称为定日镜场。定日镜将太阳光反射汇聚到安装在镜场中吸收塔顶端上的集热器，加热其中的导热介质，并将太阳能以热能形式储存，再经过热交换实现由热能向电能的转化。定日镜在工作时，控制系统根据太阳的位置实时控制定日镜的法向，使得太阳中心点发出的光线经定日镜中心反射后指向集热器中心。

图 1: 定日镜及底座示意图

图 2: 圆形定日镜场示意图

1.2 目标任务

现计划在中心位于东经 98.5°，北纬 39.4°，海拔 3000m，半径 350m 的圆形区域内建设一个圆形定日镜场（图 2）。以圆形区域中心为原点，正东方向为 x 轴正向，正北方向为 y 轴正向，垂直于地面向上方向为 z 轴正向建立坐标系，称为镜场坐标系。规划的吸收塔高度为 80m，圆柱形外表受光式集热器高 8m，直径 7m，平面矩形定日镜边长在 2m 到 8m 之间，安装高度在 2m 至 6m 之间。安装要求吸收塔周围 100m 范围内不安装定日镜，相邻定日镜底座中心之间的距离比镜面宽度多 5m 以上。为简化计算，本问题中所有“年均”指标的计算时点均为当地时间每月 21 日 9:00、10:30、12:00、13:30、15:00，据此建立模型并解决以下问题：

问题 1：给定吸收塔与定日镜位置如附件所示。定日镜尺寸 $6m \times 6m$，安装高度 4m。计算该定日镜场的年平均光学效率、年平均输出热功率，以及单位镜面面积年平均输出热功率。

问题 2：若所有定日镜尺寸及安装高度相同，设计定日镜的尺寸、安装高度、数目，以及吸收塔与定日镜的位置坐标参数，使得定日镜场达到额定年平均输出功率 60MW 的条件下，单位镜面面积年平均输出热功率尽量大。

问题 3：若定日镜尺寸及安装高度可以不同，设计定日镜场的各个参数，使得定日镜场达到额定年平均输出功率 60MW 的条件下，单位镜面面积年平均输出热功率尽量大。

2 问题分析

2.1 问题一的分析

问题一给定了定日镜场的各个参数，要求计算该镜场的年平均光学效率、年平均输出热功率和单位镜面面积年平均输出热功率。首先我们求解了一年中不同时刻的太阳方位角和太阳高度角，确定太阳入射光线的方向向量，从而根据定日镜追踪太阳位置的原理确定定日镜的镜面法向量。由于定日镜的阴影遮挡现象和集热器截断现象是两个复杂的过程，我们采用了光线追迹法（均匀采样的蒙特卡洛法）进行建模，并构建了相应的判别器来判断选取得随机光线是否会被阴影遮挡/集热器截断。用未被遮挡/截断的光线数目除以总光线数目可得阴影遮挡效率和集热器截断效率，综合运算后得到各定日镜的光学效率。再结合题目附录中的公式求得一年中不同时刻的功率数据（12*5=60 组），先将每日不同时间的数据取平均，再将每年不同日期的数据取平均，得到所求的年平均功率数据。

2.2 问题二的分析

问题二要在定日镜尺寸和安装高度相同的前提下设计镜场参数，使得镜场达到额定功率且单位镜面面积年平均输出热功率尽量大。考虑到参数数目过多会导致运算量爆炸，我们采用 Campo 布置方法 [**?**] 对参数进行了简化：将初始镜场设置为 4 个紧密嵌套的同心圆环区域，所有区域内的定日镜尺寸、高度均相同，同一区域各行上的定日镜数量相同，定日镜之间最密集排布。在此基础上，我们对简化后的镜场参数采用 PSO-GA 混合算法进行优化。综合了遗传算法 (GA) 和粒子群优化 (PSO) 的优点，PSO-GA 具有更好的综合表现。[**?**] 优化目标为：单位镜面面积年平均输出热功率尽量大；优化限制为：年平均输出热功率大于等于 $60MW$，$100m \leq$ 镜与塔的距离 $\leq 350m$。考虑到圆环形的初始镜场中极有可能存在低效率的镜点，拉低了镜场的单位镜面面积年平均输出热功率。因此对于 PSO-GA 优化后的结果，按照光学效率从低到高的顺序分批剔除镜点，直至镜场的年平均输出功率大于等于 $60MW$ ，且继续剔除一个镜点会使镜场年平均输出功率小于 $60MW$ 为止。剔除处理后得到的数据即为所求的优化镜场参数。

2.3 问题三的分析

问题三要在定日镜尺寸和安装高度可以不同的条件下设计镜场参数，使得镜场达到额定功率且单位镜面面积年平均输出热功率尽量大。相较于问题二，问题三引入了更多可以优化的参数，但同样适用问题二的 Campo 初始镜场布置-PSO-GA 优化-剔除低效镜点模型。将初始镜场设置为 4 个紧密嵌套的同心圆环区域，同一区域内的定日镜尺寸、高度相同，不同区域内的定日镜尺寸、高度可以不同，同一区域各行上的定日镜数量相同，定日镜之间最密集排布。对简化后的镜场参数采用 PSO-GA 混合算法进行优化并分批分批剔除镜点，优化目标、优化限制以及低效镜点提出原则与问题二相同。处理后得到的数据即为所求的优化镜场参数。

3 模型假设

1. 在定日镜场内，不考虑太阳光入射角和方位角的微小变化。
2. 考虑阴影遮挡效率时，集热塔的阴影近似为矩形。
3. 考虑阴影遮挡效率时，太阳光看做一族平行的光线入射且按照入射方向均匀分布。
4. 考虑集热器截断效率时，不再考虑其他镜面对太阳光锥的阻挡。
5. 考虑集热器截断效率时，假设太阳光锥光能按照半角展宽均匀分布。
6. 考虑问题二、三时，假设按照 Campo 布置镜场的单位面积年平均输出热功率最大 [**?**]。

4 符号说明

符号	解释	单位
D	以春分作为第 0 天起算的天数	-
ST	当地时间（24 小时制）	-
φ	当地纬度	°
H	海拔高度	km
ω	太阳时角	rad
α_s	太阳高度角	°
γ_s	太阳方位角	°
δ	太阳赤纬角	rad
η_{cos}	余弦效率	-
η_{at}	大气透射率	-
η_{sb}	阴影遮挡效率	-
η_{trunc}	集热器截断效率	-
η_{ref}	镜面反射率（0.92）	-
G_0	太阳常数（1.366）	kW/m^2
DNI	法向直接辐射辐照度	kW/m^2
η_i	第 i 面镜子的光学效率	-
A_i	第 i 面定日镜采光面积	m^2
E_{field}	定日镜场的输出热功率	kW
$\boldsymbol{n_i}$	第 i 面镜子的法向量	-
d_{HR}	镜面中心到集热器中心的距离	m
$\boldsymbol{s_i}$	第 i 面镜子入射光线的方向向量	-
$\boldsymbol{t_i}$	第 i 面镜子反射光线的方向向量	-
$\boldsymbol{p_{ij}}$	第 i 面镜子上选取的第 j 个点的坐标向量	-

表 1: 符号说明（注：未申明的变量以其在出现处的具体说明为准）

5 问题一模型的建立与求解

5.1 问题一模型的建立

定日镜是用来实时追踪太阳位置并将太阳光准确地反射到集热器的光学装置。欲计算该定日镜场的年平均光学效率，首先应确定一年不同时刻太阳的方位，即太阳高度角和太阳方位角，根据太阳角度运算得到不同位置的定日镜的光学效率、采光面积等数据，从而计算该定日场年平均光学效率、年平均输出热功率，以及单位镜面面积年平均输出热功率。

5.1.1 太阳方位的确定

在镜场坐标系的基础上考虑太阳的运动，将太阳与坐标系原点之间的连线与地平面之间的夹角称为太阳高度角，用 α_s 表示；将太阳与坐标系原点之间的连线在基础平面上的投影从北方沿着地平线顺时针量度得到的角称为太阳方位角，用 γ_s 表示（图 3, 图 4）。任一时刻太阳的方位可由太阳高度角 α_s 和太阳方位角 γ_s 这两个参数来描述。

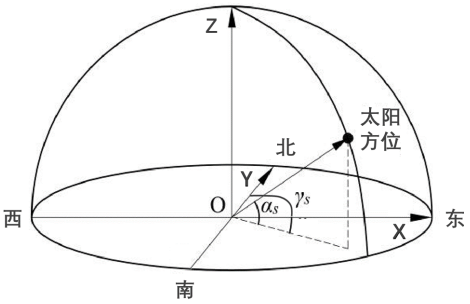

图 3: 问题 *1* 太阳方位坐标系示意图

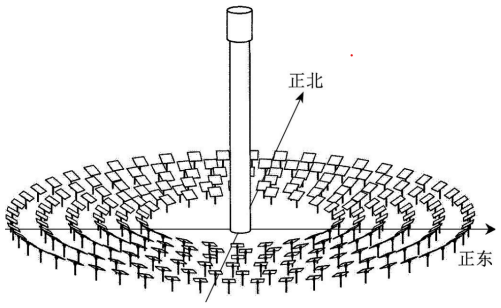

图 4: 问题 *1* 定日镜场坐标系示意图

记当地纬度 $\varphi = 39.4°$，当地时间 ST 可取 $9, 10.5, 12, 13.5, 15$。用 D 表示以春分（3 月 21 日）作为第 0 天起算的天数，即 1-12 月的 21 日分别对应 $D = 0, 31, 61, 92, 122, 153, 184, 214, 245, 275, 306, 337$。由题目给出的公式可知：

$$
\begin{aligned}
&\text{太阳赤纬角 } \delta\text{：}\ \sin\delta = \sin\frac{2\pi D}{365}\sin\left(\frac{2\pi}{360}23.45\right)\\
&\text{太阳时角 } \omega\text{：}\ \omega = \frac{\pi}{12}(ST-12)\\
&\text{太阳高度角 } \alpha_s\text{：}\ \sin\alpha_s = \cos\delta\cos\varphi\cos\omega + \sin\delta\sin\varphi\\
&\text{太阳方位角 } \gamma_s\text{：}\ \cos\gamma_s = \frac{\sin\delta - \sin\alpha_s\sin\varphi}{\cos\alpha_s\cos\varphi}.
\end{aligned}
\tag{5.1.1}
$$

对得到的 $\sin\alpha_s$ 与 $\cos\gamma_s$ 值取反三角函数，整理得 60 个时刻的太阳高度角 α_s 和太阳方位角 γ_s 数据如下表所示：

(α_s, γ_s)	9:00	10:30	12:00	13:30	15:00
1 月 21 日	(17.43,135.78)	(27.21,156.11)	(30.83,180.0)	(27.21,203.89)	(17.43,224.22)
2 月 21 日	(24.83,130.02)	(35.77,152.38)	(39.97,180.0)	(35.77,207.62)	(24.83,229.98)
3 月 21 日	(33.12,122.4)	(45.55,146.87)	(50.6,180.0)	(45.55,213.13)	(33.12,237.6)
4 月 21 日	(41.57,112.24)	(55.85,138.11)	(62.28,180.0)	(55.85,221.89)	(41.57,247.76)
5 月 21 日	(47.05,103.1)	(62.77,128.3)	(70.79,180.0)	(62.77,231.7)	(47.05,256.9)
6 月 21 日	(48.93,99.13)	(65.17,123.29)	(74.05,180.0)	(65.17,236.71)	(48.93,260.87)
7 月 21 日	(46.99,103.23)	(62.69,128.44)	(70.69,180.0)	(62.69,231.56)	(46.99,256.77)
8 月 21 日	(41.22,112.74)	(55.41,138.6)	(61.76,180.0)	(55.41,221.4)	(41.22,247.26)
9 月 21 日	(32.67,122.86)	(45.02,147.22)	(50.01,180.0)	(45.02,212.78)	(32.67,237.14)
10 月 21 日	(23.85,130.83)	(34.63,152.92)	(38.75,180.0)	(34.63,207.08)	(23.85,229.17)
11 月 21 日	(16.83,136.21)	(26.51,156.39)	(30.1,180.0)	(26.51,203.61)	(16.83,223.79)
12 月 21 日	(14.4,137.95)	(23.73,157.45)	(27.16,180.0)	(23.73,202.55)	(14.4,222.05)

表 2: 问题 *1* 不同时刻 α_s 与 γ_s 数据表

5.1.2 镜面法向的确定

考虑确定不同时刻第 i 个定日镜的镜面法向量 $\boldsymbol{n_i}$。由于同一时刻所有定日镜接受到的太阳高度角 α_s 和太阳方位角 γ_s 均相同，故每一个定日镜上的入射光线的方向向量 $\boldsymbol{s_i}$ 相同，即：

$$\boldsymbol{s_i} = \frac{1}{\sqrt{\cos^2\alpha_s \sin^2\gamma_s + \cos^2\alpha_s \cos^2\gamma_s + \sin^2\alpha_s}} (\cos\alpha_s \sin\gamma_s, -\cos\alpha_s \cos\gamma_s, -\sin\alpha_s) \tag{5.1.2}$$

由于定日镜可以实时追踪太阳位置并将太阳光准确地反射到集热器，记第 i 个定日镜的镜面中心坐标为 (x_i, y_i, z_i)，集热器的镜面中心坐标为 $(0, 0, H)$，则反射光的方向向量 $\boldsymbol{t_i}$ 为：

$$\boldsymbol{t_i} = \frac{1}{\sqrt{(x_i^2 + y_i^2 + (z_i - H)^2)}} (-x_i, -y_i, H - z_i) \tag{5.1.3}$$

将 $\boldsymbol{t_i}$ 与 $\boldsymbol{s_i}$ 相减并归一化，得到第 i 个定日镜的镜面法向量 $\boldsymbol{n_i}$（图 5）：

$$\boldsymbol{n_i} = \frac{1}{|\boldsymbol{t_i} - \boldsymbol{s_i}|} (\boldsymbol{t_i} - \boldsymbol{s_i}) \tag{5.1.4}$$

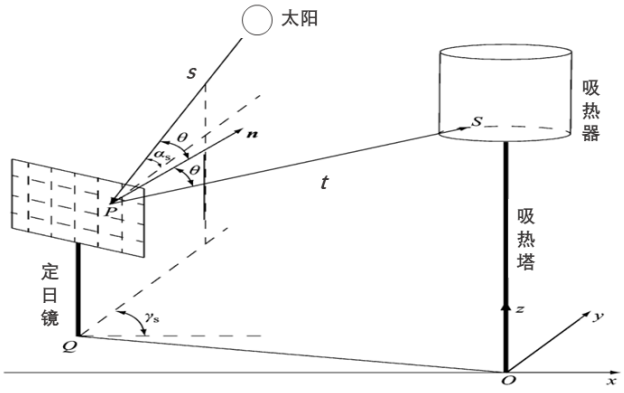

图 5: 问题 *1* 太阳光线追迹示意图

5.1.3 阴影遮挡效率

阴影遮挡损失包括三部分 [3]:1. 塔对镜场造成的阴影损失。2. 后排定日镜接收的太阳光被前方定日镜所阻挡造成的阴影损失。3. 后排定日镜在反射太阳光时被前方定日镜阻挡而未到达集热器上造成的遮挡损失。本文中我们将其分为两大类：塔造成的阴影遮挡损失 1 和定日镜造成的阴影遮挡损失 2,3。

我们采用蒙特卡洛法来计算阴影遮挡效率。对于某个固定的时刻，在每个定日镜上均匀取点，输入两个判断器分别进行以下判断：(1) 通过该点的入射光线或反射光线是否会被定日镜遮挡，是输出 0，否输出 1。(2) 该点是否被塔的阴影覆盖，是输出 0，否输出 1；将两个判断器的输出结果相乘得到综合判断结果，若为 1 则说明该点不会被阴影遮挡，若为 0 则说明该点会被遮挡。将所有点的综合判断结果相加并除以总点数，得到定日镜阴影遮挡效率 (图 6)。

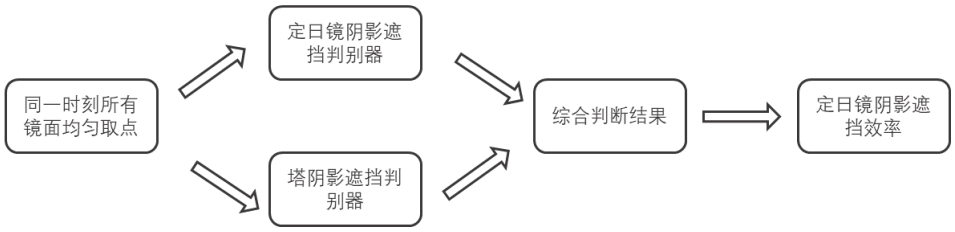

图 6: 问题 *1* 定日场阴影遮挡效率计算流程图

(1-1) 定日镜阴影遮挡建模 对于第 i 个定日镜的镜平面，以镜平面的中心为原点，以平行于镜面高的方向为 r 轴，向上为正方向，记单位向量为 $\boldsymbol{r_i}$；以平行于镜面宽的方向为 l 轴，向右为正方向，记单位向量为 $\boldsymbol{l_i}$（图 7)。设第 i 个定日镜镜面中心坐标向量为 $\boldsymbol{m_i} = (x_i, y_i, z_i)$，镜面法向量为 $\boldsymbol{n_i} = (a_i, b_i, c_i)$，则可解 $\boldsymbol{l_i}, \boldsymbol{r_i}$ 得：

$$\begin{aligned} \boldsymbol{l_i} &= \frac{1}{\sqrt{a_i^2 + b_i^2}} (b_i, -ai, 0) \\ \boldsymbol{r_i} &= \frac{1}{\sqrt{a_i^2 c_i^2 + c_i^2 b_i^2 + (a_i^2 + b_i^2)^2}} \left(a_i c_i, c_i b_i, -a_i^2 - b_i^2\right) \end{aligned} \tag{5.1.5}$$

在每个定日镜上均匀取点。对于第 i 个定日镜的 $l-r$ 坐标系，该镜面上第 j 个点的坐标向量记为 $\boldsymbol{p_{ij}} = (l_{ij}, r_{ij})$。转换为定日场坐标系，该点的坐标向量为 $\boldsymbol{p_{ij}} = \boldsymbol{m_i} + l_{ij}\boldsymbol{l_i} + r_{ij}\boldsymbol{r_i}$。

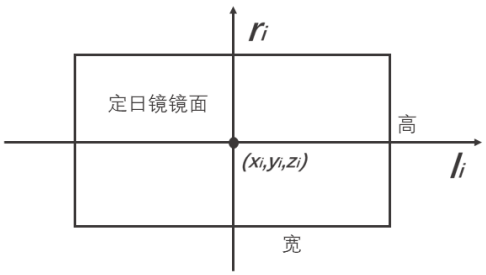

图 7: 问题 *1* 定日镜面 $l-r$ 坐标系示意图

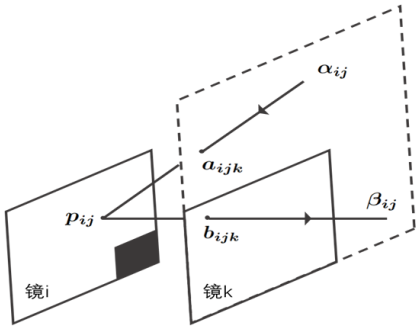

图 8: 问题 *1* 入射反射光线与镜平面交点示意图

已知第 i 个定日镜上入射光线与反射光线的方向向量分别为 $\boldsymbol{s_i}, \boldsymbol{t_i}$，设 t 为参数，则经过第 i 面定日镜上第 j 个点的入射光线向量 $\boldsymbol{\alpha_{ij}}$ 与反射光线向量 $\boldsymbol{\beta_{ij}}$ 可以参数化地表示为：

$$\begin{aligned} \boldsymbol{\alpha_{ij}} &= \boldsymbol{p_{ij}} + t\,\boldsymbol{s_i} \\ \boldsymbol{\beta_{ij}} &= \boldsymbol{p_{ij}} + t\,\boldsymbol{t_i} \end{aligned} \tag{5.1.6}$$

继而考虑第 i 个定日镜旁边的第 k 个定日镜 $(i \neq k)$，记第 k 个镜平面上任意一点的坐标向量为 $\boldsymbol{p}$，则第 k 个镜平面的镜面方程可表示为：$\boldsymbol{n_k} \cdot (\boldsymbol{p} - \boldsymbol{m_k}) = 0$ 。想要确定第 i 个镜平面上第 j 个点的入射光线或反射光线是否会与第 k 个镜平面相交，只需求解 $\boldsymbol{\alpha_{ij}}, \boldsymbol{\beta_{ij}}$ 和无限大镜平面 $\boldsymbol{n_k} \cdot (\boldsymbol{p} - \boldsymbol{m_k}) = 0$ 的交点坐标，判断交点是否在第 k 面镜子的范围内即可（图 8）。

记第 i 个镜平面上第 j 个点的入射光线与第 k 个无限大的镜平面的交点坐标向量为 $\boldsymbol{a_{ijk}}$，记第 i 个镜平面上第 j 个点的反射光线与第 k 个无限大的镜平面的交点坐标向量为 $\boldsymbol{b_{ijk}}$。联立解得 $\boldsymbol{a_{ijk}}, \boldsymbol{b_{ijk}}$ 如下：

$$\begin{cases} \boldsymbol{n_k} \cdot (\boldsymbol{p_{ij}} + t\,\boldsymbol{s_i} - \boldsymbol{m_k}) = 0 \\ \boldsymbol{a_{ijk}} = \boldsymbol{p_{ij}} + t\,\boldsymbol{s_i} \end{cases} \Rightarrow \quad \boldsymbol{a_{ijk}} = \boldsymbol{p_{ij}} + \frac{\boldsymbol{n_k} \cdot (\boldsymbol{m_k} - \boldsymbol{p_{ij}})}{\boldsymbol{n_k} \cdot \boldsymbol{s_i}} \boldsymbol{s_i} \tag{5.1.7}$$

$$\begin{cases} \boldsymbol{n_k} \cdot (\boldsymbol{p_{ij}} + t\,\boldsymbol{t_i} - \boldsymbol{m_k}) = 0 \\ \boldsymbol{b_{ijk}} = \boldsymbol{p_{ij}} + t\,\boldsymbol{t_i} \end{cases} \Rightarrow \quad \boldsymbol{b_{ijk}} = \boldsymbol{p_{ij}} + \frac{\boldsymbol{n_k} \cdot (\boldsymbol{m_k} - \boldsymbol{p_{ij}})}{\boldsymbol{n_k} \cdot \boldsymbol{s_i}} \boldsymbol{t_i} \tag{5.1.8}$$

(1-2) 定日镜阴影遮挡判别器 对每一组 i, k，构建判别器判断 $\boldsymbol{a_{ijk}}, \boldsymbol{b_{ijk}}$ 是否在第 k 面定日镜的镜子范围内。具体判别流程如下（图 9）：

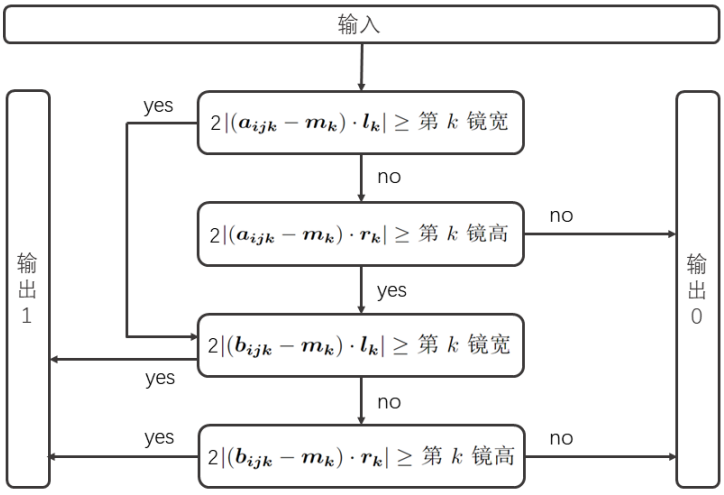

图 9: 问题 *1* 定日镜阴影遮挡判别器原理示意图

(2-1) 塔阴影遮挡建模 由题图可知，塔可视作半径为 $\frac{7}{2}m$，高为 $84m$ 的圆柱体。将塔的阴影近似为矩形，以镜场坐标系的中心为原点，以太阳光在地平面上的投影方向为 r 轴，影子方向为正方向，记单位向量为 $\boldsymbol{r}_{\text{阴}}$；以垂直于 r 轴的方向为 l 轴，r 轴顺时针 $90°$ 方向为正方向，记单位向量为 $\boldsymbol{l}_{\text{阴}}$；以竖直方向为 h 轴，向上为正方向，记单位向量为 $\boldsymbol{h}$（图 10,11）。

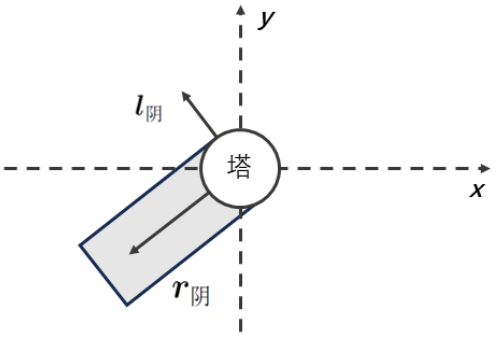

图 10: 问题 *1* 塔 $l-r$ 坐标系俯视图

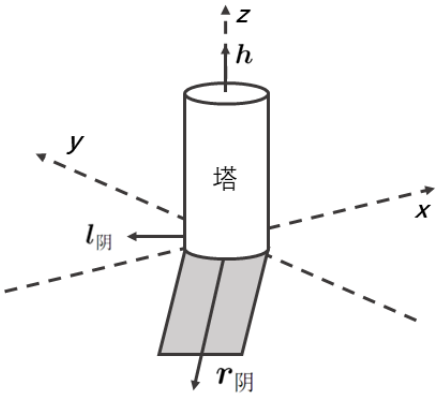

图 11: 问题 *1* 塔 $l-r-h$ 坐标系示意图

分析几何关系，易知方向向量 $\boldsymbol{r}_{阴}, \boldsymbol{l}_{阴}$ 仅与太阳方位角 γ_s 有关：

$$\begin{cases} \boldsymbol{r}_{阴} = (-\sin\gamma_s, -\cos\gamma_s, 0) \\ \boldsymbol{l}_{阴} = (-\cos\gamma_s, \sin\gamma_s, 0) \\ \boldsymbol{h} = (0,0,1) \end{cases} \tag{5.1.9}$$

若 $0 < \boldsymbol{p_{ij}} \cdot \boldsymbol{r}_{阴} \le \frac{84}{\tan\alpha_s}$ 且 $|\boldsymbol{p_{ij}} \cdot \boldsymbol{l}_{阴}| \le \frac{7}{2}$ 且 $\frac{\boldsymbol{p_{ij}} \cdot \boldsymbol{h}}{\frac{84}{\tan\alpha_s} - \boldsymbol{p_{ij}} \cdot \boldsymbol{r}_{阴}} \le \tan\alpha_s$，则点 $\boldsymbol{p_{ij}}$ 在塔的阴影中。

(2-2) 塔阴影遮挡判别器 对每一时刻的 $\boldsymbol{r}_{阴}, \boldsymbol{l}_{阴}, \boldsymbol{h}$，构建判别器判断 $\boldsymbol{p_{ij}}$ 是否会被塔的阴影覆盖。具体判别流程如下（图 12）

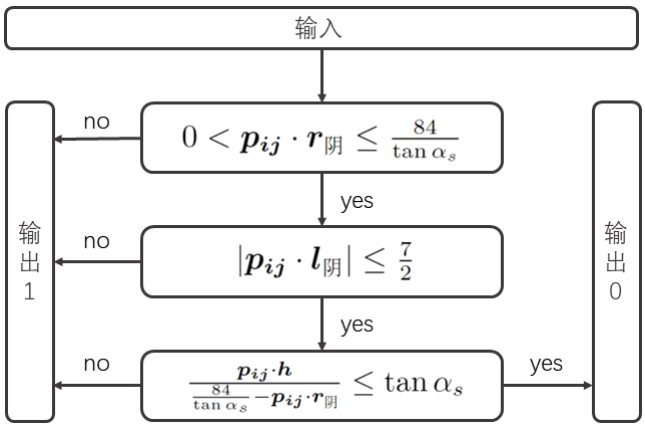

图 12: 问题 *1* 塔阴影遮挡判别器原理示意图

(3) 综合判断结果 将两个判断器的输出结果相乘得到综合判断结果，若为 1 则说明该点不会被阴影遮挡，若为 0 则说明该点会被遮挡。将一面镜子上所有点的综合判断结果相加并除以总点数，得到该定日镜的阴影遮挡效率 η_{sb}（每面镜子上取点个数记为 M）：

$$\eta_{sb} = \frac{1}{M}\sum_{j=1}^{M}(\text{定日镜阴影遮挡判别器输出}) \cdot (\text{塔阴影遮挡判别器输出}). \tag{5.1.10}$$

5.1.4 集热器截断效率

来自太阳的入射光线是一束锥形光线，其半角展宽为 $4.65mrad$[3]，根据反射原理，反射到吸热器上的光线也是呈发散的锥形形状。目标点和反射镜的距离越远，光锥的截面越大。由于镜场中定日镜到吸热塔的距离大小不一致，吸热器上的反射光斑尺寸会根据定日镜与吸热器的距离而有所变化。此外，定日镜在制造过程中也存在一定的跟踪精度以及面型精度。跟踪精度的存在会影响反射光斑是否精准地投射到目标点上，而定日镜面型误差的存在会使反射光斑的形状不再具有规则性。[3] 以上种种因素使镜场反射光斑并不能全部落入吸热器接收范围内，而造成了一定的光溢出现象。故考虑集热器的截断效率 η_{trunc}：

$$\eta_{trunc} = \frac{\text{集热器接收能量}}{\text{镜面全反射能量} - \text{阴影遮挡损失能量}}. \tag{5.1.11}$$

由于反射光锥在集热器上的投影形状不规则，我们采用光线追迹法。追迹一束太阳光中任意一条光线在经过定日镜反射后落入集热器的位置，用集热器接受的光线数目除以反射光线的数目，即可得到集热器截断效率 η_{trunc}。

(1) 反射光线建模 对于第 i 个定日镜上均匀取得的第 j 个点 $\boldsymbol{p_{ij}}$（非阴影遮挡区域），反射光线呈锥形，半角展宽 $\theta = 4.65mrad$。设反射光线方向向量 $\boldsymbol{t_i} = (a_i, b_i, c_i)$，则可在镜场坐标系中找到另外两个方向向量 $\boldsymbol{l_i}, \boldsymbol{r_i}$，满足 $\boldsymbol{l_i}, \boldsymbol{r_i}, \boldsymbol{t_i}$ 互相垂直（图 13），即：

$$\begin{aligned} \boldsymbol{l_i} &= \frac{1}{\sqrt{b_i^2 + c_i^2}} (0, c_i, -b_i) \\ \boldsymbol{r_i} &= \frac{1}{\sqrt{a_i^2 b_i^2 + a_i^2 c_i^2 + (b_i^2 + c_i^2)^2}} \left(-b_i^2 - c_i^2, a_i b_i, a_i c_i, \right) \end{aligned} \tag{5.1.12}$$

在第 i 个定日镜上第 j 个点的反射光锥中随机取光线。设取得的第 k 个光线的方向向量为 $\boldsymbol{t_{ijk}}$，$\boldsymbol{t_{ijk}}$ 与 $\boldsymbol{t_s}$ 的夹角记为 α，$\boldsymbol{t_{ijk}}$ 在光锥切面圆上的投影与 $\boldsymbol{l_i}$ 的夹角记为 β。令随机反射光线的取法满足 $0 \le \alpha \le \theta, 0 \le \beta \le 2\pi$ 的均匀分布（图 14）。

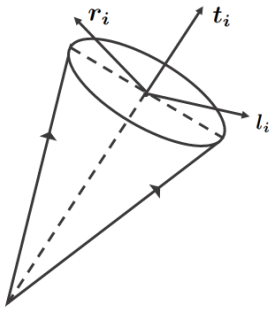

图 13: 问题 *1* 反射光锥 $l-r-t$ 坐标系示意图

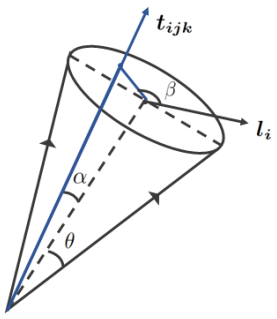

图 14: 问题 *1* 随机反射光线示意图

由几何关系可得，随机反射光线的方向向量 $\boldsymbol{t_{ijk}} = \boldsymbol{t_i} + \tan\alpha\,(\cos\beta\,\boldsymbol{l_i} + \sin\beta\,\boldsymbol{r_i})$，则方向为 $\boldsymbol{t_{ijk}}$ 的随机反射光线向量 $\boldsymbol{\gamma_{ijk}}$ 可以参数化地表示为：$\boldsymbol{\gamma_{ijk}} = \boldsymbol{p_{ij}} + t\,\boldsymbol{t_{ijk}}$。

(2) 光线追迹判别器 对每一组 i,j,k，构建判别器判断 $\boldsymbol{\gamma_{ijk}}$ 与集热器曲面外侧是否有交点。在镜场坐标系中，集热器的曲面方程为：

$$\begin{cases} x^2+y^2=(\frac{7}{2})^2 \\ 76\le z\le 84 \end{cases} \tag{5.1.13}$$

记 $\boldsymbol{\gamma_{ijk}}=(x,y,z),\boldsymbol{p_{ij}}=(x_{ij},y_{ij},z_{ij}),\boldsymbol{t_{ijk}}=(x_{ijk},y_{ijk},z_{ijk})$，由 $\boldsymbol{\gamma_{ijk}}=\boldsymbol{p_{ij}}+t\,\boldsymbol{t_{ijk}}$ 可得：

$$\begin{cases} x=x_{ij}+t\,x_{ijk} \\ y=y_{ij}+t\,y_{ijk} \\ z=z_{ij}+t\,z_{ijk} \end{cases} \tag{5.1.14}$$

其中 t 为参数。对于某一确定时刻，$x_{ij},y_{ij},z_{ij},x_{ijk},y_{ijk},z_{ijk}$ 均为确定常数。由于 (5.1.13) 对 z 的范围有所限制，先仅考虑变量 x,y。将 (5.1.14) 中 x,y 的表达式代入 (5.1.13) 中，得到关于参数 t 的一元二次方程：

$$(x_{ij}+t\,x_{ijk})^2+(y_{ij}+t\,y_{ijk})^2=(\frac{7}{2})^2 \tag{5.1.15}$$

针对方程 (5.1.15)，构建判别流程如下（图 15）：

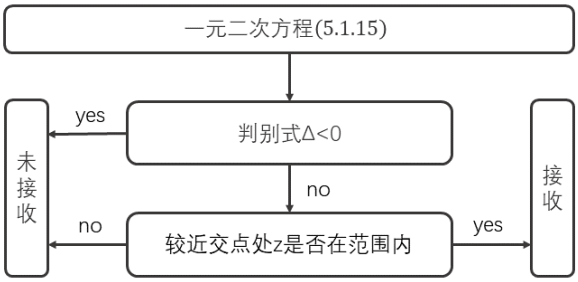

图 15: 问题 *1* 光线追踪判别器原理示意图

(3) 截断效率计算 我们追迹了未被阴影遮挡处的随机反射光线落入集热器的位置，用集热器接受的光线数目除以随机反射光线的总数目，即可得到集热器截断效率 η_{trunc}。

$$\eta_{trunc}=\frac{\text{集热器接收光线数目}}{\text{随机反射光线总数目}}. \tag{5.1.16}$$

5.1.5 余弦效率

对于第 i 面定日镜，其入射光线与反射光线的方向向量分别记为 $\boldsymbol{s_i},\boldsymbol{t_i}$，余弦效率 $\eta_{\cos}$[1]：

$$\eta_{cos}=\boldsymbol{s_i}\cdot\boldsymbol{t_i} \tag{5.1.17}$$

5.1.6 光学效率

设 d_{HR} 表示镜面中心到集热器中心的距离，则：
大气透射率：$\eta_{\mathrm{at}}=0.99321-0.0001176d_{\mathrm{HR}}+1.97\times10^{-8}\times d_{\mathrm{HR}}^2 \quad (d_{\mathrm{HR}}\le 1000)$
定日镜的光学效率：$\eta=\eta_{\mathrm{sb}}\eta_{\cos}\eta_{\mathrm{at}}\eta_{\mathrm{trunc}}\eta_{\mathrm{ref}} \quad (\eta_{\mathrm{ref}}=0.92)$

5.1.7 输出热功率

DNI (法向直接辐射辐照度) 是指地球上垂直于太阳光线的平面单位面积上、单位时间内接收到的太阳辐射能量, 可按以下公式近似计算：（$G_0 = 1.366\ \mathrm{kW/m^2}, H = 3\mathrm{km}$）

$$\begin{cases} \mathrm{DNI} = G_0 \left[a + b \exp\left(-\frac{c}{\sin \alpha_s}\right)\right] \\ a = 0.4237 - 0.00821(6 - H)^2 \\ b = 0.5055 + 0.00595(6.5 - H)^2 \\ c = 0.2711 + 0.01858(2.5 - H)^2 \end{cases} \tag{5.1.18}$$

设 N 为定日镜总数，A_i 为第 i 面定日镜采光面积，η_i 为第 i 面镜子的光学效率，则定日镜场的输出热功率 E_{field} 为：

$$E_{\mathrm{field}} = \mathrm{DNI} \cdot \sum_{i}^{N} A_i \eta_i, \tag{5.1.19}$$

5.2 问题一模型的求解

定日镜共计 1745 面，每面镜子上均匀选取 16 个点，考虑周围 5 面镜子的阴影遮挡，α 在 $[0, \theta]$ 区间内均匀选取 4 个值，β 在 $[0, 2\pi]$ 区间内均匀选取 8 个值。虽然理论上均匀采样数目越多得到的结果越精确，但现实受算力影响（程序运行时间 7.5h）我们的设置采样数目较为保守。对上述采样用 python 遍历输入模型（详见附录 A），得到的结论数据如下表：

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 ($\mathrm{kW/m^2}$)
1 月 21 日	0.4705	0.7199	0.8905	0.8567	0.4526
2 月 21 日	0.4838	0.7404	0.8865	0.8507	0.4998
3 月 21 日	0.4876	0.7611	0.9040	0.8131	0.5293
4 月 21 日	0.4995	0.7793	0.9045	0.8088	0.5593
5 月 21 日	0.5041	0.7893	0.8982	0.8106	0.5722
6 月 21 日	0.5046	0.7924	0.8934	0.8113	0.5749
7 月 21 日	0.5041	0.7892	0.8983	0.8106	0.5722
8 月 21 日	0.4984	0.7786	0.9035	0.8086	0.5576
9 月 21 日	0.4869	0.7601	0.9038	0.8134	0.5274
10 月 21 日	0.4720	0.7378	0.8969	0.8230	0.4838
11 月 21 日	0.4608	0.7182	0.8994	0.8329	0.4398
12 月 21 日	0.4522	0.7111	0.8937	0.8332	0.4170

表 3: 问题 *1* 每月 *21* 日平均光学效率及输出功率

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 ($\mathrm{kW/m^2}$)
0.4854	0.7565	0.8977	0.8227	32.3843	0.5155

表 4: 问题 *1* 年平均光学效率及输出功率表

6 问题二模型的建立与求解

6.1 问题二模型的建立

我们要在定日镜尺寸和安装高度相同的前提下，设计定日镜的尺寸、安装高度、数目，以及吸收塔与定日镜的位置坐标，使得定日镜场达到额定年平均输出功率 $60MW$ 且单位镜面面积年平均输出热功率尽量大。考虑到参数数目过多会导致运算量爆炸，我们采用了 Campo 布置方法 [**?**] 对镜场模型参数进行简化，并采用 PSO-GA 算法对模型优化问题进行求解。

6.1.1 初始镜场的 Campo 布置

Campo 是由 FJ Collado 等人 [**?**] 提出的一种圆形定日镜场的布置方式。镜场由内到外分为不同的环形区域，同一区域内的定日镜尺寸、高度相同，各行上的定日镜数量也相同，从而可以大大减小定日镜场的参数维数（图 16）。首先从第一个布置区域的第一行开始排布，首行第一个定日镜放置在镜场的正北方向，其余定日镜以不发生机械碰撞为原则进行周向均匀布置。第一行的半径 $R_1(m)$ 由第一行的定日镜数量 $Nhel_1$ 可通过计算得出：

$$\begin{cases} R_1 = (DM \cdot Nhel_1)/2\pi \\ DM = LW + desp \end{cases} \tag{6.1.1}$$

其中 $LW(m)$ 表示定日镜宽，$LH(m)$ 表示定日镜高，$DM(m)$ 为定日镜的特征圆直径，定日镜之间的安全距离 $desp = 5m$。第二行定日镜数目与第一行相同，与第一行定日镜交错放置且使相邻行定日镜的特征圆相切，即相邻行的定日镜之间以最密方式排列（图 17）。

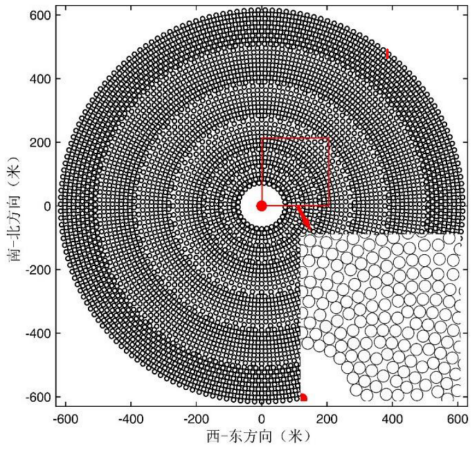

图 16: 问题 *2 Campo* 初始镜场排布方式示意图

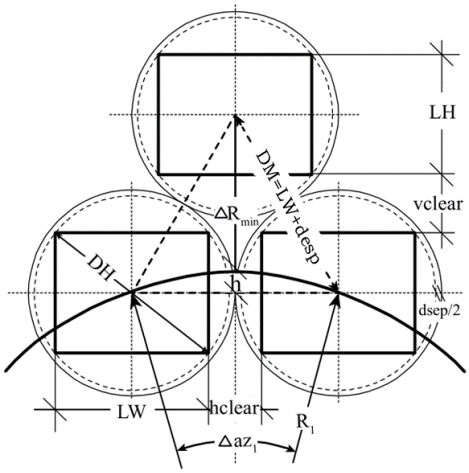

图 17: 问题 *2* 定日镜最密集排布下的几何关系示意图

由于在同一个区域各行定日镜的数量相同，所以奇数行与偶数行交错布置且连续行之间的径向间距保持不变。在最密集的排布方式下，相邻行半径的最小增量 ΔR_{min} 为：

$$\begin{cases} \Delta R_{\min} = DM \cdot \cos 30^\circ - h \\ h = R_1 - \sqrt{R_1^2 - \left(DM^2/4\right)} \end{cases} \tag{6.1.2}$$

由于同一区域内各行上的定日镜数量保持不变，所以同一行上相邻定日镜之间的方位间距会随着行半径的增加逐渐增大。当布置区域最后一行上两个相邻定日镜之间允许放置一个额外的定日镜时，开始进行下一区域的布置，其排布原理同第一个区域相同。因为下一区域内各行的定日镜数量为上一区域内各行定日镜数量的两倍,故各区域内每行的定日镜数量 $Nhel_n$ 与各区域首行的行半径 R_1^n 可由下式计算得出（n 为被布置区域的序号）:

$$\begin{aligned} Nhel_n &= Nhel_1 \cdot 2^{n-1} \\ R_n &= R_1 \cdot 2^{n-1} \end{aligned} \tag{6.1.3}$$

鉴于同一区域内相邻行之间的径向间距保持不变，可以根据这种特殊的布置关系计算出第 n 区域内所能允许布置的最大行数 $Nrows_n$

$$Nrows_n = \frac{2^{n-1} \cdot R_1}{\Delta R_{\min}} \cong \text{round}\left[\frac{2^{n-1} \cdot DM \cdot Nhel_1}{2\pi\left(DM \cdot \cos 30^\circ - h\right)}\right] \tag{6.1.4}$$

6.1.2 基于初始镜场的 PSO-GA 优化

遗传算法 (GA) GA 是一种计算模型，选择、交叉和变异是其最常用的三种遗传操作。选择是从整个种群中挑选出几个适应度较高的个体来培育新一代的过程。交叉是两个被选中的亲代个体交换部分基因以产生新个体的过程。突变会改变一个个体的一个或多个基因值，这可以保持一代到下一代的遗传多样性，且 GA 群体可以通过突变操作获得更好的解决方案。在选择之后，应用交叉和变异从这些被选中的个体中产生新的种群。

粒子群优化 (PSO) PSO 是一种基于种群的优化方法，其中每个粒子都会更新其位置和速度。在每次迭代中，每个粒子都会通过跟踪自身之前的最佳位置（pbest）和群组之前的最佳位置（gbest）来更新其速度和位置向量。

$$\begin{aligned} &\mathrm{v_i(t+1) = m \cdot v_i(t) + c_1 r_1 \left(pbest_i(t) - x_i(t)\right) + c_2 r_2 \left(gbest_i(t) - x_i(t)\right)} \\ &x_i(t+1) = x_i(t) + v_i(t+1) \\ &\mathrm{m} = \begin{cases} \mathrm{m_{min}} + \frac{(\mathrm{m_{max}} - \mathrm{m_{min}})(\mathrm{fit} - \mathrm{fit_{min}})}{\mathrm{fit_{ave}} - \mathrm{fit_{min}}} \leq \mathrm{fit_{ave}} \\ \mathrm{m_{max}} \mathrm{fit} > \mathrm{t_{ave}} \end{cases} \end{aligned} \tag{6.1.5}$$

PSO-GA 优化 每种优化算法都有各自的优缺点。混合算法的思想能综合不同优化算法的优点，又能避免它们的缺点。[**?**] 在 PSO-GA 算法中有七个参数 $m, c_1, c_2, r_1, r_2, Pc$ 和 Pm 用于控制优化过程。m 是自适应权重，用于根据上一代种群中每个粒子的适合度控制算法的探索能力；c_1 和 c_2 是两个加速度系数，均设置为 1.5。r_1 和 r_2 是两个随机数，均匀分布在区间 $[0,1]$ 内；Pc 是 PSO-GA 中 GA 部分的交叉操作概率，设为 0.8；Pm 是 PSO-GA 中 GA 部分的变异操作概率，设为 0.3。

首先随机生成一个 PSO 群体，根据每个粒子的适应度确定 pbest 和 gbest。继而根据每个粒子的位置和速度向量更新种群，执行交叉和突变操作并进一步更新种群中的粒子。然后重新计算适应度，更新 pbest 和 gbest。最后检查停止标准，即当当前生成量达到最大生成量（MG）或种群的适应度在 50 次迭代中保持不变。PSO-GA 原理示意图如（图 18）所示。

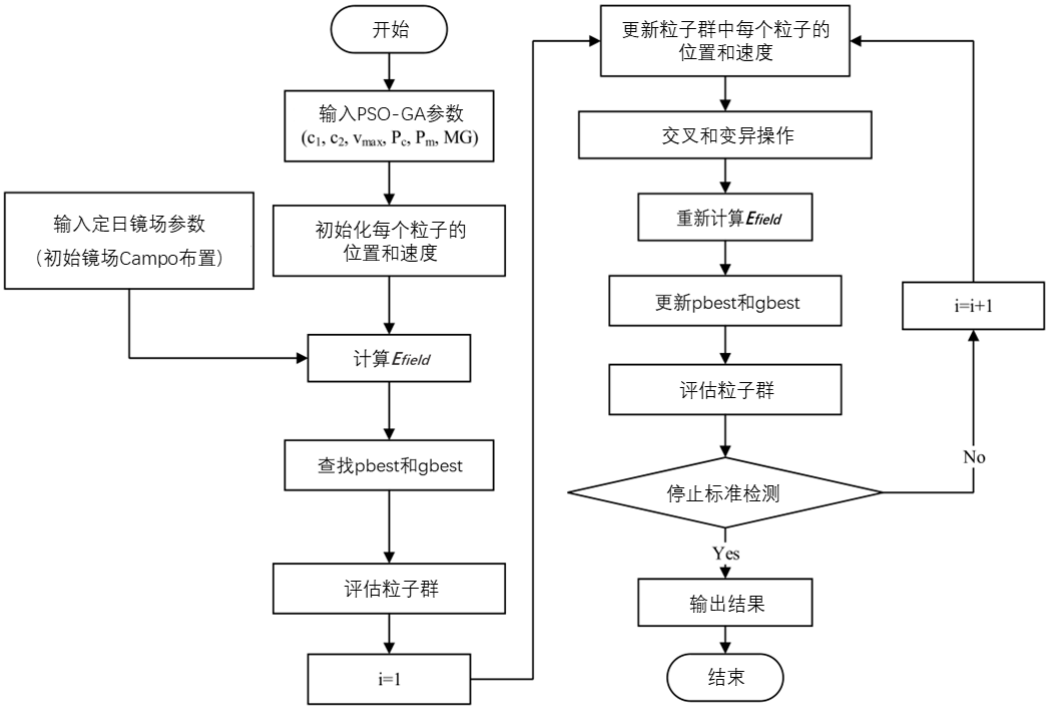

图 18: 问题 *2 PSO-GA* 算法原理示意图

鉴于所有定日镜的尺寸和安装高度相同，初始 Campo 建模后我们仅需优化以下参数：

符号	含义	维数
LH	定日镜镜高	1
LW	定日镜镜宽	1
h	定日镜安装高度	1
r_i	第 i 个区域第一行的半径	$i=4$
H_y	集热塔的 y 轴坐标	1

表 5: 问题 *2* 模型优化参数列表

优化限制：年平均输出热功率大于等于 $60MW$，$100m \leq$ 镜与塔的距离 $\leq 350m$。
优化目标：单位镜面面积年平均输出热功率尽量大。

6.1.3 低效率镜点剔除

初始镜场的 Campo 布置为圆环形，经过 PSO-GA 优化调参后整体布局仍为圆环形。鉴于该镜场位于北半球北纬 39.4°，圆环形的镜场中极有可能存在低效率的镜点，拉低了镜场的单位镜面面积年平均输出热功率。因此对于 PSO-GA 优化后的结果，需要设定合适的检验指标对各定日镜的年平均效率进行检测，将不达标的定日镜点提出，使得定日镜场在达到额定功率的条件下单位镜面面积年平均输出热功率尽量大。

6.2 问题二模型的求解

将上述定日镜场参数输入 PSO-GA 模型 (详见附录 B) 得到优化后的参数数据。由于镜场单位面积平均输出热功率主要与定日镜的光学效率有关，所以计算优化后的定日镜的年平均光学效率作为剔除低效率镜点的依据。按照光学效率从低到高的顺序分批剔除镜点，单位镜面面积年平均输出热功率逐渐增大，镜场的年平均输出功率逐渐减小，直至镜场的年平均输出功率大于等于 $60MW$，且继续剔除一个镜点会使镜场年平均输出功率小于 $60MW$ 为止。处理后得到镜场的优化分布如（图 19）所示，定日镜的年平均功率以热图的形式呈现。

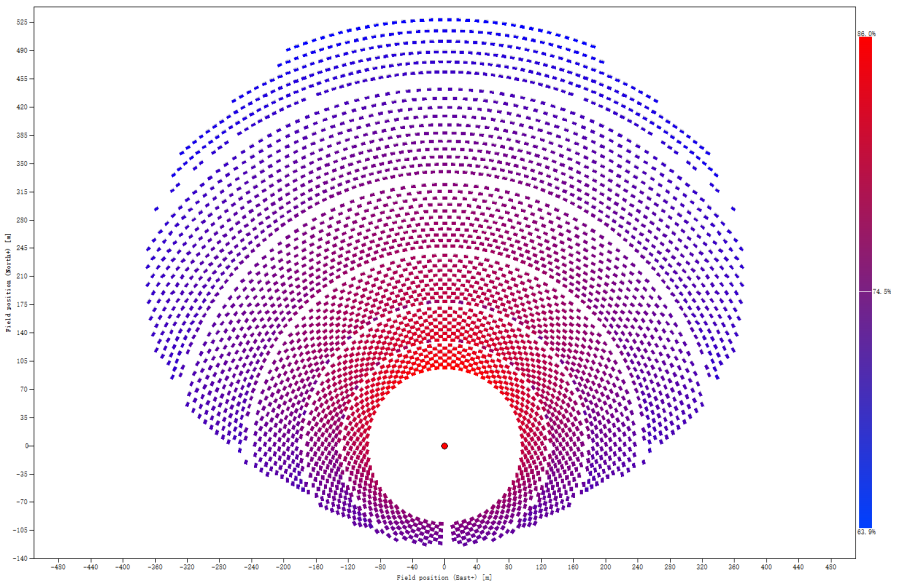

图 19: 问题 *2* 镜场优化分布功率热图

将处理后的定日镜和集热塔坐标储存在附件 result2.xlsx 中，计算整理得结论数据如下表。

7 问题三模型的建立与求解

7.1 问题三模型的建立

问题三要求在定日镜尺寸及安装高度可以不同的条件下设计定日镜场的各个参数，使得镜场达到额定年平均输出功率 $60MW$ 且单位镜面面积年平均输出热功率尽量大。我们仍采用问题二中的 Campo 方法布置初始镜场，再用 PSO-GA 算法对模型优化问题进行求解，最后进行低效率镜点的剔除。初始 Campo 建模后我们需要优化以下参数：

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	0.7406	0.8812	0.9672	0.9845	0.7594
2 月 21 日	0.7623	0.8808	0.9943	0.9856	0.7825
3 月 21 日	0.7595	0.8746	0.9964	0.9867	0.8146
4 月 21 日	0.7488	0.8609	0.9963	0.9880	0.8370
5 月 21 日	0.7370	0.8466	0.9962	0.9888	0.8391
6 月 21 日	0.7317	0.8402	0.9962	0.9891	0.8375
7 月 21 日	0.7372	0.8468	0.9962	0.9888	0.8393
8 月 21 日	0.7494	0.8617	0.9963	0.9879	0.8362
9 月 21 日	0.7599	0.8751	0.9964	0.9867	0.8122
10 月 21 日	0.7615	0.8811	0.9931	0.9855	0.7559
11 月 21 日	0.7371	0.8811	0.9630	0.9844	0.6599
12 月 21 日	0.7192	0.8787	0.9428	0.9839	0.7375

表 6: 问题 *2* 每月 *21* 日平均光学效率及输出功率

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)
0.7453	0.8674	0.9862	0.9867	60.3426	0.7927

表 7: 问题 *2* 年平均光学效率及输出功率表

吸收塔位置坐标	定日镜尺寸 (宽 × 高)	定日镜安装高度 (m)	定日镜总面数	定日镜总面积 (m^2)
(0,-210)	5*4	2.5	3743	74860

表 8: 问题 *2* 设计参数表

符号	含义	维数
LH_i	第 i 个区域内定日镜镜高	$i=4$
LW_i	第 i 个区域内定日镜镜宽	$i=4$
h_i	第 i 个区域内定日镜安装高度	$i=4$
r_i	第 i 个区域第一行的半径	$i=4$
H_y	集热塔的 y 轴坐标	1

表 9: 问题 *3* 模型优化参数列表

优化限制： 年平均输出热功率大于等于 $60MW$，$100m \leq$ 镜与塔的距离 $\leq 350m$。
优化目标： 单位镜面面积年平均输出热功率尽量大。

7.2 问题三模型的求解

将上述定日镜场参数输入 PSO-GA 模型 (详见附录 C) 得到优化后的参数数据。按照光学效率从低到高的顺序分批剔除镜点，单位镜面面积年平均输出热功率逐渐增大，镜场的年平均输出功率逐渐减小，直至镜场的年平均输出功率大于等于 $60MW$，且继续剔除一个镜点会使镜场年平均输出功率小于 $60MW$ 为止。处理后得到镜场的优化分布如（图 20）所示，定日镜的年平均功率以热图的形式呈现。

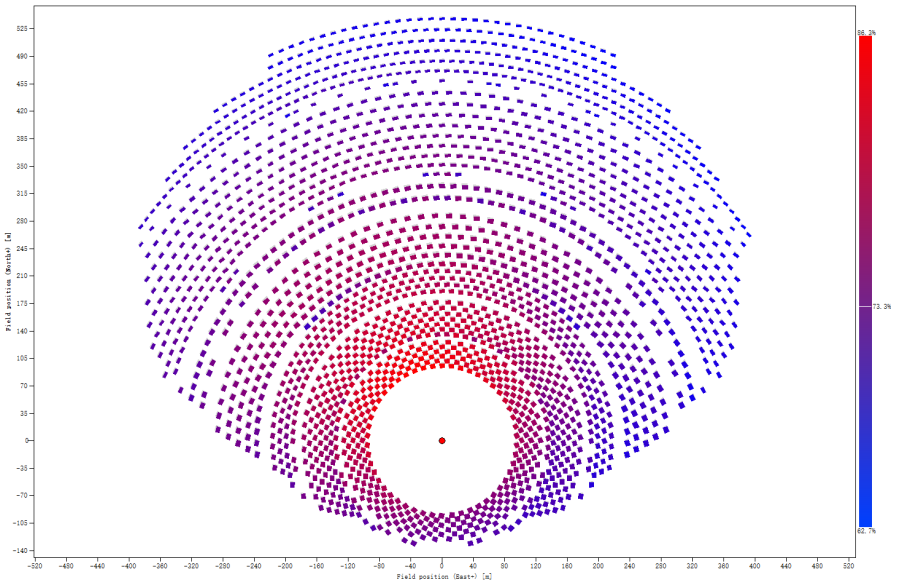

图 20: 问题 *3* 镜场优化分布功率热图

将处理后的定日镜和集热塔尺寸及坐标储存在附件 result3.xlsx 中，整理得结论数据如下。

8 模型的灵敏度分析

最后，为了检验光线追迹模型的稳定性，我们对问题 1 中结论数据进行了插值，插值曲

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	0.7561	0.8799	0.9530	0.9758	0.7982
2 月 21 日	0.7575	0.8793	0.9839	0.9776	0.8224
3 月 21 日	0.7537	0.8730	0.9922	0.9793	0.8561
4 月 21 日	0.7429	0.8588	0.9916	0.9810	0.8797
5 月 21 日	0.7312	0.8440	0.9913	0.9821	0.8818
6 月 21 日	0.7258	0.8374	0.9914	0.9825	0.8802
7 月 21 日	0.7313	0.8442	0.9913	0.9821	0.8820
8 月 21 日	0.7435	0.8596	0.9916	0.9809	0.8788
9 月 21 日	0.7541	0.8735	0.9922	0.9793	0.8536
10 月 21 日	0.7575	0.8796	0.9815	0.9774	0.7945
11 月 21 日	0.7559	0.8798	0.9487	0.9757	0.6935
12 月 21 日	0.7545	0.8792	0.9296	0.9749	0.7751

表 10: 问题 *3* 每月 *21* 日平均光学效率及输出功率

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)
0.7470	0.8657	0.9782	0.9791	61.4237	0.8331

表 11: 问题 *3* 年平均光学效率及输出功率表

吸收塔位置坐标	定日镜尺寸 (宽 × 高)	定日镜安装高度 (m)	定日镜总面数	定日镜总面积 (m^2)
(0,-200)	-	-	2005	73732

表 12: 问题 *3* 年平均光学效率及输出功率表

面图与等高线图如下（图 21）。观察得到的“日期-时间-参量”曲面图可知：曲面的变化较为平缓，光线追迹的蒙特卡洛算法灵敏度较低，模型稳定性很好。

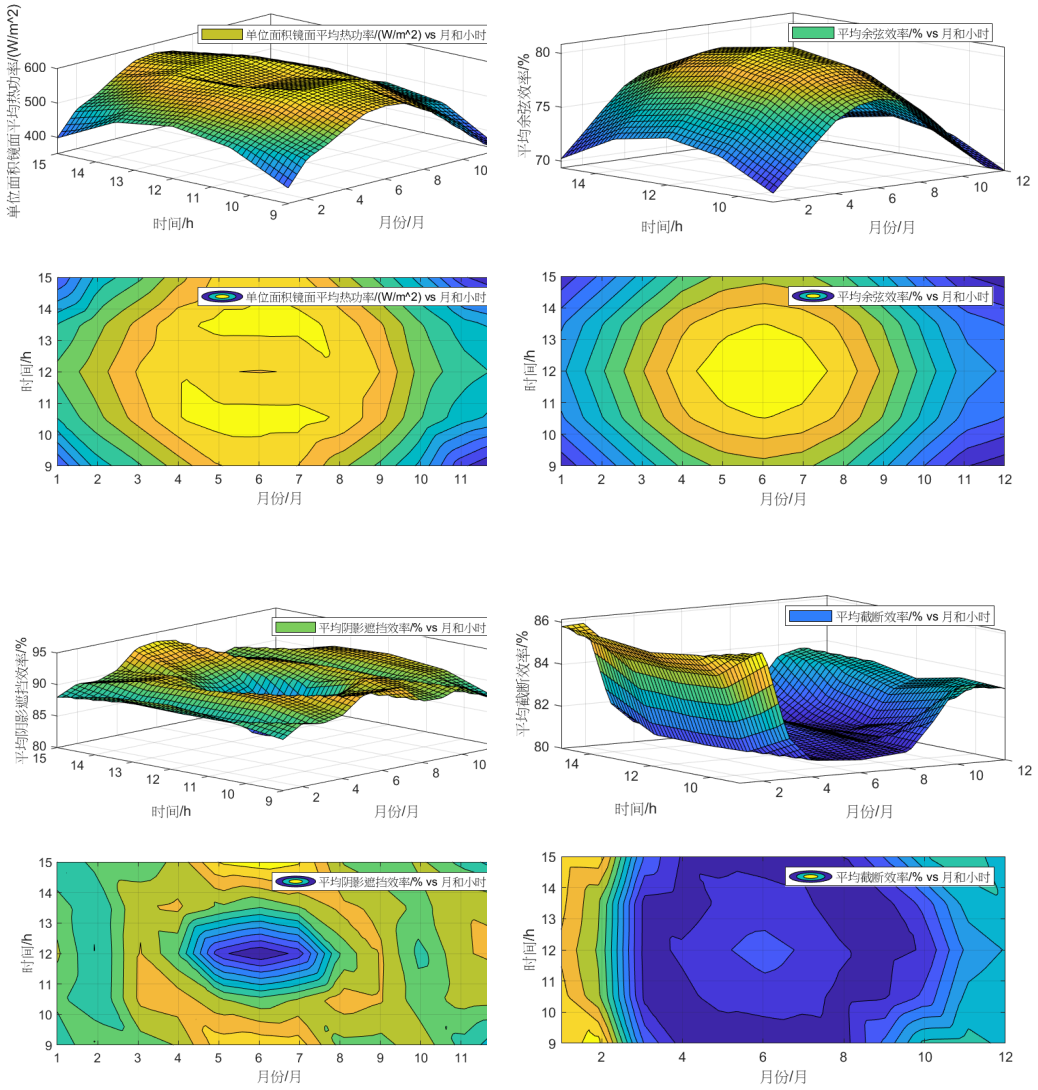

图 21: 问题 *1* 结论数据插值曲面图与等高线图

9 模型的评价与改进

9.1 问题一模型的评价与改进

模型优点

1. 周全考虑对阴影遮挡效率、截断效率造成损失的情况，建模直接，结论精确.

2. 从统计意义上对截断效率，阴影遮挡效率采用均匀取点的蒙特卡洛法进行计算，单点误差服从正态分布，使得总体平均误差较小，可信度高。

模型缺点

1. 复杂度上计算时间较长。

2. 阴影与遮挡效率的计算采用蒙特卡洛算法存在随机性，有一定的系统误差。

模型改进

1. 优化代码，用并行式计算减少计算时间。

2. 若以计算精确度为优先，则可以考虑将截断模型的蒙特卡洛计算模拟改用太阳光能量分布的高斯模型进行改进。

3. 若以降低时间计算复杂度为优先，考虑到截断效率平均都较高（$> 80\%$），可以考虑对截断效率以镜子关于塔的距离远近做分类，会极大减少计算量和时间。

9.2 问题二、三模型的评价与改进

模型优点

1、利用 Campo 布置，很大程度减少变量数，大大降低计算量。

2、PSO-GA 算法是群体智能算法，有利于在复杂函数内找到全局最优解。

3、截断效率用高斯分布近似，减少计算量。

模型缺点

1、计算复杂度太大，耗时很久。

2、PSO-GA 算法收敛速度慢，对初值敏感，不够稳定。

3、提前假设 Campo 布置为最佳布置方法，实际也有调整空间。

4、每个区域内所有定日镜的尺寸与高度可以不相等，固定为相等难以找到最优解。

5、高斯分布得到的截断效率与真实截断效率差距较大，并且没有考虑阴影遮挡，造成效率差距较大。

模型改进

1、优化算法计算过程，加速计算速度。

2、适当细化变量，增加部分变量使得优化更加精确。

3、对真实截断效率进行曲线拟合，取得比高斯分布更加精确的结果。

10 参考文献

[1] 刘建兴. 塔式光热电站光学效率建模仿真及定日镜场优化布置 [D]. 兰州: 兰州交通大学, 2022.

[2] 杜宇航，刘向民，王兴平，蒋志浩. 塔式光热电站定日镜不同聚焦策略的影响分析 [J] . 动力工程学报, 2022.40(5)：426-432

[3] 张平，奚正稳，华文瀚，王娟娟，孙登科. 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

[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] 高博，刘建兴，孙浩，刘二林. 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, 2022.10(43)：119-125

[6] O. Farges, J.J. Bezian, M. El Haf.Global optimization of solar power tower systems using a Monte Carlo algorithm Application to a redesign of the PS10 solar thermal power plant[J] . Renewable Energy, 2018 , 119：345-353

[7] Chao L, Zhai R, et al. Optimization of a heliostat field layout using hybrid PSO-GA algorithm[J]. Applied Thermal Engineering, 2018, 128(06):23-35

[8] Yu, S.; Zhang, J.; Zheng, S.; Sun, H. Provincial carbon intensity abatement potential estimation in China: A PSO–GA-optimized multi-factor environmental learning curve method. Energy Policy2015, 77, 46–55.

附录

附件清单

```
    result2.xlsx
    result3.xlsx
    问题1代码实现
    问题2代码实现
    问题3代码实现
    论文编译
    参考文献
```

A 问题一代码实现

确定太阳入射光线方向向量 (sun_vector.py)

```python
import datetime as datetime
import itertools
import pandas as pd
import numpy as np
import math


# 太阳时角计算
def get_omega(hour=9):
    omega = math.pi/12*(hour-12)
    return omega


def sun_vector(hour, phi=39.4, date='20220621', askfor='i') -> np.array:
    phi = phi*math.pi/180
    omega = get_omega(hour)      # 太阳时角

    cos_omega = math.cos(omega)
    sin_phi = math.sin(phi)
    cos_phi = math.cos(phi)
    date0 = datetime.date(2022,3,21)
    date = datetime.date(int(date[:4]),int(date[4:6]),int(date[6:]))
    D = (date-date0).days

    sin_delta = math.sin(2*math.pi*D/365)*math.sin(23.45*2*math.pi/360)  # 太阳赤纬角计算
    cos_delta = math.sqrt(1-sin_delta**2)

    sin_alpha_s = cos_delta*cos_phi*cos_omega + sin_delta * sin_phi
    cos_alpha_s = math.sqrt(1-sin_alpha_s**2)

    cos_gamma_s = round((sin_delta - sin_alpha_s*sin_phi)/(cos_alpha_s*cos_phi),5)
    if hour>12:
        sin_gamma_s = math.sqrt(1-cos_gamma_s**2)
    else:
        sin_gamma_s = -math.sqrt(1-cos_gamma_s**2)
    # 入射光线方向向量:
```

```
    if askfor=='i':
        i = np.array([cos_alpha_s*sin_gamma_s,cos_alpha_s*-cos_gamma_s,-sin_alpha_s])
        return i
    elif askfor=='sin_alpha_s':
        return sin_alpha_s
    elif askfor=='hlg':
        alpha_s = round(np.rad2deg(math.acos(cos_alpha_s)),2)
        if hour>12:
            gamma_s = round(360-np.rad2deg(math.acos(cos_gamma_s)),2)
        else:
            gamma_s = round(np.rad2deg(math.acos(cos_gamma_s)),2)
        return f'({alpha_s},{gamma_s})'
    elif askfor=='alpha_gamma_s':
        return sin_gamma_s,cos_gamma_s,sin_alpha_s/cos_alpha_s
    else:
        return None


def hlg_csv():
    hours = [9, 10.5, 12, 13.5, 15]
    dates = [f"2022{str(i).rjust(2, '0')}21" for i in range(1, 13)]
    df = pd.DataFrame(index=dates, columns=hours)
    for date, hour in itertools.product(dates, hours):
        df.loc[date, hour] = sun_vector(hour, phi=39.4, date=date, askfor='hlg')
    return df


if __name__ == '__main__':
    df = hlg_csv()
    df.to_csv('angle.csv')
```

确定镜子各个参数 (class_mirror.py)

```
import numpy as np
import pandas as pd
from sun_vector import sun_vector



def normalize_vector(vector):
    return vector/np.linalg.norm(vector)


class mirror(object):
    tower_loc = np.array([0, 0, 80])

    def __init__(self, name, x, y, center_z=4, length=6, width=6):
        self.top5_list = None
        self.len2center = None
        self.eta_cos = None
        self.n = None
        self.light_area = None
        self.name = name
        self.x = x
        self.y = y
        self.center_z = center_z
```

```
        self.length = length
        self.width = width
        self.area = length * width
        # 镜子中心坐标
        self.mirror_loc = np.array([self.x, self.y, self.center_z])
        # 单位反射光线向量
        self.reflect_vec = normalize_vector(self.tower_loc - self.mirror_loc)
        # 计算大气透射率
        self.d_hr = np.linalg.norm(self.tower_loc - self.mirror_loc)

    # 镜面单位法向量
    def normal_vector(self, sun_vec):
        n = -sun_vec + self.reflect_vec
        n = n/np.linalg.norm(n)
        self.n = n
        return n

    def eta_cos(self, sun_vec):
        eta_cos = abs(sum(sun_vec * self.n))
        self.eta_cos = eta_cos
        return eta_cos

    def len2center(self,other):
        len2center = np.linalg.norm(self.mirror_loc - other.mirror_loc)
        return len2center

    def top5_close(self, mirror_list):
        for mirror_k in mirror_list:
            mirror_k.len2center = mirror.len2center(mirror_k, self)
        top5_list = sorted(mirror_list)[1:5]
        self.top5_list = top5_list
        return self

    def __lt__(self, other):
        return self.len2center < other.len2center
    # # 采光面积 用法向量和入射光线来算
    # def light_area(self, sun_vec, area=36):
    #     cos_theta = sum(sun_vec * self.n)
    #     self.light_area = area*cos_theta
    #     return area*cos_theta

def main():
    hour = 9
    sun_vec = sun_vector(hour, askfor='i')
    x, y = 150, -150
    mirror1 = mirror('mirror1', x, y)
    print('Successful Class')

if __name__ == '__main__':
    main()
```

计算阴影遮挡效率 (eta_sb.py)

```
import itertools
import math
import time
```

```
4
5  import numpy as np
6  import pandas as pd
7  from sun_vector import sun_vector
8  from class_mirror import mirror
9  from eta_trunc import eta_trunc_1point
10
11
12 def normalize_vector(vector):
13     return vector/np.linalg.norm(vector)
14
15 def eta_sb(mirror_i,sun_vec,r_shadow,l_shadow,tan_alpha_s,width=6,height=6):
16     # start = time.time()
17     alpha_list = np.linspace(0, 4.65e-3, 4)
18     beta_list = np.linspace(0, 2 * np.pi, 8)
19     mirror_i_loc = mirror_i.mirror_loc
20     top5_list = mirror_i.top5_list       # exclude itself
21
22     # 大柱子挡住 直接返回
23     xij_r_shadow = np.dot(mirror_i_loc,r_shadow)
24     if xij_r_shadow > 0 and xij_r_shadow <= 84 / tan_alpha_s and abs(np.dot(mirror_i_loc, ↩
                                                 l_shadow)) <= 3.5 and mirror_i_loc[2] / (
25             84 / tan_alpha_s - xij_r_shadow) <= tan_alpha_s:
26         return [0,0]
27
28     ni = mirror_i.n; ni_x = ni[0];ni_y = ni[1];ni_z = ni[2]
29     list_x = np.linspace(-width/2, width/2, 4)
30     list_y = np.linspace(-height/2, height/2, 4)
31     l_i = normalize_vector(np.array([ni_y, -ni_x, 0]))
32     r_i = normalize_vector(np.array([-ni_x*ni_z, -ni_y*ni_z, ni_x**2+ni_y**2]))
33     reflect_vec = mirror_i.reflect_vec
34     count_df = pd.DataFrame({'eta_sb':[]})
35
36     # print(round(end - start, 5), '最前面的一段s')
37
38     # start_monte = time.time()
39     for xij, yij in itertools.product(list_x,list_y):
40         count_df.loc[f'({xij},{yij})','eta_sb'] = 0
41         vec_xij = xij*l_i + yij*r_i + mirror_i_loc
42         for mirror_k in top5_list:
43             # start = time.time()
44             mirror_k_n = mirror_k.n
45             mirror_k_loc = mirror_k.mirror_loc
46             nk_x = mirror_k_n[0]
47             nk_y = mirror_k_n[1]
48             nk_z = mirror_k_n[2]
49             l_k = normalize_vector(np.array([nk_y, -nk_x, 0]))
50             r_k = normalize_vector(np.array([-nk_x * nk_z, -nk_y * nk_z, nk_x ** 2 + nk_y **↩
                                                  2]))
51             aijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, ↩
                                                 sun_vec)*sun_vec
52             bijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, ↩
                                                 reflect_vec)*reflect_vec
53             # print(round(end - start, 5), '第一s')
54             # start = time.time()
55             if abs(np.dot((aijk - mirror_k_loc),l_k)) >= width/2:
56                 if abs(np.dot((aijk - mirror_k_loc),r_k)) >= height/2:
```

```
                    if abs(np.dot((bijk - mirror_k_loc),l_k)) >= width/2:
                        if abs(np.dot((bijk - mirror_k_loc) ,r_k)) >= height/2:
                            count_df.loc[f'({xij},{yij})','eta_sb'] = 1
                            count_df.loc[f'({xij},{yij})','eta_trunc_1point'] = eta_trunc_1point(reflect_vec,vec_xij,alpha_list,beta_list)
            # end = time.time()
            # print(round(end-start,5),'圆锥蒙特卡洛')
    # end_monte = time.time()
    # print(round(end_monte - start_monte, 5), '板蒙特卡洛 套蒙特卡洛s')

    # end = time.time()
    # print(round(end - start, 5), '一个点s')
    eta_sb = sum(count_df['eta_sb'])/len(count_df['eta_sb'])
    count_trunc = count_df[count_df['eta_sb'] == 1].copy()
    eta_trunc_whole = count_trunc['eta_trunc_1point'].mean()

    return [eta_sb,eta_trunc_whole]

# def get_nearst10_for():
#
def eta_at(d_hr):
    eta_at = 0.99321 - 0.0001176*d_hr + d_hr**2*1.97e-8
    return eta_at

# def eta_trunc(): # todo huge workload!!!!
#     return
#
# eta_ref = 0.92
# def eta()->pd.Series:
#     eta.name = 'eta'
#     return eta
```

计算集热器截断效率 (eta_trunc.py)

```
import numpy as np
import pandas as pd
from sun_vector import sun_vector
import class_mirror as c_m
import itertools
import math
import time
from class_mirror import mirror

alpha = [0,2.325*10**(-4),4.65*10**(-3)]
beta = [0, 0.4*np.pi, 0.8*np.pi, 1.2*np.pi, 1.6*np.pi]

def normalize_vector(vector):
    return vector/np.linalg.norm(vector)

def cut_off(reflect_vec, chosen_x, alpha, beta):
    # reflect_vec = mirror.reflect_vec
    e_l = c_m.normalize_vector(np.array([0,reflect_vec[2], -reflect_vec[1]]))
```

```
    e_h = c_m.normalize_vector(np.array([-reflect_vec[2]**2-reflect_vec[1]**2, reflect_vec[0]*reflect_vec[1], reflect_vec[0]*reflect_vec[2]]))
    t_cut = reflect_vec + np.tan(alpha)*(np.cos(beta)*e_l + np.sin(beta)*e_h)
    a = t_cut[0]**2 + t_cut[1]**2
    b = 2*np.dot(t_cut[0:2],chosen_x[0:2])
    c = chosen_x[0]**2+chosen_x[1]**2-12.25
    sol = quadratic_solver(a,b,c)
    if sol == 'Error':
        return 0
    elif len(sol) == 1: #只有一个交点 相切
        if 76 <= chosen_x[2] + sol[0]*t_cut[2] <= 84:
            return 1
        else:
            return 0
    else:
        t = np.min(np.abs(sol))
        if 76 <= chosen_x[2] + t*t_cut[2] <= 84 :
            return 1
        else:
            return 0



def quadratic_solver(a,b,c):
    delta = b**2-4*a*c
    if delta < 0:
        return 'Error'
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return [(-b + np.sqrt(delta))/(2*a), (-b - np.sqrt(delta))/(2*a)]

def eta_trunc_1point(reflect_vec,vec_xij,alpha_list,beta_list): # todo huge workload!!!!!
    count_series = pd.Series() # monte_carlo
    for alpha, beta in itertools.product(alpha_list,beta_list):
        count_series.loc[f'({alpha},{beta})'] = cut_off(reflect_vec, vec_xij, alpha, beta)
    eta_trunc_1point = sum(count_series)/len(count_series)
    return eta_trunc_1point
```

输出结论数据文件 (q1_debug.py)

```
import math
import numpy as np
import pandas as pd
from sun_vector import sun_vector
from class_mirror import mirror, normalize_vector
import itertools
from eta_sb import *
import warnings
import time
warnings.filterwarnings('ignore')
from eta_sb import eta_sb


def dni(sin_alpha_s):
```

```
    g0 = 1366
    H = 3
    a = 0.4237 - 0.000821*((6-H)**2)
    b = 0.5055 + 0.00595*((6.5-H)**2)
    c = 0.2711 + 0.01858*((2.5-H)**2)
    dni = g0 * (a+b*math.exp(-c/sin_alpha_s))
    return dni


def main():
    hours = [9, 10.5, 12, 13.5, 15]
    dates = [f"2022{str(i).rjust(2,'0')}21" for i in range(3,5)]
    df = pd.read_csv(r'C:\Users\Administrator\Desktop\2023国赛\A题\Q1\mirror_loc.csv')
    df.columns = ['x', 'y']
    df['(x,y)'] = list(zip(df['x'],df['y'],df.index+1))
    df['id'] = df.index+1
    date_mean_table = pd.DataFrame({'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                    eta_trunc': [], 'Efield': [], '↩
                                                    Efield_mirror_mean': []})
    hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], 'eta_trunc': [], ↩
                                                    'Efield': [],
                       'Efield_mirror_mean': []}

    df['mirrors'] = df['(x,y)'].apply(lambda x: mirror(x[2], x[0], x[1]))
    mirror_list = list(df['mirrors'])
    df['mirrors'] = df['mirrors'].apply(lambda x: mirror.top5_close(x, mirror_list))

    for date, hour in itertools.product(dates, hours):
        start = time.time()
        print('date',date)
        sun_vec = sun_vector(hour, phi=39.4, date=date)
        sin_alpha_s = sun_vector(hour, date=date, askfor='sin_alpha_s', phi=39.4)
        sin_gamma_s,cos_gamma_s,tan_alpha_s = sun_vector(hour, phi=39.4, date=date, askfor='↩
                                                    alpha_gamma_s')
        r_shadow = np.array([-sin_gamma_s, -cos_gamma_s, 0])
        l_shadow = np.array([cos_gamma_s,-sin_gamma_s,0])

        df['normal_vec'] = df['mirrors'].apply(lambda x: mirror.normal_vector(x, sun_vec))
        df['eta_cos'] = df['mirrors'].apply(lambda x: mirror.eta_cos(x, sun_vec))
        df['eta_at'] = df['mirrors'].apply(lambda x: eta_at(x.d_hr))
        df['mirror_i_loc'] = df['mirrors'].apply(lambda x: x.mirror_loc)
        df['mirror_area'] = df['mirrors'].apply(lambda x: x.area)

        # todo big work!!!!!!!!!!!!!
        df['eta_sb_trunc'] = df.apply(lambda x: eta_sb(x['mirrors'],sun_vec,r_shadow,↩
                                                    l_shadow,tan_alpha_s,width=6,height=6↩
                                                    ),axis=1)            # todo get a ↩
                                                    series

        df['eta_sb'] = df['eta_sb_trunc'].apply(lambda x: x[0])
        df['eta_trunc'] = df['eta_sb_trunc'].apply(lambda x: x[1])
        df['eta'] = df['eta_sb']*df['eta_at']*df['eta_cos']*df['eta_trunc']*0.92
        E_field = dni(sin_alpha_s)*sum(df['mirror_area']*df['eta'])
        hour_mean_table['hour'].append(hour)
        hour_mean_table['eta'].append(df['eta'].mean())
        hour_mean_table['eta_cos'].append(df['eta_cos'].mean())
        hour_mean_table['eta_sb'].append(df['eta_sb'].mean())
```

```
        hour_mean_table['eta_trunc'].append(df['eta_trunc'].mean())
        hour_mean_table['Efield'].append(E_field)
        hour_mean_table['Efield_mirror_mean'].append(E_field/sum(df['mirror_area']))
        end = time.time()
        print('一天一个时辰',end-start,'s')
        insert_df = pd.DataFrame(hour_mean_table)
        insert_df.to_csv(f'{date}{hour}.csv')
        if len(hour_mean_table['hour']) == 5:
            hour_mean_map = pd.DataFrame(hour_mean_table)
            hour_mean_map.to_csv(f'{date}.csv')
            date_mean_table.loc[date] = hour_mean_map.mean()
            hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                    eta_trunc': [], 'Efield': [],
                       'Efield_mirror_mean': []}

    date_mean_table.to_csv('Q1.csv')
if __name__ == '__main__':
    main()
```

B 问题二代码实现

确定镜子各个参数 (get_mirros_pos.py)

```
import pandas as pd

from class_mirror import mirror, normalize_vector
import itertools
from eta_sb import *
import warnings
import time

def get_mirrors_pos(r1,r2,r3,r4,c1,c2,c3,c4,
                    w1,w2,w3,w4,h1,h2,h3,h4,y_tower):
    # 定义参数
    tower_x = 0
    tower_y = y_tower
    tower_z = 0
    tower_radius = 100
    num_iterations = 6 #迭代次数
    mirrors_num = np.random.randint(5e6,6e6)
    df = pd.DataFrame({'x':np.random.random(mirrors_num),'y':np.random.random(mirrors_num)})

    # 定义六个半径和对应的尺寸
    radii = [r1]*4
    sizes = [c1, w1, h1]*4

    # 循环验证
    for iteration in range(1, num_iterations+1):
        test_df = perform_iteration(iteration,df,'20220321',hour=12)
    df = test_df

    # 生成均匀分布的点
    for radius, size in zip(radii, sizes):
        # 计算每个圆环上均匀分布的点数量，这里假设为100个
```

```
        num_points = 100

        # 生成均匀分布的角度
        angles = np.linspace(0, 2 * np.pi, num_points, endpoint=False)

        # 计算每个点的坐标
        x_coordinates = tower_x + radius * np.cos(angles)
        z_coordinates = tower_z + radius * np.sin(angles)

        # 创建一个DataFrame存储当前半径内的点
        radius_df = pd.DataFrame({'x': x_coordinates, 'y': tower_y, 'z': z_coordinates})

        # 添加尺寸信息
        radius_df['c'] = size[0]
        radius_df['w'] = size[1]
        radius_df['h'] = size[2]

        # 将当前半径内的点添加到总的DataFrame中
        df = df.append(radius_df, ignore_index=True)

    # 筛选出不在塔周围100米半径内的点
    distance_to_tower = np.sqrt((df['x'] - tower_x) ** 2 + (df['z'] - tower_z) ** 2)
    df = df[distance_to_tower > tower_radius]
    return df

def perform_iteration(iteration_number,df,date,hour):
    # 初始化参数
    df.columns = ['x', 'y']
    df['(x,y)'] = list(zip(df['x'],df['y'],df.index+1))
    df['id'] = df.index+1
    date_mean_table = pd.DataFrame({'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                    eta_trunc': [], 'Efield': [], '↩
                                                    Efield_mirror_mean': []})
    hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], 'eta_trunc': [], ↩
                                                    'Efield': [],
                       'Efield_mirror_mean': []}

    df['mirrors'] = df['(x,y)'].apply(lambda x: mirror(x[2], x[0], x[1]))
    mirror_list = list(df['mirrors'])
    df['mirrors'] = df['mirrors'].apply(lambda x: mirror.top5_close(x, mirror_list))
    print('date',date)
    sun_vec = sun_vector(hour, phi=39.4, date=date)
    sin_alpha_s = sun_vector(hour, date=date, askfor='sin_alpha_s', phi=39.4)
    sin_gamma_s,cos_gamma_s,tan_alpha_s = sun_vector(hour, phi=39.4, date=date, askfor='↩
                                                    alpha_gamma_s')
    r_shadow = np.array([-sin_gamma_s, -cos_gamma_s, 0])
    l_shadow = np.array([cos_gamma_s,-sin_gamma_s,0])
    df['eta_sb_trunc'] = df.apply(lambda x: eta_sb(x['mirrors'], sun_vec, r_shadow, l_shadow↩
                                                    , tan_alpha_s, width=6, height=6), axis=1↩
                                                    )
    return df
```

计算阴影遮挡效率 (eta_sb.py)

```
import itertools
import math
```

```
3  import time
4
5  import numpy as np
6  import pandas as pd
7  from sun_vector import sun_vector
8  from class_mirror import mirror
9  from eta_trunc import eta_trunc_1point
10
11
12 def normalize_vector(vector):
13     return vector/np.linalg.norm(vector)
14
15 def eta_sb(mirror_i,sun_vec,r_shadow,l_shadow,tan_alpha_s,width=6,height=6,center_z=4):
16     alpha_list = np.linspace(0, 4.65e-3, 4)
17     beta_list = np.linspace(0, 7/4 * np.pi, 8)
18     mirror_i_loc = mirror_i.mirror_loc
19     top5_list = mirror_i.top5_list       # exclude itself
20
21     # 大柱子挡住 直接返回
22     xij_r_shadow = np.dot(mirror_i_loc,r_shadow)
23     if xij_r_shadow > 0 and xij_r_shadow <= 84 / tan_alpha_s and abs(np.dot(mirror_i_loc, ↩
                                                l_shadow)) <= 3.5 and mirror_i_loc[2] / (
24             84 / tan_alpha_s - xij_r_shadow) <= tan_alpha_s:
25         return [0,0]
26
27     ni = mirror_i.n; ni_x = ni[0];ni_y = ni[1];ni_z = ni[2]
28     list_x = np.linspace(-width/2, width/2, 4)
29     list_y = np.linspace(-height/2, height/2, 4)
30     l_i = normalize_vector(np.array([ni_y, -ni_x, 0]))
31     r_i = normalize_vector(np.array([-ni_x*ni_z, -ni_y*ni_z, ni_x**2+ni_y**2]))
32     reflect_vec = mirror_i.reflect_vec
33     count_df = pd.DataFrame({'eta_sb':[]})
34
35
36     for xij, yij in itertools.product(list_x,list_y):
37         count_df.loc[f'({xij},{yij})','eta_sb'] = 0
38         vec_xij = xij*l_i + yij*r_i + mirror_i_loc
39         for mirror_k in top5_list:
40             mirror_k_n = mirror_k.n
41             mirror_k_loc = mirror_k.mirror_loc
42             nk_x = mirror_k_n[0]
43             nk_y = mirror_k_n[1]
44             nk_z = mirror_k_n[2]
45             l_k = normalize_vector(np.array([nk_y, -nk_x, 0]))
46             r_k = normalize_vector(np.array([-nk_x * nk_z, -nk_y * nk_z, nk_x ** 2 + nk_y **↩
                                                    2]))
47             aijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, ↩
                                                    sun_vec)*sun_vec
48             bijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, ↩
                                                    reflect_vec)*reflect_vec
49
50             if abs(np.dot((aijk - mirror_k_loc),l_k)) >= width/2:
51                 if abs(np.dot((aijk - mirror_k_loc),r_k)) >= height/2:
52                     if abs(np.dot((bijk - mirror_k_loc),l_k)) >= width/2:
53                         if abs(np.dot((bijk - mirror_k_loc) ,r_k)) >= height/2:
54                             count_df.loc[f'({xij},{yij})','eta_sb'] = 1
55                             count_df.loc[f'({xij},{yij})','eta_trunc_1point'] = ↩
```

```
                                                                    eta_trunc_1point(↩
                                                                    reflect_vec,↩
                                                                    vec_xij,↩
                                                                    alpha_list,↩
                                                                    beta_list)
56
57
58      eta_sb = sum(count_df['eta_sb'])/len(count_df['eta_sb'])
59      count_trunc = count_df[count_df['eta_sb'] == 1].copy()
60      eta_trunc_whole = count_trunc['eta_trunc_1point'].mean()
61
62      return [eta_sb,eta_trunc_whole]
63
64
65  def eta_at(d_hr):
66      eta_at = 0.99321 - 0.0001176*d_hr + d_hr**2*1.97e-8
67      return eta_at
```

计算集热器截断效率 (eta_trunc.py)

```
1   import numpy as np
2   import pandas as pd
3   from sun_vector import sun_vector
4   import class_mirror as c_m
5   import itertools
6   import math
7   import time
8   from class_mirror import mirror
9
10  alpha = [0,2.325*10**(-4),4.65*10**(-3)]
11  beta = [0, 0.4*np.pi, 0.8*np.pi, 1.2*np.pi, 1.6*np.pi]
12
13  def normalize_vector(vector):
14      return vector/np.linalg.norm(vector)
15
16  def cut_off(reflect_vec, chosen_x, alpha, beta):
17      # reflect_vec = mirror.reflect_vec
18      e_l = c_m.normalize_vector(np.array([0,reflect_vec[2], -reflect_vec[1]]))
19      e_h = c_m.normalize_vector(np.array([-reflect_vec[2]**2-reflect_vec[1]**2, reflect_vec[0↩
                                                ]*reflect_vec[1], reflect_vec[0]*↩
                                                reflect_vec[2]]))
20      t_cut = reflect_vec + np.tan(alpha)*(np.cos(beta)*e_l + np.sin(beta)*e_h)
21      a = t_cut[0]**2 + t_cut[1]**2
22      b = 2*np.dot(t_cut[0:2],chosen_x[0:2])
23      c = chosen_x[0]**2+chosen_x[1]**2-12.25
24      sol = quadratic_solver(a,b,c)
25      if sol == 'Error':
26          return 0
27      elif len(sol) == 1: #只有一个交点 相切
28          if 76 <= chosen_x[2] + sol[0]*t_cut[2] <= 84:
29              return 1
30          else:
31              return 0
32      else:
33          t = np.min(np.abs(sol))
34          if 76 <= chosen_x[2] + t*t_cut[2] <= 84 :
```

```
            return 1
        else:
            return 0



def quadratic_solver(a,b,c):
    delta = b**2-4*a*c
    if delta < 0:
        return 'Error'
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return [(-b + np.sqrt(delta))/(2*a), (-b - np.sqrt(delta))/(2*a)]

def eta_trunc_1point(reflect_vec,vec_xij,alpha_list,beta_list): # todo huge workload!!!!!
    count_series = pd.Series() # monte_carlo
    for alpha, beta in itertools.product(alpha_list,beta_list):
        count_series.loc[f'({alpha},{beta})'] = cut_off(reflect_vec, vec_xij, alpha, beta)
    eta_trunc_1point = sum(count_series)/len(count_series)
    return eta_trunc_1point
```

PSO-GA 算法 (pso_ga_2.py)

```
import pandas as pd
from class_mirror import mirror, normalize_vector
import itertools
from eta_sb import *
import warnings
import time
from get_mirros_pos import get_mirrors_pos

def dni(sin_alpha_s):
    g0 = 1366
    H = 3
    a = 0.4237 - 0.000821*((6-H)**2)
    b = 0.5055 + 0.00595*((6.5-H)**2)
    c = 0.2711 + 0.01858*((2.5-H)**2)
    dni = g0 * (a+b*math.exp(-c/sin_alpha_s))
    return dni


# 定义适应度函数（目标函数）
def objective_function(r1, r2, r3, r4, c1,  c2, c3, c4,w1, w2, w3, w4, h1, h2, h3, h4 ,↩
                                                y_tower):
    hours = [9, 10.5, 12, 13.5, 15]
    dates = [f"2022{str(i).rjust(2,'0')}21" for i in range(3,5)]

    df_get_mirrors_pos = get_mirrors_pos(r1,r2,r3,r4,c1,c2,c3,c4,
                                w1,w2,w3,w4,h1,h2,h3,h4,y_tower)  # 得到每个镜子初始化坐标初↩
                                                                                始化
    df = df_get_mirrors_pos.copy()

    df.columns = ['x', 'y','c','w','h']
    df['(x,y)'] = list(zip(df['x'],df['y'],df.index+1))
    df['id'] = df.index+1
```

```
    date_mean_table = pd.DataFrame({'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                    eta_trunc': [], 'Efield': [], '↩
                                                    Efield_mirror_mean': []})
    hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], 'eta_trunc': [], ↩
                                                    'Efield': [],
                       'Efield_mirror_mean': []}

    df['mirrors'] = df.apply(lambda x: mirror(x['(x,y)'][2], x['(x,y)'][0], x['(x,y)'][1],x[↩
                                                    'c'],x['w'],x['j']))
    mirror_list = list(df['mirrors'])
    df['mirrors'] = df['mirrors'].apply(lambda x: mirror.top5_close(x, mirror_list))
    E_field = []
    for date, hour in itertools.product(dates, hours):
        print('date',date)
        sun_vec = sun_vector(hour, phi=39.4, date=date)
        sin_alpha_s = sun_vector(hour, date=date, askfor='sin_alpha_s', phi=39.4)
        sin_gamma_s,cos_gamma_s,tan_alpha_s = sun_vector(hour, phi=39.4, date=date, askfor='↩
                                                        alpha_gamma_s')
        r_shadow = np.array([-sin_gamma_s, -cos_gamma_s, 0])
        l_shadow = np.array([cos_gamma_s,-sin_gamma_s,0])

        df['normal_vec'] = df['mirrors'].apply(lambda x: mirror.normal_vector(x, sun_vec))
        df['eta_cos'] = df['mirrors'].apply(lambda x: mirror.eta_cos(x, sun_vec))
        df['eta_at'] = df['mirrors'].apply(lambda x: eta_at(x.d_hr))
        df['mirror_i_loc'] = df['mirrors'].apply(lambda x: x.mirror_loc)
        df['mirror_area'] = df['mirrors'].apply(lambda x: x.area)

        df['eta_sb_trunc'] = df.apply(lambda x: eta_sb(x['mirrors'], sun_vec, r_shadow, ↩
                                                        l_shadow, tan_alpha_s, width=x['w'], ↩
                                                        height=x['h'], center_z=x['c']),axis=↩
                                                        1)            # todo get a series

        df['eta_sb'] = df['eta_sb_trunc'].apply(lambda x: x[0])
        df['eta_trunc'] = df['eta_sb_trunc'].apply(lambda x: x[1])
        df['eta'] = df['eta_sb']*df['eta_at']*df['eta_cos']*df['eta_trunc']*0.92
        E_field.append(dni(sin_alpha_s)*sum(df['mirror_area']*df['eta']))
    E_field_mean = pd.Series(E_field).mean()
    return E_field_mean


# PSO-GA算法
def pso_ga(objective_function, num_particles, num_dimensions, max_iterations,
           m, c1, c2, Pc, Pm):
    # 初始化粒子群和GA种群
    swarm_position = np.random.rand(num_particles, num_dimensions)
    swarm_velocity = np.random.rand(num_particles, num_dimensions)
    ga_population = np.random.rand(num_particles, num_dimensions)


    best_swarm_position = swarm_position.copy()
    best_swarm_value = np.zeros(num_particles)

    for i in range(num_particles):
        r1, c1, w1, h1, y_tower = list(swarm_position[i])
        r1, r2, r3, r4 = [r1] * 4
        c1, c2, c3, c4 = [c1] * 4
        w1, w2, w3, w4 = [w1] * 4
```

```
        h1, h2, h3, h4 = [h1] * 4
        best_swarm_value[i] = objective_function(r1, r2, r3, r4,  c1, c2, c3, c4,w1, w2, w3,↩
                                                 w4, h1, h2, h3, h4, y_tower)

    global_best_index = np.argmin(best_swarm_value)
    global_best_position = swarm_position[global_best_index]
    global_best_value = best_swarm_value[global_best_index]

    for iteration in range(max_iterations):
        for i in range(num_particles):
            current_value = objective_function(swarm_position[i])

            if current_value < best_swarm_value[i]:
                best_swarm_position[i] = swarm_position[i]
                best_swarm_value[i] = current_value

            if current_value < global_best_value:
                global_best_position = swarm_position[i]
                global_best_value = current_value

            r1, r2 = np.random.rand(), np.random.rand()
            swarm_velocity[i] = (m * swarm_velocity[i] +
                                 c1 * r1 * (best_swarm_position[i] - swarm_position[i]) +
                                 c2 * r2 * (global_best_position - swarm_position[i]))
            swarm_position[i] += swarm_velocity[i]

        selected_parents = np.random.choice(num_particles, num_particles // 2, replace=True)
        offspring = []
        for i in range(0, num_particles, 2):
            parent1, parent2 = selected_parents[i], selected_parents[i + 1]
            crossover_point = np.random.randint(num_dimensions)
            child1 = np.concatenate((ga_population[parent1][:crossover_point], ga_population↩
                                                    [parent2][crossover_point:]))
            child2 = np.concatenate((ga_population[parent2][:crossover_point], ga_population↩
                                                    [parent1][crossover_point:]))
            offspring.extend([child1, child2])

        offspring = np.array(offspring)

        crossover_mask = np.random.rand(num_particles, num_dimensions) < Pc
        ga_population[crossover_mask] = offspring[crossover_mask]

        mutation_mask = np.random.rand(num_particles, num_dimensions) < Pm
        mutation_values = np.random.rand(num_particles, num_dimensions)
        ga_population[mutation_mask] = mutation_values[mutation_mask]
    return global_best_position, global_best_value


def main():
    num_particles = 20
    num_dimensions = 5
    max_iterations = 100
    m = 0.5
    c1 = 1.5
    c2 = 1.5
    Pc = 0.8
    Pm = 0.3
```

```
134
135     best_position, best_value = pso_ga(objective_function, num_particles, num_dimensions, ↩
                                                    max_iterations,
136                                         m, c1, c2, Pc, Pm)
137     series = pd.Series(best_position,best_position)
138     series.to_csv('result.csv')
139 if __name__ == '__main__':
140     main()
```

C 问题三代码实现

确定镜子各个参数 (get_mirros_pos.py)

```
1  import pandas as pd
2
3  from class_mirror import mirror, normalize_vector
4  import itertools
5  from eta_sb import *
6  import warnings
7  import time
8
9  def get_mirrors_pos(r1,r2,r3,r4,c1,c2,c3,c4,
10                      w1,w2,w3,w4,h1,h2,h3,h4,y_tower):
11     # 定义参数
12     tower_x = 0
13     tower_y = y_tower
14     tower_z = 0
15     tower_radius = 100
16     num_iterations = 6 #迭代次数
17     mirrors_num = np.random.randint(5e5,6e5)
18     df = pd.DataFrame({'x':np.random.random(mirrors_num),'y':np.random.random(mirrors_num)})
19
20     # 定义六个半径和对应的尺寸
21     radii = [r1, r2, r3, r4,]
22     sizes = [c1, w1, h1, c2, w2, h2, c3, w3, h3, c4, w4, h4]
23
24     # 循环验证
25     for iteration in range(1, num_iterations+1):
26         test_df = perform_iteration(iteration,df,'20220321',hour=12)
27     df = test_df
28
29     # 生成均匀分布的点
30     for radius, size in zip(radii, sizes):
31         # 计算每个圆环上均匀分布的点数量，这里假设为100个
32         num_points = 100
33
34         # 生成均匀分布的角度
35         angles = np.linspace(0, 2 * np.pi, num_points, endpoint=False)
36
37         # 计算每个点的坐标
38         x_coordinates = tower_x + radius * np.cos(angles)
39         z_coordinates = tower_z + radius * np.sin(angles)
40
41         # 创建一个DataFrame存储当前半径内的点
```

```
        radius_df = pd.DataFrame({'x': x_coordinates, 'y': tower_y, 'z': z_coordinates})

        # 添加尺寸信息
        radius_df['c'] = size[0]
        radius_df['w'] = size[1]
        radius_df['h'] = size[2]

        # 将当前半径内的点添加到总的DataFrame中
        df = df.append(radius_df, ignore_index=True)

    # 筛选出不在塔周围100米半径内的点
    distance_to_tower = np.sqrt((df['x'] - tower_x) ** 2 + (df['z'] - tower_z) ** 2)
    df = df[distance_to_tower > tower_radius]
    return df

def perform_iteration(iteration_number,df,date,hour):
    # 初始化参数
    df.columns = ['x', 'y']
    df['(x,y)'] = list(zip(df['x'],df['y'],df.index+1))
    df['id'] = df.index+1
    date_mean_table = pd.DataFrame({'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                    eta_trunc': [], 'Efield': [], '↩
                                                    Efield_mirror_mean': []})
    hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], 'eta_trunc': [], ↩
                                                    'Efield': [],
                       'Efield_mirror_mean': []}

    df['mirrors'] = df['(x,y)'].apply(lambda x: mirror(x[2], x[0], x[1]))
    mirror_list = list(df['mirrors'])
    df['mirrors'] = df['mirrors'].swifter.apply(lambda x: mirror.top5_close(x, mirror_list))
    print('date',date)
    sun_vec = sun_vector(hour, phi=39.4, date=date)
    sin_alpha_s = sun_vector(hour, date=date, askfor='sin_alpha_s', phi=39.4)
    sin_gamma_s,cos_gamma_s,tan_alpha_s = sun_vector(hour, phi=39.4, date=date, askfor='↩
                                                    alpha_gamma_s')
    r_shadow = np.array([-sin_gamma_s, -cos_gamma_s, 0])
    l_shadow = np.array([cos_gamma_s,-sin_gamma_s,0])
    df['eta_sb_trunc'] = df.apply(lambda x: eta_sb(x['mirrors'], sun_vec, r_shadow, l_shadow↩
                                                    , tan_alpha_s, width=6, height=6), axis=1↩
                                                    )
    return df
```

计算阴影遮挡效率 (eta_sb.py)

```
import itertools
import math
import time

import numpy as np
import pandas as pd
from sun_vector import sun_vector
from class_mirror import mirror
from eta_trunc import eta_trunc_1point


def normalize_vector(vector):
```

```
    return vector/np.linalg.norm(vector)

def eta_sb(mirror_i,sun_vec,r_shadow,l_shadow,tan_alpha_s,width=6,height=6,center_z=4):
    alpha_list = np.linspace(0, 4.65e-3, 4)
    beta_list = np.linspace(0, 7/4 * np.pi, 8)
    mirror_i_loc = mirror_i.mirror_loc
    top5_list = mirror_i.top5_list       # exclude itself

    # 大柱子挡住 直接返回
    xij_r_shadow = np.dot(mirror_i_loc,r_shadow)
    if xij_r_shadow > 0 and xij_r_shadow <= 84 / tan_alpha_s and abs(np.dot(mirror_i_loc, l_shadow)) <= 3.5 and mirror_i_loc[2] / (
            84 / tan_alpha_s - xij_r_shadow) <= tan_alpha_s:
        return [0,0]

    ni = mirror_i.n; ni_x = ni[0];ni_y = ni[1];ni_z = ni[2]
    list_x = np.linspace(-width/2, width/2, 4)
    list_y = np.linspace(-height/2, height/2, 4)
    l_i = normalize_vector(np.array([ni_y, -ni_x, 0]))
    r_i = normalize_vector(np.array([-ni_x*ni_z, -ni_y*ni_z, ni_x**2+ni_y**2]))
    reflect_vec = mirror_i.reflect_vec
    count_df = pd.DataFrame({'eta_sb':[]})


    for xij, yij in itertools.product(list_x,list_y):
        count_df.loc[f'({xij},{yij})','eta_sb'] = 0
        vec_xij = xij*l_i + yij*r_i + mirror_i_loc
        for mirror_k in top5_list:
            mirror_k_n = mirror_k.n
            mirror_k_loc = mirror_k.mirror_loc
            nk_x = mirror_k_n[0]
            nk_y = mirror_k_n[1]
            nk_z = mirror_k_n[2]
            l_k = normalize_vector(np.array([nk_y, -nk_x, 0]))
            r_k = normalize_vector(np.array([-nk_x * nk_z, -nk_y * nk_z, nk_x ** 2 + nk_y **2]))
            aijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, sun_vec)*sun_vec
            bijk = vec_xij + np.dot(mirror_k_n,(mirror_k_loc-vec_xij))/np.dot(mirror_k_n, reflect_vec)*reflect_vec

            if abs(np.dot((aijk - mirror_k_loc),l_k)) >= width/2:
                if abs(np.dot((aijk - mirror_k_loc),r_k)) >= height/2:
                    if abs(np.dot((bijk - mirror_k_loc),l_k)) >= width/2:
                        if abs(np.dot((bijk - mirror_k_loc) ,r_k)) >= height/2:
                            count_df.loc[f'({xij},{yij})','eta_sb'] = 1
                            count_df.loc[f'({xij},{yij})','eta_trunc_1point'] = eta_trunc_1point(reflect_vec,vec_xij,alpha_list,beta_list)


    eta_sb = sum(count_df['eta_sb'])/len(count_df['eta_sb'])
    count_trunc = count_df[count_df['eta_sb'] == 1].copy()
    eta_trunc_whole = count_trunc['eta_trunc_1point'].mean()
```

```
61
62      return [eta_sb,eta_trunc_whole]
63
64
65  def eta_at(d_hr):
66      eta_at = 0.99321 - 0.0001176*d_hr + d_hr**2*1.97e-8
67      return eta_at
```

计算集热器截断效率 (eta_trunc.py)

```
1   import numpy as np
2   import pandas as pd
3   from sun_vector import sun_vector
4   import class_mirror as c_m
5   import itertools
6   import math
7   import time
8   from class_mirror import mirror
9
10  alpha = [0,2.325*10**(-4),4.65*10**(-3)]
11  beta = [0, 0.4*np.pi, 0.8*np.pi, 1.2*np.pi, 1.6*np.pi]
12
13  def normalize_vector(vector):
14      return vector/np.linalg.norm(vector)
15
16  def cut_off(reflect_vec, chosen_x, alpha, beta):
17      # reflect_vec = mirror.reflect_vec
18      e_l = c_m.normalize_vector(np.array([0,reflect_vec[2], -reflect_vec[1]]))
19      e_h = c_m.normalize_vector(np.array([-reflect_vec[2]**2-reflect_vec[1]**2, reflect_vec[0↩
                                                ]*reflect_vec[1], reflect_vec[0]*↩
                                                reflect_vec[2]]))
20      t_cut = reflect_vec + np.tan(alpha)*(np.cos(beta)*e_l + np.sin(beta)*e_h)
21      a = t_cut[0]**2 + t_cut[1]**2
22      b = 2*np.dot(t_cut[0:2],chosen_x[0:2])
23      c = chosen_x[0]**2+chosen_x[1]**2-12.25
24      sol = quadratic_solver(a,b,c)
25      if sol == 'Error':
26          return 0
27      elif len(sol) == 1: #只有一个交点 相切
28          if 76 <= chosen_x[2] + sol[0]*t_cut[2] <= 84:
29              return 1
30          else:
31              return 0
32      else:
33          t = np.min(np.abs(sol))
34          if 76 <= chosen_x[2] + t*t_cut[2] <= 84 :
35              return 1
36          else:
37              return 0
38
39
40
41  def quadratic_solver(a,b,c):
42      delta = b**2-4*a*c
43      if delta < 0:
44          return 'Error'
```

```
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return [(-b + np.sqrt(delta))/(2*a), (-b - np.sqrt(delta))/(2*a)]

def eta_trunc_1point(reflect_vec,vec_xij,alpha_list,beta_list): # todo huge workload!!!!!
    count_series = pd.Series() # monte_carlo
    for alpha, beta in itertools.product(alpha_list,beta_list):
        count_series.loc[f'({alpha},{beta})'] = cut_off(reflect_vec, vec_xij, alpha, beta)
    eta_trunc_1point = sum(count_series)/len(count_series)
    return eta_trunc_1point
```

PSO-GA 算法 (pso_ga_3.py)

```
import pandas as pd
from class_mirror import mirror, normalize_vector
import itertools
from eta_sb import *
import warnings
import time
from get_mirros_pos import get_mirrors_pos

def dni(sin_alpha_s):
    g0 = 1366
    H = 3
    a = 0.4237 - 0.000821*((6-H)**2)
    b = 0.5055 + 0.00595*((6.5-H)**2)
    c = 0.2711 + 0.01858*((2.5-H)**2)
    dni = g0 * (a+b*math.exp(-c/sin_alpha_s))
    return dni


# 定义适应度函数（目标函数）
def objective_function(r1, r2, r3, r4, c1, c2, c3, c4,w1, w2, w3, w4, h1, h2, h3, h4, ↩
                                                y_tower):
    hours = [9, 10.5, 12, 13.5, 15]
    dates = [f"2022{str(i).rjust(2,'0')}21" for i in range(3,5)]
    df_get_mirrors_pos = get_mirrors_pos(r1,r2,r3,r4,c1,c2,c3,c4,
                                w1,w2,w3,w4,h1,h2,h3,h4,y_tower)  # 得到每个镜子初始化坐标初↩
                                                                                始化
    df = df_get_mirrors_pos.copy()

    df.columns = ['x', 'y','c','w','h']
    df['(x,y)'] = list(zip(df['x'],df['y'],df.index+1))
    df['id'] = df.index+1
    date_mean_table = pd.DataFrame({'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], '↩
                                                eta_trunc': [], 'Efield': [], '↩
                                                Efield_mirror_mean': []})
    hour_mean_table = {'hour': [], 'eta': [], 'eta_cos': [], 'eta_sb': [], 'eta_trunc': [], ↩
                                                'Efield': [],
                       'Efield_mirror_mean': []}

    df['mirrors'] = df.apply(lambda x: mirror(x['(x,y)'][2], x['(x,y)'][0], x['(x,y)'][1],x[↩
                                                'c'],x['w'],x['j']))
    mirror_list = list(df['mirrors'])
    df['mirrors'] = df['mirrors'].apply(lambda x: mirror.top5_close(x, mirror_list))
```

```
    E_field = []
    for date, hour in itertools.product(dates, hours):
        print('date',date)
        sun_vec = sun_vector(hour, phi=39.4, date=date)
        sin_alpha_s = sun_vector(hour, date=date, askfor='sin_alpha_s', phi=39.4)
        sin_gamma_s,cos_gamma_s,tan_alpha_s = sun_vector(hour, phi=39.4, date=date, askfor='alpha_gamma_s')
        r_shadow = np.array([-sin_gamma_s, -cos_gamma_s, 0])
        l_shadow = np.array([cos_gamma_s,-sin_gamma_s,0])

        df['normal_vec'] = df['mirrors'].apply(lambda x: mirror.normal_vector(x, sun_vec))
        df['eta_cos'] = df['mirrors'].apply(lambda x: mirror.eta_cos(x, sun_vec))
        df['eta_at'] = df['mirrors'].apply(lambda x: eta_at(x.d_hr))
        df['mirror_i_loc'] = df['mirrors'].apply(lambda x: x.mirror_loc)
        df['mirror_area'] = df['mirrors'].apply(lambda x: x.area)

        df['eta_sb_trunc'] = df.apply(lambda x: eta_sb(x['mirrors'], sun_vec, r_shadow, l_shadow, tan_alpha_s, width=x['w'], height=x['h'], center_z=x['c']),axis=1)            # todo get a series

        df['eta_sb'] = df['eta_sb_trunc'].apply(lambda x: x[0])
        df['eta_trunc'] = df['eta_sb_trunc'].apply(lambda x: x[1])
        df['eta'] = df['eta_sb']*df['eta_at']*df['eta_cos']*df['eta_trunc']*0.92
        E_field.append(dni(sin_alpha_s)*sum(df['mirror_area']*df['eta']))
    E_field_mean = pd.Series(E_field).mean()
    return E_field_mean


# PSO-GA算法
def pso_ga(objective_function, num_particles, num_dimensions, max_iterations,
           m, c1, c2, Pc, Pm):
    # 初始化粒子群和GA种群
    swarm_position = np.random.rand(num_particles, num_dimensions)
    swarm_velocity = np.random.rand(num_particles, num_dimensions)
    ga_population = np.random.rand(num_particles, num_dimensions)

    best_swarm_position = swarm_position.copy()
    best_swarm_value = np.zeros(num_particles)

    for i in range(num_particles):
        r1, r2, r3, r4, c1, c2, c3, c4,\
        w1, w2, w3, w4, h1, h2, h3, h4, y_tower = list(swarm_position[i])
        best_swarm_value[i] = objective_function(r1, r2, r3, r4,  c1, c2, c3, c4,w1, w2, w3, w4, h1, h2, h3, h4, y_tower)

    global_best_index = np.argmin(best_swarm_value)
    global_best_position = swarm_position[global_best_index]
    global_best_value = best_swarm_value[global_best_index]

    for iteration in range(max_iterations):
        for i in range(num_particles):
            current_value = objective_function(swarm_position[i])

            if current_value < best_swarm_value[i]:
                best_swarm_position[i] = swarm_position[i]
                best_swarm_value[i] = current_value
```

```
89
90              if current_value < global_best_value:
91                  global_best_position = swarm_position[i]
92                  global_best_value = current_value
93
94              r1, r2 = np.random.rand(), np.random.rand()
95              swarm_velocity[i] = (m * swarm_velocity[i] +
96                                   c1 * r1 * (best_swarm_position[i] - swarm_position[i]) +
97                                   c2 * r2 * (global_best_position - swarm_position[i]))
98              swarm_position[i] += swarm_velocity[i]
99
100         selected_parents = np.random.choice(num_particles, num_particles // 2, replace=True)
101         offspring = []
102         for i in range(0, num_particles, 2):
103             parent1, parent2 = selected_parents[i], selected_parents[i + 1]
104             crossover_point = np.random.randint(num_dimensions)
105             child1 = np.concatenate((ga_population[parent1][:crossover_point], ga_population↩
                                                                [parent2][crossover_point:]))
106             child2 = np.concatenate((ga_population[parent2][:crossover_point], ga_population↩
                                                                [parent1][crossover_point:]))
107             offspring.extend([child1, child2])
108
109         offspring = np.array(offspring)
110
111         crossover_mask = np.random.rand(num_particles, num_dimensions) < Pc
112         ga_population[crossover_mask] = offspring[crossover_mask]
113
114         mutation_mask = np.random.rand(num_particles, num_dimensions) < Pm
115         mutation_values = np.random.rand(num_particles, num_dimensions)
116         ga_population[mutation_mask] = mutation_values[mutation_mask]
117     return global_best_position, global_best_value
118
119
120 def main():
121     num_particles = 20
122     num_dimensions = 17
123     max_iterations = 100
124     m = 0.5
125     c1 = 1.5
126     c2 = 1.5
127     Pc = 0.8
128     Pm = 0.3
129
130     best_position, best_value = pso_ga(objective_function, num_particles, num_dimensions, ↩
                                                        max_iterations,
131                                        m, c1, c2, Pc, Pm)
132     series = pd.Series(best_position,best_position)
133     series.to_csv('result.csv')
134 if __name__ == '__main__':
135     main()
```