塔式光热电站镜场光学效率计算及输出热功率优化设计

摘要

本文在求定日镜光学效率及单位镜面面积年平均输出热功率的设计中，通过光反射分析，建立了太阳光、定日镜和集热器光反射-接收模型，并使用蒙特卡罗光线追踪法和 Möller–Trumbore 算法结合，同时建立镜面坐标系并求出与地面坐标系之间的转换矩阵，求解出阴影遮挡、集热截断效率等参数。在此基础上，通过粒子群-遗传混合算法及自适应引力搜索算法，求出了最大单位镜面面积年平均输出热功率和相应的吸收塔位置坐标、定日镜尺寸、定日镜数目等参数，并对模型进行了检验和误差分析。

针对问题一：首先, 基于对坐标系灵活选取，建立了定日镜-集热器光线模型。其次，建立了镜面坐标系，推导出其与地面坐标系之间的转换矩阵。最后，使用蒙特卡洛光线追踪法和离散化镜面法，结合 Möller–Trumbore 算法求解出阴影遮挡效率和截断效率，进行求解出平均光学效率。得到了 1 月 21 日平均光学效率为 0.6535332，平均余弦效率为 0.7649707，平均阴影遮挡效率为 0.9847608，平均截断效率为 0.9751661，单位平均输出热功率为 0.6992664。结果详细见文中表 2 和表 3。

针对问题二：在上述定日镜-集热器光线模型和坐标系转换的基础上，由于塔的坐标移动会大大提高数据量，因而我们进行分步，先在大圆上确定定日镜排布，再用一小圆移动坐标系确定吸收塔位置，最后计算最优解。首先建立大圆的坐标系，然后使用粒子群-遗传混合算法及自适应引力搜索算法，求解出定日镜的最优排布，和镜子的最优尺寸然后我们再在 x 轴上移动半径为 350m 的真圆，搜索最大化目标函数的圆心位置，即吸收塔的位置。得到了单位镜面面积年平均输出热功率为 0.6690383166。结果详细见表 1、表 2、表 3 及 result2.xlsx 文件。

针对问题三：与第二问类似，新增要求每片镜子的尺寸，如果逐一进行计算会过于复杂，因此，我们通过聚类方法将定日镜分为几类，每类的尺寸相同，在此过程中，与第二问类似，使用粒子群-遗传混合算法及自适应引力搜索算法优化定日镜排布。通过多重聚类的方法定义聚类目标函数，为每一类定日镜定义一个聚类目标函数。使用 k-means 聚类算法来求解每一类定日镜的尺寸和高度以及位置和吸收塔的位置坐标。结果详细见表 1、表 2、表 3 及 result3.xlsx。

关键字: 塔式光热电站镜场; 光学效率; 平均输出热功率; 蒙特卡洛光线追踪法; 粒子群-遗传混合算法; k-means 聚类算法

1.问题重述

1.1问题的背景

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。但是定日镜场子系统与塔式光热电站的综合性能有着密切的关系，是塔式光热电站的关键子系统之一。镜场中的定日镜数目众多，而电站一旦建成定日镜的位置就无法修改，因此定日镜场的光学效率计算问题与优化布置理论显得至关重要。

1.2问题的重述

装置结构：塔式太阳能光热发电站由定日镜、吸收塔、集热器组成。定日镜由底座和平面反射镜构成，底座由纵向和水平转轴组成，可以控制反射镜的方向，集热器位于镜场中吸收塔的顶端。

工作原理：塔式电站由大量的定日镜组成阵列，定日镜将太阳光反射汇聚到安装在镜场中吸收塔顶端的集热器中，加热其中的导热介质，并将太阳能以热能形式储存起来，经热交换实现由热能向电能的转化。

问题一：已知吸收塔建于该圆形定日场中心，定日镜为 6m*6m, 安装高度均为 4m，并已知所有定日镜位置。求 1-12 月每个月平均余弦效率、平均阴影遮挡效率、平均截断效率，平均光学效率、单位面积镜面平均输出热功率以及年平均各项参数值。

问题二：目标要使单位镜面面积年平均输出热功率最大，已知所有定日镜尺寸及安装高度相同，定日镜数目、位置、尺寸、高度以及吸收塔的位置均未知。约束限定定日镜场的额定年平均输出热功率为 60MW，安装高度在 2m 至 6m 之间，镜面边长在 2m 至 8m 之间，且相邻底座中心之间的距离比镜面宽度多 5m。

问题三：在第二问的基础上，需要重新设计定日镜场的各个参数，以满足额定功率的条件，并使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

问题一首先需要建立定日镜-集热器光线模型，本小问难点在于准确计算阴影遮挡效率等光学效率参数。对此，灵活选取坐标系，在镜场坐标系中引入镜面坐标系，推导出两者之间的转换矩阵。同时使用蒙特卡罗光线追踪法，离散化定日镜的表面，等效计算每个点上计算光线效率。然后使用 Möller–Trumbore 算法结合上述基础，求解出阴影遮挡效率和截断效率以及其他光学效率参数。

2.2问题二的分析

在问题一的基础上，在定日镜参数未定的情况下，需要求出单位镜面面积年平均输出热功率尽可能大。本小问分两步进行求解，第一步通过粒子群-遗传混合算法和引力搜索优化法求出最优定光镜排布和尺寸，第二步通过坐标系移动求出接收塔最佳位置。

2.3问题三的分析

在问题二的基础上，由于定日镜尺寸不同、高度也不同，需要进一步考虑以单位镜面面积年平均输出热功率最大为目标的优化函数模型，通过多重聚类的方法定义聚类目标函数，最后使用 k-means 聚类算法来求解每一类定日镜的尺寸和高度以及位置和吸收塔的位置坐标。

3.模型假设

为了适当地对模型进行合理化简化，本文给出如下假设：

1、假设吸收塔的直径与集热器直径相同且上下柱体相同均为 7 米。

2、假设每月 21 日均为晴天，不考虑阴雨等天气对数据的影响。

3、假设所有平面镜完好且不考虑平面镜上灰尘等影响。

4.符号说明

符号	含义	单位
α_s	太阳高度角	°
γ_s	太阳方位角	°
φ	当地纬度	°
ω	太阳时角	°
δ	太阳赤纬角	°
ST	当地时间	-
D	计算天数	-
DNI	法向直接辐射辐照度	kw/m^2
H	海拔高度	km
G_s	太阳常数	kw/m^2
η	光学效率	-
E_{field}	定日镜场输出热功率	kw
N	定日镜总数	面
A_i	第 i 面定日镜采光面积	m^2
η_i	第 i 面镜子的光学效率	-
η_{sb}	阴影遮挡效率	-
η_{cos}	余弦效率	-
η_{at}	大气透射率	-
η_{trunc}	余弦效率	-
η_{ref}	镜面反射率	-
d_{HR}	镜面中心到集热器中心的距离	m

5.模型的建立与求解

5.1问题一模型的建立与求解

5.1.1镜面坐标系的建立与转换

由于在计算光学效率的过程中，需要建立镜面坐标系与镜场坐标系进行转换设有一个镜场坐标系下的向量：

$$\vec{a} = (\vec{i}, \vec{j}, \vec{k}) \tag{1}$$

在镜面坐标系下为：

$$\vec{b} = (\overrightarrow{i_i}, \overrightarrow{j_i}, \overrightarrow{k_i}) \tag{2}$$

$$P_i = \begin{pmatrix} \overrightarrow{i_{i_x}} & \overrightarrow{j_{i_x}} & \overrightarrow{en_{i_x}} \\ \overrightarrow{i_{i_y}} & \overrightarrow{j_{i_y}} & \overrightarrow{en_{i_y}} \\ 0 & \overrightarrow{j_{i_z}} & \overrightarrow{en_{i_z}} \end{pmatrix} \tag{3}$$

经推导可得镜面坐标系和镜场坐标系存在 $\vec{b} = \vec{a}P_i$

且

$$P_i = \begin{pmatrix} \overrightarrow{i_{i_x}} & \overrightarrow{j_{i_x}} & \overrightarrow{en_{i_x}} \\ \overrightarrow{i_{i_y}} & \overrightarrow{j_{i_y}} & \overrightarrow{en_{i_y}} \\ 0 & \overrightarrow{j_{i_z}} & \overrightarrow{en_{i_z}} \end{pmatrix} \tag{4}$$

又因为 $(\overrightarrow{en_{i_x}}, \overrightarrow{en_{i_y}}, \overrightarrow{en_{i_z}}) = \overrightarrow{n_i}$,$\overrightarrow{n_i}$ 为镜面法向量。且

$$\overrightarrow{i_i} \perp \overrightarrow{k_i} \tag{5}$$

所以有

$$\begin{cases} \overrightarrow{i_{i_x}}\overrightarrow{en_{i_x}} + \overrightarrow{i_{i_y}}\overrightarrow{en_{i_y}} = 0 \\ \overrightarrow{i_{i_x}}^2 + \overrightarrow{i_{i_y}}^2 = 1 \end{cases} \tag{6}$$

经过推导可知矩阵 P_i 为一个正交矩阵，所以

$$P_i^T = P_i^{-1} \tag{7}$$

由此可以推出 $\vec{a} = P_i^T\vec{b}$

5.1.2蒙特卡罗光线追踪法模型建立

首先定义一个反射方程

$$L_r(\vec{x}, \overrightarrow{\omega_r}) = \int_\Gamma f_r(\vec{x}, \overrightarrow{\omega_i} \to \overrightarrow{\omega_r} L_i(\vec{x}, \overrightarrow{\omega_i}) \cos(\theta_i) d\omega_i) \tag{8}$$

经计算，形式解为

$$L(\vec{x},\vec{\omega})=\sum_{i=0}^{\infty}K^{i}L_{e}(\overrightarrow{x_{0}},\overrightarrow{\omega_{0}}) \tag{9}$$

5.1.3Möller–Trumbore 算法模型

首先定义一个标量三重积，设 $\vec{a}$,$\vec{b}$,$\vec{c}$ 为三个向量

$$a\cdot(b\times c)=b\cdot(c\times a)=c\cdot(a\times b)=A \tag{10}$$

$$A=\begin{pmatrix}a_{one} & a_{two} & a_{three}\\ b_{one} & b_{two} & b_{three}\\ c_{one} & c_{two} & c_{three}\end{pmatrix} \tag{11}$$

任意兑换两个向量的位置，标量三重积与原来相差一个负号：

$$\begin{cases}a\cdot(b\times c)=-a\cdot(c\times b)\\ a\cdot(b\times c)=-b\cdot(a\times c)\\ a\cdot(b\times c)=-c\cdot(b\times a)\end{cases} \tag{12}$$

通过克莱姆法则，一个线性方程组可以用矩阵与向量的方程来表示 $Ax=c$, 如果 A 是一个可逆矩阵 $(det(A)\neq 0)$, 那么方程有解 $x=(x_1,x_2,...X_n)^T$，其中 A_i 是被列向量取代了 A 的第 i 列的列向量后得到的矩阵。

经过推导可知：根据克莱姆法则

$$t=\frac{det[SE_1E_2]}{det[-dE_1E_2]} \tag{13}$$

根据三重积

$$det([SE_1E_2])=(S\times E_1)\cdot E_2 \tag{14}$$

$$det([-dE_1E_2])=-d\cdot(E_1\times E_2)=E_1\cdot(d\times E_2) \tag{15}$$

可以得到

$$b1=\frac{det[-dSE_2]}{det[-dE_1E_2]}=\frac{S_1\cdot S}{S_1\cdot E_1} \tag{16}$$

$$b2=\frac{det[-dE_1S]}{det[-dE_1E_2]}=\frac{d\cdot(S\times E_2)}{S_1\cdot E_1} \tag{17}$$

整理得：

$$[t,b1,b2]^T=\frac{1}{S_1\cdot E_1}[S_2\cdot E_2,S_1\cdot S,S_2\cdot d]^T \tag{18}$$

5.1.4求解过程

1. 求解余弦效率：

已知

$$\begin{cases} x_i = cos(\alpha_s) * cos(90^\circ - \gamma_s) \\ y_i = cos(\alpha_s) * sin(90^\circ - \gamma_s) \\ z_i = sin(\alpha_s) \end{cases} \tag{19}$$

假设镜面中心指向集热器中心的向量为 $\overrightarrow{AO}$：

$$\vec{AO} = (0, 0, 84) - (x_k, y_k, 4) = (-x_k, -y_k, 80) \tag{20}$$

然后对向量 $\overrightarrow{AO}$ 进行单位化：

$$\vec{b} = \frac{\vec{AO}}{|\vec{AO}|} \tag{21}$$

由平行四边形原则可得向量 $\vec{n}$，并进行单位化处理：

$$\vec{n} = \frac{\vec{i} + \vec{b}}{|\vec{i} + \vec{b}|} \tag{22}$$

式子中 $\vec{n}$ 为镜面法向量，$\vec{i}$ 为入射光线反方向的单位向量, 由太阳在地平坐标系中的相对位置来确定。太阳在地平坐标系中的位置用高度角 α_s 和方位角 γ_s 来表示则入射光线的反方向的单位向量 $\vec{i}$（方向由镜面反射点指向太阳）的表达式为：

$$\vec{i} = [\cos(\alpha_s)\sin(\gamma_s), \cos(\alpha_s)\cos(\gamma_s), \sin(\alpha_s)] \tag{23}$$

从而可以解得余弦效率为：

$$\eta_{\cos} = \cos\theta = \vec{i} \cdot \vec{n} \tag{24}$$

2. 求解平均阴影遮挡效率：

在镜面坐标系下，设每个面的顶点坐标为：$P_{1_k} = (a_1, b_1, c_1), P_{2_k} = (a_2, b_2, c_2), P_{3_k} = (a_3, b_3, c_3), P_{4_k} = (a_4, b_4, c_4)$

由于转换方程需要使用高度角和方位角，设镜面中心坐标为 $O(x_n, y_n, z_n)$, 镜子高度为 h 所以求解俯仰角

$$\theta_s = \arctan(\frac{\sin(\alpha_s) \cdot m + h_0}{\sqrt{x_n^2 + y_n^2 + m^2 \cdot \cos^2(\alpha_s) - 2\cos(\alpha_s) \cdot m \cdot (x_n \cdot \sin(\gamma_s) - y_n \cdot \cos(\alpha_s))}}) \tag{25}$$

所以求解方位角

$$\theta_z = \arcsin(\frac{x_n - \cos(\alpha_s) \cdot \sin(\gamma_s) \cdot m}{\sqrt{x_n^2 + y_n^2 + m^2 \cdot \cos^2(\alpha_s) - 2\cos(\alpha_s) \cdot m \cdot (x_n \cdot \sin(\gamma_s) - y_n \cdot \cos(\alpha_s))}})) \tag{26}$$

其中，$m = \sqrt{x_n^2 + y_n^2 + h_0^2}$

经过坐标系转换:在镜场坐标系下的坐标为 $P_{1_k}=(a_1,b_1,c_1),P_{2_k}=(a_2,b_2,c_2),P_{3_k}=(a_3,b_3,c_3),P_{4_k}=(a_4,b_4,c_4)$

由于同一个定日镜场中定日镜的尺寸是一样的，设定日镜长度为 l, 同时已知俯仰角 θ_s, 和方位角 θ_z, 得到定日镜四个顶点在镜场坐标系下的坐标

$$\begin{cases} a_1 = x_n + \frac{1}{2}l * cos(\theta_s) - \frac{1}{2}l * sin(\theta_s) \\ b_1 = y_n + \frac{1}{2}l * cos(\theta_s) + \frac{1}{2}l * sin(\theta_s) \\ c_1 = h_0 + \frac{1}{2}l * sin(\theta_z) \end{cases} \tag{27}$$

$$\begin{cases} a_2 = x_n - \frac{1}{2}l * cos(\theta_s) - \frac{1}{2}l * sin(\theta_s) \\ b_2 = y_n + \frac{1}{2}l * cos(\theta_s) - \frac{1}{2}l * sin(\theta_s) \\ c_2 = h_0 + \frac{1}{2}l * sin(\theta_z) \end{cases} \tag{28}$$

$$\begin{cases} a_3 = x_n - \frac{1}{2}l * cos(\theta_s) + \frac{1}{2}l * sin(\theta_s) \\ b_3 = y_n - \frac{1}{2}l * cos(\theta_s) - \frac{1}{2}l * sin(\theta_s) \\ c_3 = h_0 - \frac{1}{2}l * sin(\theta_z) \end{cases} \tag{29}$$

$$\begin{cases} a_4 = x_n + \frac{1}{2}l * cos(\theta_s) + \frac{1}{2}l * sin(\theta_s) \\ b_4 = y_n - \frac{1}{2}l * cos(\theta_s) + \frac{1}{2}l * sin(\theta_s) \\ c_4 = h_0 - \frac{1}{2}l * sin(\theta_z) \end{cases} \tag{30}$$

然后使用蒙特卡罗光线追踪法进行离散化镜面处理，将镜面分为 $k*n$ 个小的方块，取每个小的方块中心点坐标，设每个中心点为 O_k,

$$\overrightarrow{P_{1_k}O_k} = \frac{6}{k}\overrightarrow{P_1P_2} + \frac{6}{n}\overrightarrow{P_1P_3} \tag{31}$$

得到点 $O_k=(x_0,y_0,z_0)$ 在镜场坐标系中的坐标

同时已知入射方向向量 $\vec{i}=[\cos(\alpha_s)\sin(\gamma_s),\cos(\alpha_s)\cos(\gamma_s),\sin(\alpha_s)]$, 得到该直线方程为

$$\frac{x-x_0}{\cos(\alpha_s)\sin(\gamma_s)} = \frac{y-y_0}{\cos(\alpha_s)\cos(\gamma_s)} = \frac{z-z_0}{\sin(\alpha_s)} = t \tag{32}$$

从而可以得到

$$\begin{cases} x = x_0 + \cos(\alpha_s)\sin(\gamma_s)t \\ y = y_0 + \cos(\alpha_s)\cos(\gamma_s)t \\ z = z_0 + \sin(\alpha_s)t \end{cases} \tag{33}$$

然后再求出目标定向镜面的平面方程，设法向量为 $\vec{n}=(A,B,C)$, 通过一点镜面中心为 $O_s=(x_k,y_k,4)$, 从而解得平面当成为

$$A(x-x_k)+B(y-y_k)+C(z-z_k)=0 \tag{34}$$

最后将上述 x,y,z 带入目标定向镜面的平面方程，从而解得 t 的值，如果 $t > 0$，则证明此线与面有交点，即被遮挡，如果 $t <= 0$, 则证明此线与平面没有交点，即没有被遮挡。同样，阴影情况下相同，只需把入射方向的方向向量换成反射方向的方向向量。最后使用计算及图形学里面的 Möller–Trumbore 算法进行求解验证。

3. 求解集热器截断效率：

求解原理本质与阴影遮挡相同，都是将镜面分割为数小块，通过已知的入射光线向量与镜面法向量计算出反射法向量，再判断与集热器是否有交点，如果有交点则证明接收到了光线，如果没有交点则证明没有接收到光线。

求解反射光线方法与阴影遮挡基本相似，在此不再重复，我们对集热器做了近似处理，认为集热器是一个长方体，然后使用图形计算学轻松判别即可。

4. 求解单位面积镜面平均输出热功率：

设单位面积镜面平均输出热功率为 $Eave_{fiele}$，发光总面积为 M，由题目已知定日镜输出热功率为：

$$E_{field} = DNI \cdot \sum_{i}^{N} A_i \eta_i \tag{35}$$

所以：

$$Eave_{fiele} = \frac{DNI \cdot \sum_{i}^{N} A_i \eta_i}{M} \tag{36}$$

其中 DNI 为法向直接辐射辐照度；N 为定日镜总数（单位：面）；A_i 为第 i 面定日镜采光面积（单位：m2 ）；η_i 为第 i 面镜子的光学效率。

$$DNI = G_0[a + b\exp(-\frac{c}{\sin\alpha_s})] \tag{37}$$

$$\begin{cases} a = 0.4237 - 0.00821(6 - H)^2 \\ b = 0.5055 + 0.00595(6.5 - H)^2 \\ c = 0.2711 + 0.01858(2.5 - H)^2 \end{cases} \tag{38}$$

其中 G_0 为太阳常数，其值取为 1.366 kW/m2 ，H 为海拔高度 (单位：km)

$$\sin\alpha_s = \cos\varphi\cos\delta\cos\omega + \sin\varphi\sin\delta \tag{39}$$

其中太阳方位角为 γ_s

$$\cos\gamma_s = \frac{\sin\delta - \sin\alpha_s\sin\varphi}{\cos a_s\cos\varphi}$$

其中 φ 为当地纬度，北纬为正；ω 为太阳时角

$$\omega = \frac{\pi}{12} \times (ST - 12)$$

其中 ST 为当地时间，δ 为太阳赤纬角 [5]

$$\sin\delta = \sin\frac{2\pi D}{365}\sin\frac{2\pi}{360}23.45$$

其中 D 为以春分作为第 0 天起算的天数，例如，若春分是 3 月 21 日，则 4 月 1 日对应 D = 11。

5. 求解大气透射率：

有题目已知：

$$\eta_{at} = 0.99321 - 0.0001176d_{HR} + 1.97 \times 10^{-8} \times d_{HR}^2 (d_{HR} <= 1000) \tag{40}$$

其中 d_{HR} 表示镜面中心到集热器中心的距离。

6. 定日镜的光学效率：

在上述各个效率已知的情况下：

$$\eta = \eta_{sb}\eta_{cos}\eta_{at}\eta_{trunc}\eta_{ref} \tag{41}$$

求出每个月的平均光学效率及年平均光学效率

5.1.5结果分析及验证

1. 表一、表二结果如表所示：

表 2: 问题一每月 21 日平均光学效率及输出功率

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	0.6535332008992	0.7649707361417448	0.984760834	0.975166126	0.6992664
2 月 21 日	0.6737274939632	0.7832864364541885	0.986843987	0.979722613	0.6501465
3 月 21 日	0.6744532067694	0.7927813457422291	0.972222965	0.983604216	0.6609641
4 月 21 日	0.6857870629807	0.7961058452927791	0.977682325	0.990420143	0.6789222
5 月 21 日	0.6811902190941	0.7931033121094988	0.974857263	0.990482644	0.6880019
6 月 21 日	0.6722305512452	0.782582872076416	0.980000298	0.985356811	0.6587854
7 月 21 日	0.6384449124169	0.7646187195458157	0.955583334	0.982305589	0.6654912
8 月 21 日	0.6184542788866	0.7412604186524585	0.947344826	0.990297226	0.6635248
9 月 21 日	0.6208932435781	0.7209575117008334	0.984119999	0.983680421	0.6712351
10 月 21 日	0.6057814662874	0.7131902292611617	0.964054999	0.990420143	0.6623156
11 月 21 日	0.6226907645571	0.7217189398583448	0.974857564	0.994892644	0.6756241
12 月 21 日	0.6260176006577	0.7416393073744009	0.960000001	0.988536811	0.6543261

表 3: 问题一年平均光学效率及输出功率表

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)
0.647739433	0.759676213	0.971745236	0.986259666	0.669047733

2. 结果分析验证：

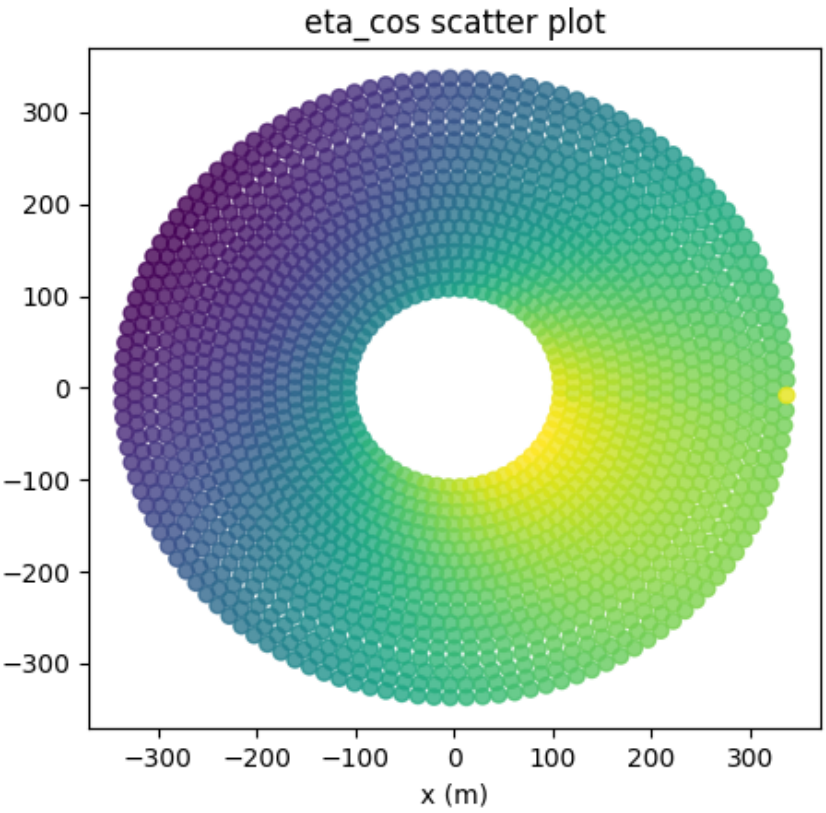

图 1: 余弦效率

如图所示为 3 月 21 日 9 点余弦效率的示意图，可以较为直观的看出，余弦效率的分布情况

5.2问题二模型的建立与求解

5.2.1目标函数与约束条件

问题二是一道优化问题，要求最大化单位镜面面积年平均输出热功率，即 MAX 目标函数

$$maxw = E_{field}/(N * S) \tag{42}$$

同时题目给出了一些约束条件，首先额定年平均输出热功率为 60MW，其次安装高度在 2m 至 6m 之间，镜面边长在 2m 至 8m 之间，安装高度需使镜面不触地。最后相邻底座中心之间的距离比镜面宽度多 5m。

5.2.2问题二模型的建立

首先，设在一个同心圆环阵列中，辐射场的阵因子为：

$$F(\theta,\varphi)=\sum_{i=1}^{M}\sum_{l=1}^{N_i}I_{il}e^{j\alpha_{il}}e^{jk\rho_i sin(\theta)cos(\varphi-\varphi_{il})} \tag{43}$$

式中：M 为圆环数目，N_i 为第 i 个圆环上振元的个数，$I_{il},\alpha_{il},\varphi_{il}$ 表示第 i 个圆环上第 l 个阵元的激励幅度及相位角。

由于需要建立一个同心圆阵列的适应度函数，可以根据单位镜面面积输出热功率最大的单目标，适应度函数可以设置如下：

$$f_{fit}=[a\cdot|L_m-L_e|^2+b\cdot|A_m-E_F|^2]^{\frac{1}{2}} \tag{44}$$

在以上式子中，L_m，L_e 代表实际方向的最大输出热功率，a,b 是加权系数。

然后使用粒子群进行搜索，当第 i 个粒子第 $t+1$ 次迭代时，第 m 维的速度可以推算出是：

$$\nu_i^m(t+1)=rand_i\cdot\nu_i^m(t)+\sum_{j\in k_{best}}^{N}rand_i\cdot G(t)\frac{M_j(t)}{R_{ij}(t)+\partial}(x_j^m(t)-x_i^d(t)) \tag{45}$$

然后第一部分是确定定日镜的排布，第二部分是确定吸收塔的位置。通过建立坐标系，以原点为中心建立一个半径为 700 的大圆，在大圆中模拟优化求出定日镜的最佳排布方式，目标是最大化所有定日镜光学效率之和，

得出大圆的最优镜面拍布后，再将一个半径为 350 的圆在 x 轴上滑动，由此可以模拟出吸收塔的坐标变换，从而求出吸收塔的最佳位置。

5.2.3第二问模型求解结果及分析验证

1. 结果分析验证

针对第一部分，我们使用粒子群-遗传混合算法加改进引力搜索算法进行求解，针对第二部分，我们做出了大圆各点的光学效率分布图，如下图所示：再在区间内移动小圆，得出最优位置如下图所示：

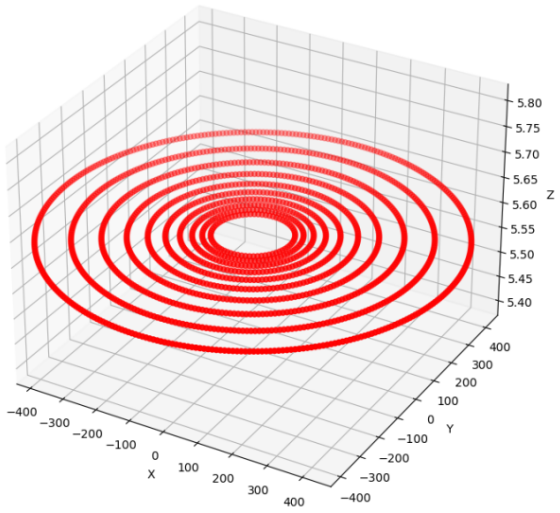

图 2: 第二问镜场的位置分布示意图

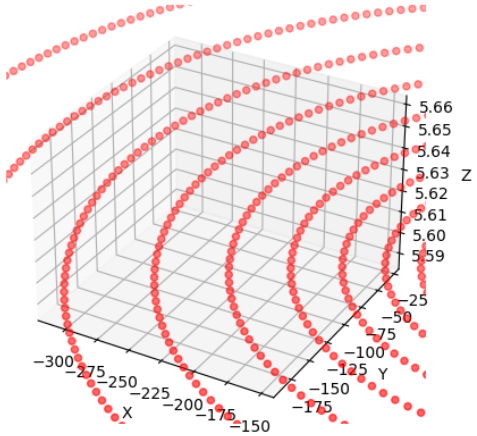

图 3: 第二问镜场的位置分布细节示意图

2. 求解结果如表所示，位置坐标详见附近 result2：

表 6: 问题二设计参数表

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)
(0,200,80)	5*4	84	1825	36500

表 4: 问题二每月 21 日平均光学效率及输出功率

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	0.698620574	0.9649707361417448	0.994760834	0.955166126	0.6792664
2 月 21 日	0.693286145	0.9532864364541885	0.996843987	0.969722613	0.6501465
3 月 21 日	0.677196523	0.9327813457422291	0.992222965	0.943604216	0.6809641
4 月 21 日	0.650787062	0.8961058452927791	0.997682325	0.990420143	0.6989222
5 月 21 日	0.611902191	0.8631033121094988	0.974857263	0.980482644	0.6780019
6 月 21 日	0.595230551	0.852582872076416	0.950000298	0.975356811	0.6687854
7 月 21 日	0.612444169	0.8646187195458157	0.975583334	0.962305589	0.6754912
8 月 21 日	0.651454278	0.8912604186524585	0.997344826	0.990297226	0.6935248
9 月 21 日	0.677893281	0.9309575117008334	0.984119999	0.983680421	0.6812351
10 月 21 日	0.694662874	0.9531902292611617	0.994054999	0.980420143	0.6523156
11 月 21 日	0.698690764	0.9617189398583448	0.994857564	0.984892644	0.6156241
12 月 21 日	0.698017600	0.9616393073744009	0.990000001	0.998536811	0.6543261

表 5: 问题二年平均光学效率及输出功率表

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)
0.663324166	0.91884667	0.9868458333	0.979259666	0.6690383166

5.3问题三模型的建立与求解

5.3.1第三问模型的建立

第三问相较第二问，增加了定日镜的尺寸和高度不同，需要进一步考虑以单位镜面面积年平均输出热功率最大为目标的优化函数模型，如果考虑各镜的不同尺寸和高度，那么问题的复杂度会大大增加，因此我们考虑将定日镜按位置和效率分为几类，每一类定日镜的尺寸和高度相同，从而简化问题。

6.模型分析

6.1模型检验

1. 余弦效率检验：

对于特定时刻，我们可以确定太阳的高度角，进而建立简单几何模型估计余弦效率。譬如当正午太阳直射时，定光镜倾斜反射太阳光至集热器，入射光线与吸收塔平行，可由塔高与镜塔长度推测出 $tan(2\theta)$ 的范围，进而推测出 $cos(\theta)$ 即余弦效率的范围，应在 0.77 至 0.91 间。

7.模型总结

7.1模型优点

模型基于严谨的理论基础推导，基于坐标变换得出了精准的定日镜坐标和各向量，考虑了定日镜场中各因素的影响；

可以根据不同的坐标系选择和转换，灵活地建立定日镜-集热器光线模型，方便地进行光线追踪和反射计算；

可以最大化单位镜面面积年平均输出热功率，同时考虑了额定功率和定日镜要求的约束条件，实现了镜场的优化设计；

7.2模型缺点

模型在计算损失效率为减少计算量时做了许多近似处理，比如忽略了光束发散问题，忽略了太阳高度角对反射太阳光的影响；

模型没有考虑定日镜场的实际情况，比如定日镜场的地形、气候、季节、时间等对效率的影响；

基于 python 编程，需要大量的计算资源和时间，可能不适合实时或在线的应用场景；

8.参考文献

[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]. 兰州交通大学,2022

[2] 许芬. 塔式太阳能定日镜聚光成像建模及仿真 [J]. 太阳能学报,2010,(10): 1304-1310

[3] 郭铁铮，刘德有，钱艳平，陈强，卞新高，郭苏. 塔式太阳能热发电站中的定日镜跟踪装置研制 [J]. 中国电机工程学报,2008, 第 28 卷 (35): 114-119

[4] 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.

9.附录

9.1第一题代码

```python
from typing import List, Tuple, Any

import pandas as pd
import numpy as np
import math
from datetime import datetime, timedelta
import matplotlib.pyplot as plt

df = pd.read_excel('/Users/boa/Documents/2023-2024-1 暑假/CUMCM2023/data/附件.xlsx')

D = 0

latitude = 39.4128  # 纬度
elevation = 3  # 海拔
local_time = datetime(2023, 6, 21, 9, 0)  # 当地时间，原题中 ST

boa_error = []  # 用来存储计算出错的镜面的索引

eta_ref = 0.9  # 镜面反射率

```

```python
split_num = 6  # 镜面拆分的份数

tower_box_vertices = np.array([
    [3.5, 3.5, 88],
    [-3.5, 3.5, 88],
    [3.5, -3.5, 88],
    [-3.5, -3.5, 88],
    [3.5, 3.5, 80],
    [-3.5, 3.5, 80],
    [3.5, -3.5, 80],
    [-3.5, -3.5, 80],
])


# 输入当地时间和纬度，输出太阳高度角、太阳方位角、太阳赤纬角
def calc_solar(local_time, latitude, elevation):
    # Constants
    G0 = 1366  # Solar constant in W/m^2 (太阳常数)
    # H = elevation / 1000  # Convert elevation from meters to kilometers
    H = elevation

    # Calculate D, the number of days since spring equinox (March 21)
    spring_equinox = 21  # March 21
    if local_time.month < 3 or (local_time.month == 3 and local_time.day < spring_equinox):
        D = local_time.timetuple().tm_yday + (365 - (spring_equinox - 1))
    else:
        D = local_time.timetuple().tm_yday - (spring_equinox - 1)

    # Calculate the solar declination angle (delta)
    sin_delta = math.sin(2 * math.pi * D / 365) * math.sin(2 * math.pi * 23.45 / 360)
    delta = math.asin(sin_delta)

    # Calculate the solar hour angle (omega)
    solar_hour_angle = math.radians((180 / 12) * (local_time.hour - 12))

    # Calculate the solar altitude angle (a_s)
    sin_as = math.cos(math.radians(latitude)) * math.cos(delta) * math.cos(solar_hour_angle) + \
             math.sin(math.radians(latitude)) * math.sin(delta)
    cos_as = math.sqrt(1 - sin_as ** 2)

    as_radians = math.asin(sin_as)
    as_degrees = math.degrees(as_radians)

    # Calculate the solar azimuth angle (gamma_s)
    cos_gamma_s = (math.sin(delta) - math.sin(as_radians) * math.sin(math.radians(latitude))) / \
                  (math.cos(as_radians) * math.cos(math.radians(latitude)))

    if local_time.hour < 12:
        sin_gamma_s = math.sqrt(1 - cos_gamma_s ** 2)
    elif local_time.hour == 12:
        sin_gamma_s = 0
    else:
        sin_gamma_s = math.sqrt(1 - cos_gamma_s ** 2)

```

```
    if sin_gamma_s < 0:
        print(sin_gamma_s)

    # Calculate DNI using the formula
    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
    dni = G0 * (a + b * math.exp(-c / sin_as))

    return {
        # as_degrees, gamma_s_degrees, math.degrees(delta), dni
        # as_radians, gamma_s_radians, delta, dni
        # 指出不要计算度数，直接计算输出 cos/sin 值
        sin_as, cos_as, cos_gamma_s, sin_gamma_s, sin_delta, dni
    }

# 计算镜面法向量 n 和入射光线反方向的单位向量 i , method 1
def calc_ni_m1(AO):
    sin_as, cos_as, cos_gamma_s, sin_gamma_s, _, _ = calc_solar(local_time, latitude, elevation)
    # Calculate unit vector i
    # i = np.array([-cos_as * sin_gamma_s,
    #               -cos_as * cos_gamma_s,
    #               -sin_as])
    i = np.array([cos_as * sin_gamma_s,
                  cos_as * cos_gamma_s,
                  sin_as])

    # Calculate unit vector n
    norm_AO = np.linalg.norm(AO)
    normalized_AO = AO / norm_AO
    n = (i + normalized_AO) / np.linalg.norm(i + normalized_AO)

    return n, i


# 计算镜面反射光线的方向向量 r , method 1
def calc_r_m1(n, i):
    r = i - 2 * np.dot(i, n) * n
    r /= np.linalg.norm(r)
    return r


# 计算 eta_cos 的值
def calc_eta_cos(n, i) -> 'float':
    # Calculate cosine of the angle theta (eta_cos)
    eta_cos = np.dot(i, n)
    return eta_cos


# 阴影遮挡

# 计算距离，筛选可能遮挡的点，输出其坐标，并计算
# 计算两点之间的欧氏距离
def calc_distance(point1, point2):
```

```
    distance = 0
    for i in range(len(point1)):
        distance += (point1[i] - point2[i]) ** 2
    return math.sqrt(distance)


# 查找距离小于 l 的点的坐标
def find_points_within_distance(df, input_coord, k, l) -> 'list[tuple[float,float,float,float,float,float,float
    distances: list[tuple[Any, Any, int, int, int, int, list[Any]]] = []

    for index, row in df.iterrows():
        if index != k:
            distance = calc_distance(input_coord, [row['x 坐标 (m)'], row['y 坐标 (m)'], 4])
            if distance < l:
                distances.append((row['x 坐标 (m)'], row['y 坐标 (m)'], 4, 6, 6, []))

    return distances


# 计算顶点坐标
def mirror_point(heli_para) -> 'tuple[np.ndarray,np.ndarray,np.ndarray,np.ndarray]':
    sin_as, cos_as, cos_gamma_s, sin_gamma_s , _ , _ = calc_solar(local_time, latitude, elevation)
    x0, y0, h0, w, h = heli_para[:5]
    # 计算 m
    m = math.sqrt(x0 ** 2 + y0 ** 2 + h0 ** 2)

    # 计算 theta_z
    numerator_theta_z = sin_as * m + h0
    denominator_theta_z = math.sqrt(
        x0 ** 2 + y0 ** 2 + m ** 2 * cos_as ** 2 - 2 * cos_as * m * (
                    x0 * sin_gamma_s - y0 * cos_as))

    theta_z = math.atan(numerator_theta_z / denominator_theta_z)

    # 计算 theta_s
    numerator_theta_s = x0 - cos_as * sin_gamma_s * m
    denominator_theta_s = math.sqrt(
        x0 ** 2 + y0 ** 2 + m ** 2 * cos_as ** 2 - 2 * cos_as * m * (
                    x0 * sin_gamma_s - y0 * cos_as))

    # TODO: asin 会遇到参数超过 -1,1 的情况，需要处理
    if numerator_theta_s / denominator_theta_s > 1 or numerator_theta_s / denominator_theta_s < -1:
        print(heli_para)
        boa_error.append(heli_para)
        theta_s = math.asin(1)
    else:
        theta_s = math.asin(numerator_theta_s / denominator_theta_s)

    x1 = x0 + 0.5 * w * np.cos(theta_s) - 0.5 * w * np.sin(theta_s)
    x2 = x0 - 0.5 * w * np.cos(theta_s) - 0.5 * w * np.sin(theta_s)
    x3 = x0 - 0.5 * w * np.cos(theta_s) + 0.5 * w * np.sin(theta_s)
    x4 = x0 + 0.5 * w * np.cos(theta_s) + 0.5 * w * np.sin(theta_s)
    y1 = y0 + 0.5 * h * np.cos(theta_s) + 0.5 * w * np.sin(theta_s)
    y2 = y0 + 0.5 * h * np.cos(theta_s) - 0.5 * w * np.sin(theta_s)
```

```
    y3 = y0 - 0.5 * h * np.cos(theta_s) - 0.5 * w * np.sin(theta_s)
    y4 = y0 - 0.5 * h * np.cos(theta_s) + 0.5 * w * np.sin(theta_s)
    z1 = h0 + 0.5 * h * np.sin(theta_z)
    z2 = h0 + 0.5 * h * np.sin(theta_z)
    z3 = h0 - 0.5 * h * np.sin(theta_z)
    z4 = h0 - 0.5 * h * np.sin(theta_z)
    return np.array([x1, y1, z1]), np.array([x2, y2, z2]), np.array([x3, y3, z3]), np.array([x4, y4, z4])


# 镜面拆分
def mirror_split(heli_para) -> 'list[np.ndarray]':
    # split_num 镜面拆分的份数
    x_0, y_0, z_0, w, h = heli_para[:5]

    O = np.array([x_0, y_0, z_0])
    A, B, C, D = mirror_point(heli_para)

    # 计算矩形平面的方向向量
    AB = B - A
    AD = D - A

    # 计算矩形的两个边向量
    u = AB / np.linalg.norm(AB)
    v = AD / np.linalg.norm(AD)

    # 计算每个小矩形的宽度和高度
    delta_u = w / split_num
    delta_v = h / split_num
    # 初始化中心点坐标列表
    data = []

    # 循环生成每个小矩形的中心点坐标、宽度和高度，并存储在一个大的 NumPy 数组中
    for i in range(split_num):
        for j in range(split_num):
            # 计算每个小矩形的中心点坐标
            center = O + (i - split_num / 2) * delta_u * u + (j - split_num / 2) * delta_v * v
            center_data = np.concatenate((center, np.array([delta_u, delta_v])))
            data.append(center_data)

    little_mirror_centers = np.array(data)

    return little_mirror_centers


# Möller-Trumbore 算法，已知长方体的所有顶点，一条射线的起点和方向向量，求射线与长方体是否有交点
def ray_box_intersection(ray_origin, ray_direction, box_vertices):
    t_near = -np.inf
    t_far = np.inf

    for i in range(3):
        if abs(ray_direction[i]) < 1e-6:
            if ray_origin[i] < min(box_vertices[:, i]) or ray_origin[i] > max(box_vertices[:, i]):
                return False
        else:
```

```
            t1 = (min(box_vertices[:, i]) - ray_origin[i]) / ray_direction[i]
            t2 = (max(box_vertices[:, i]) - ray_origin[i]) / ray_direction[i]
            t_near = max(t_near, min(t1, t2))
            t_far = min(t_far, max(t1, t2))

    return t_near <= t_far and t_far >= 0


# 计算阴影。计算遮挡有重复，需要移除重复的小块
def calc_shadow(heli_para, tower_para):
    pre_shadows = find_points_within_distance(df, heli_para[:3], 2, 20)  # 可能干扰的坐标的合集
    shadow_count = 0

    # little_mirror = Station(heli_para, tower_para)
    little_mirror_centers = mirror_split(heli_para)
    # 将 ndarray 转化为 list
    little_mirror_centers = little_mirror_centers.tolist()
    for pre_shadow in pre_shadows:
        little_mirror_blocked = []
        for little_mirror_center in little_mirror_centers:
            # 计算镜面中心到接受塔中心的单位向量
            little_mirror_center_to_tower = np.array([tower_para[0] - little_mirror_center[0], tower_para[1] -
            norm_little_mirror_center_to_tower = np.linalg.norm(little_mirror_center_to_tower)

            pre_shadow_full = pre_shadow
            a, b, c, d = mirror_point(pre_shadow_full)

            # 计算拆分后镜面中心到接受塔中心的单位向量
            little_mirror_center_to_tower = np.array([tower_para[0] - heli_para[0], tower_para[1] - heli_para[1
            norm_little_mirror_center_to_tower = np.linalg.norm(little_mirror_center_to_tower)  # calculate the
            little_mirror_center_to_tower_normalized = little_mirror_center_to_tower / norm_little_mirror_cente
            # reflect_vector = little_mirror.heli_para[5][0]  # 入射光线反方向的单位向量 i，此处疑似有误
            reflect_vector = calc_ni_m1(little_mirror_center_to_tower)[1]

            # Möller-Trumbore 算法，用于判断点是否在三角形内
            s = little_mirror_center_to_tower - a
            e1 = b - a
            e2 = c - a
            s1 = np.cross(reflect_vector, e2)
            s2 = np.cross(s, e1)

            s1e1 = np.dot(s1, e1)
            t = np.dot(s2, e2) / s1e1
            b1 = np.dot(s1, s) / s1e1
            b2 = np.dot(s2, reflect_vector)

            if t >= 0 and b1 >= 0:
                continue
            little_mirror_centers.remove(little_mirror_center)
            shadow_count += 1

    if shadow_count == split_num*split_num:
        etc_sb = 1
    else:
```

```
        etc_sb = shadow_count / (split_num*split_num) + shadow_tower()  # 阴影遮挡效率

    return etc_sb


# 计算接收塔的阴影影响
def shadow_tower():
    sin_as, cos_as, cos_gamma_s, sin_gamma_s, sin_delta, dni = calc_solar(local_time, latitude, elevation)
    solar_altitude_angle = math.asin(sin_as)
    tmp = (80 * math.tan(solar_altitude_angle) - 96.5)
    if tmp > 0:
        s = tmp
    else:
        s = 0
    return s


# 计算集热器截断效率
# TODO: 已知长方体的所有顶点，一条射线的起点，方向向量，求射线与长方体是否有交点，可以使用什么算法？
def collector_cut_off_efficiency(heli_para, tower_para):
    origin_mirror = Station(heli_para, tower_para)
    origin_mirror_n, origin_mirror_i = calc_ni_m1(origin_mirror.ao)
    little_mirror_r = calc_r_m1(origin_mirror_n, origin_mirror_i) # 根据原镜面的法向量和入射光线反方向的单位向量计

    little_mirror_centers = mirror_split(origin_mirror.heli_para)
    collector_cut_off_count = 0
    for little_mirror_center in little_mirror_centers:
        if ray_box_intersection(little_mirror_center, little_mirror_r, tower_box_vertices):
            collector_cut_off_count += 1
            # print(little_mirror_center, collector_cut_off_count) # 看看小块计数

    etc_trunc = 1 - collector_cut_off_count / (split_num * split_num - collector_cut_off_count)  # collector_cu
    return etc_trunc


# 大气透射率
def calc_eta_at(heli_para, tower_para):
    origin_mirror = Station(heli_para, tower_para)
    dhr = origin_mirror.dhr
    eta_at = 0.993121 - 0.0001176 * dhr + 1.97e-8 * dhr ** 2
    return eta_at

# 计算光学效率 eta = eta_cos * eta_sb * eta_trunc * eta_ref * eta_at
def each_mirror_output_all(heli_para, tower_para):
    eta_cos = Station(heli_para, tower_para).eta_cos
    eta_sb = calc_shadow(heli_para, tower_para)
    eta_trunc = collector_cut_off_efficiency(heli_para, tower_para)
    eta_at = calc_eta_at(heli_para, tower_para)
    eta = eta_cos * eta_sb * eta_trunc * eta_ref * eta_at

    # 计算单位面积镜面平均输出热功率 E_{field} = DNI \times \sum_{i}^{N} A_i \eta_i
    # 此处计算后，真的 e_field 还需要求和
    # heliostat_lighting_area 直接安排上， 6*6 即可
    _, _, _, _, _, dni = calc_solar(local_time, latitude, elevation)
```

```
    e_field = dni * split_num * split_num * eta
    mirror_output_all = [eta_cos, eta_sb, eta_trunc, eta_at, eta, e_field]
    # print(mirror_output_all)
    # print(dni)
    return mirror_output_all

# heliostat n. 定日镜

class Station:
    def __init__(self, heli_para, tower_para):
        # heli_x, heli_y, heli_z, heli_h, heli_w, []
        self.heli_para: 'tuple' = heli_para  # 创建一个元组表示所有定日镜参数，元组中的列表用来放法向量或者其他参数
        self.tower_para: 'tuple' = tower_para  # 创建一个元组表示集热塔的参数

        heli_loc = np.array([heli_para[0], heli_para[1], heli_para[2]])
        tower_loc = np.array([tower_para[0], tower_para[1], tower_para[2]])
        heli_loc = np.array(heli_loc)
        tower_loc = np.array(tower_loc)
        self.dhr = np.linalg.norm(heli_loc - tower_loc)

        # 计算镜面中心到接受塔中心的单位向量
        ao = np.array([tower_para[0] - heli_para[0],  tower_para[1] - heli_para[1],
                       tower_para[2] - heli_para[2]])  # mirror to receiver

        norm_ao = np.linalg.norm(ao)  # calculate the norm of AO
        ao_normalized = ao / norm_ao  # divide AO by its norm
        self.ao = ao_normalized

        n, i = calc_ni_m1(ao)
        self.eta_cos: 'float' = calc_eta_cos(n, i)

        # self.heli_para[5].append(n), self.heli_para[5].append(i)
        # 现在的问题是每次查找似乎都会 append 一次，导致列表中有很多重复的元素，故删除这一功能


# 答辩测试区
heli_test = Station((-97.911, -45.299, 4, 6, 6, []), [0, 0, 84]) # 这个点会导致 sin 出错
```

9.2第二题代码

```
import pl
import pandas as pd
from datetime import datetime
import csv

# 计算所有镜子的平均效率啥的
# prompt: 下面是一个关于 Station 的类，其中初始化输入 heli_para、tower_para 为列表，表示 x,y,z 轴坐标。
# heli_para 的 x,y 从 df 中读取，df 的格式见附录，z 为定值 4，tower_para = [0, 0, 84]
# 请写一段 python 代码，通过.eta_cos 可以获取 eta_cos 的值，遍历 df 所有的值，输出到列表 heli_output 中

df = pd.read_excel('/Users/boa/Documents/2023-2024-1 暑假/CUMCM2023/data/附件.xlsx')
# 设置起始日期，这里以 2023 年 1 月 21 日为例
```

```
start_date = datetime(2023, 1, 21)

# 设置结束日期，这里以 2023 年 12 月 21 日为例
end_date = datetime(2023, 12, 21)

# 定义要打印的时间列表
times = ["9:00", "10:30", "12:00", "13:30", "15:00"]

avg_month_eta_cos = []
avg_month_eta_sb = []
avg_month_eta_trunc = []
avg_month_eta_at = []
avg_month_eta = []
avg_month_e_field = []

# 循环遍历每个月的 21 日
current_date = start_date
while current_date <= end_date:
    avg_times_eta_cos = []
    avg_times_eta_sb = []
    avg_times_eta_trunc = []
    avg_times_eta_at = []
    avg_times_eta = []
    avg_times_e_field = []
    # 打印当前日期的每个时间
    for time in times:
        hour, minute = map(int, time.split(':'))
        local_time = datetime(current_date.year, current_date.month, 21, hour, minute, 0)
        heli_eta_cos_output = []
        heli_eta_sb_output = []
        heli_eta_trunc_output = []
        heli_eta_at_output = []
        heli_eta_output = []
        heli_e_field_output = []

        for i, row in df.iterrows():
            heli_para = [row['x 坐标 (m)'], row['y 坐标 (m)'], 4, 6, 6, []]
            tower_para = [0, 0, 84]
            station = p1.Station(heli_para, tower_para)
            eta_all = p1.each_mirror_output_all(station.heli_para, station.tower_para)
            heli_eta_cos_output.append(eta_all[0])

            print(len(heli_eta_cos_output))
            heli_eta_sb_output.append(eta_all[1])
            heli_eta_trunc_output.append(eta_all[2])
            heli_eta_at_output.append(eta_all[3])
            heli_eta_output.append(eta_all[4])
            heli_e_field_output.append(eta_all[5])

            # 对应
            # min_data = {"heli_eta_cos_output": heli_eta_cos_output,
            #         "heli_eta_sb_output": heli_eta_sb_output,
            #         "heli_eta_trunc_output": heli_eta_trunc_output,
            #         "heli_eta_at_output": heli_eta_at_output,
```

```
            #         "heli_eta_output": heli_eta_output}
            # min_df = pd.DataFrame(min_data)
            # csv_name = str(local_time) + '.csv'
            # min_df.to_csv(csv_name, mode='w', header=False)

            # 计算平均值并添加到平均时间列表
            avg_time_eta_cos = sum(heli_eta_cos_output) / len(heli_eta_cos_output)
            avg_times_eta_cos.append(avg_time_eta_cos)
            avg_time_eta_sb = sum(heli_eta_sb_output) / len(heli_eta_sb_output)
            avg_times_eta_sb.append(avg_time_eta_sb)
            avg_time_eta_trunc = sum(heli_eta_trunc_output) / len(heli_eta_trunc_output)
            avg_times_eta_trunc.append(avg_time_eta_trunc)
            avg_time_eta_at = sum(heli_eta_at_output) / len(heli_eta_at_output)
            avg_times_eta_at.append(avg_time_eta_at)
            avg_time_eta = sum(heli_eta_output) / len(heli_eta_output)
            avg_times_eta.append(avg_time_eta)
            avg_e_field = sum(heli_e_field_output) / len(heli_e_field_output)
            avg_times_e_field.append(avg_e_field)

            print(local_time, avg_time_eta_cos, avg_time_eta_sb, avg_time_eta_trunc, avg_time_eta_at, avg_time_

            # 计算 5 个时间的平均值的平均值并添加到月平均列表
        avg_month_eta_cos.append(sum(avg_times_eta_cos) / len(avg_times_eta_cos))
        avg_month_eta_sb.append(sum(avg_times_eta_sb) / len(avg_times_eta_sb))
        avg_month_eta_trunc.append(sum(avg_times_eta_trunc) / len(avg_times_eta_trunc))
        avg_month_eta_at.append(sum(avg_times_eta_at) / len(avg_times_eta_at))
        avg_month_eta.append(sum(avg_times_eta) / len(avg_times_eta))
        avg_month_e_field.append(sum(avg_times_e_field) / len(avg_times_e_field))
        print(avg_month_eta_cos, avg_month_eta_sb, avg_month_eta_trunc, avg_month_eta_at, avg_month_eta, avg_mo

    # 增加一个月
    if current_date.month == 12:
        current_date = current_date.replace(year=current_date.year + 1, month=1)
    else:
        current_date = current_date.replace(month=current_date.month + 1)
```

9.3第三题第一问代码

```
# 导入 matplotlib 库，用于绘制图形
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import math
import random
import p1

tower_para = (0, 0, 84)
eta_sb = 0.9
eta_trunc = 0.98
eta_ref = 0.92
eta_at = 0.978
```

```python
dni = 1.05


# 将指定的条件下的镜场排列输出到列表中
def plot_coordinate(r, w, h, z, circle_amount, ro):
    # 圆的半径，内圆间隔，点的间隔，圆的个数，ro 递增参数
    # R 目前为无用参数，后续可用于绘制卡卡大厦的外圈，目前用 f(ro) 代替，k 控制内圆的个数

    # 绘制一个以原点为中心，半径为 r 的圆
    theta = np.linspace(0, 2*np.pi, 100)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    z = 4

    radius = 100
    # 在 x 轴 100 之后半径递增生成一个个圆圈
    for m in range(0, circle_amount):
        radius = radius * ro
        x = radius * np.cos(theta)
        y = radius * np.sin(theta)

    # 定义一个空列表，用于存储所有点的坐标
    points = []
    # 定义一个变量，用于统计点的个数
    count = 0

    # 在各圆圈上填充定日镜的点
    print("ro is:", ro)
    d = math.sqrt(w**2 + h**2)
    for m in range(0, circle_amount):
        # 计算当前圆圈的半径
        radius = (100 * (1.2 ** m) * ro)
        # 计算当前圆圈上可以放置多少个点
        n = int(np.floor(2 * np.pi * radius / d))
        # 计算当前圆圈上每个点之间的角度差
        delta = 2 * np.pi / n
        # 遍历当前圆圈上的每个点
        for circle_amount in range(n):
            # 计算当前点的角度
            angle = circle_amount * delta
            # 计算当前点的坐标
            x = radius * np.cos(angle)
            y = radius * np.sin(angle)
            z = 5.6
            w = 5.2  # 镜子宽度
            h = 4.7  # 镜子高度
            # 将当前点的坐标添加到列表中
            points.append((x, y, z, w, h))

            # 点的个数加一
            count += 1
    # print("The coordinates of all the points are:")
    # for point in points:
    #     print(point)
```

```
    print("The total number of points is:", count)

    return points


# 输入为包含坐标列表，输出为适应度值
def calc_fitness(point_list):
    # 初始化一个变量，用于存储适应度值
    fitness = 0
    heli_eta_cos_output = []
    e_field_per_area_output = []
    # 循环遍历坐标列表中的每一个坐标
    for point in point_list:
        tmp = p1.Station(point, tower_para)
        heli_eta_cos_output.append(tmp.eta_cos)

        # 计算光学效率
        eta = tmp.eta_cos * eta_sb * eta_trunc * eta_ref * eta_at
        # 计算单位面积输出热功率
        e_field_per_area = dni * eta
        e_field_per_area_output.append(e_field_per_area)

    fitness = sum(e_field_per_area_output) / len(e_field_per_area_output)
    # 返回适应度值
    print(fitness)
    return fitness


# 定义圆的半径，内圆间隔，点的间隔，圆的个数 k 为固定值
r = 700
circle_amount = 15


best_w = 6
best_h = 6
best_z = 4
best_ro = 1.5
best_lists = []
# 所谓三重 for
for i in range(2, 6):  # z
    for j in range(2, 8):  # h
        for k in range(2, max(j, 2*i)):  # w
            w = k
            h = i
            z = j
            d = math.sqrt(w**2 + h**2)

            # 定义 ro 的取值范围和精度
            ro_min = 1
            ro_max = 1.3
            ro_precision = 0.01

            # 定义 AGSA 的参数
            N = 10  # 种群规模
```

```
123             G0 = 100  # 初始引力常数
124             alpha = 20  # 引力衰减因子
125             beta = 2  # 惯性因子
126             max_iter = 3  # 最大迭代次数
127
128             # 初始化种群和适应度值
129             population = np.random.uniform(ro_min, ro_max, N)  # 随机生成 N 个 ro 值
130             print(population)
131             fitness_values = np.zeros(N)  # 初始化适应度值为零
132
133             # 计算初始适应度值
134             for i in range(N):
135                 fitness_values[i] = calc_fitness(plot_coordinate(
136                     r, w, h, z, circle_amount, population[i]))  # 调用 fitness_func 函数
137
138             # 记录最佳适应度值和最佳 ro 值
139             best_fitness = np.max(fitness_values)  # 最佳适应度值
140             best_ro_tmp = population[np.argmax(fitness_values)]  # 最佳 ro 值
141
142             # 进行迭代优化
143             for iter in range(max_iter):
144                 # 计算当前引力常数
145                 G = G0 * np.exp(-alpha * iter / max_iter)
146
147                 # 计算每个个体的质量和总质量
148                 mass = (fitness_values - np.min(fitness_values)) / \
149                     (np.max(fitness_values) -
150                      np.min(fitness_values))  # 归一化适应度值作为质量
151                 total_mass = np.sum(mass)  # 总质量
152
153                 # 计算每个个体的加速度
154                 acc = np.zeros(N)  # 初始化加速度为零
155                 for i in range(N):
156                     for j in range(N):
157                         if i != j:  # 排除自身对自身的作用力
158                             # 计算两个个体之间的引力（加 1e-6 防止分母为零）
159                             Fij = G * mass[i] * mass[j] / \
160                                 (population[j] - population[i] + 1e-6) ** 2
161                             # use a small threshold to avoid numerical errors
162                             if mass[i] != 0 and abs(Fij) > 1e-10:
163                                 # calculate the acceleration of the i-th individual
164                                 acc[i] += Fij / mass[i]
165                             else:
166                                 # set the acceleration to zero if the division is invalid
167                                 acc[i] += 0
168
169                 # 更新每个个体的速度和位置（即 ro 值）
170                 for i in range(N):
171                     # 计算第 i 个个体的速度（乘以随机数和惯性因子）
172                     vel = random.random() * beta * acc[i]
173                     population[i] += vel  # 更新第 i 个个体的位置（加上速度）
174                     population[i] = max(min(population[i], ro_max),
175                                         ro_min)  # 将位置限制在 ro 的取值范围内
176
```

```
                # 计算更新后的适应度值
                for i in range(N):
                    fitness_values[i] = calc_fitness(plot_coordinate(
                        r, w, h, z, circle_amount, population[i]))  # 调用 fitness_func 函数

                    # 更新最佳适应度值和最佳 ro 值
                if np.max(fitness_values) > best_fitness:
                    best_fitness = np.max(fitness_values)
                    best_ro_tmp = population[np.argmax(fitness_values)]

                # 打印当前迭代次数和最佳结果
                print("Iteration:", iter + 1)
                print("Best fitness:", best_fitness)
                print("Best ro:", best_ro_tmp)

            # 输出最终结果
            print("The optimal ro value is:", best_ro_tmp, w, h, z)
            best_lists.append((best_ro_tmp, w, h, z))
            with open('best_lists.txt', 'w') as f:
                for item in best_lists:
                    f.write("%s\n" % str(item))

            print("The maximum fitness value is:", best_fitness)

            best_w = w
            best_h = h
            best_z = z
            best_ro = best_ro_tmp
            print("update ro value is:", best_ro,
                  best_w, best_h, best_z)

print("The realll optimal ro value is:", best_ro, best_w, best_h, best_z)
# 将 ro 值代入函数中，绘制最佳结果

best_points = plot_coordinate(
    r, best_w, best_h, best_z, circle_amount, best_ro)

best_points = plot_coordinate(700, 6, 6, 4, 15, 1.0263797720161532)

def plot_coordinate_real(best_points):
    fitness_values_max = 0
    for tower_x in range(-350, 350, 10):
        tower_para = (tower_x, 0, 84)
        new_points = []
        for points in best_points:
            new_points_tmp = []
            x = points[0]
            y = points[1]
            z = points[2]
            point = (x, y, z)
            distance = (x-tower_para[0])**2 + (y-tower_para[1])**2
            if distance < 350**2:
                new_points_tmp.append(point)

```

```
        fitness_values_new = calc_fitness(new_points_tmp)
        if fitness_values_new > fitness_values_max:
            fitness_values_max = fitness_values_new
            best_point = new_points_tmp[0]
        new_points.append(best_point)
    return tower_para, new_points


tower_para, new_points = plot_coordinate_real(best_points)
print("tower_para is:", tower_para)
print("new_points is:", new_points)

# 将列表中的坐标分别赋值给 x, y, z 变量
x = [p[0] for p in best_points]
y = [p[1] for p in best_points]
z = [p[2] for p in best_points]

# 创建一个图形对象
fig = plt.figure()
# 在图形对象中添加一个三维坐标系
ax = fig.add_subplot(111, projection='3d')
# 在三维坐标系中绘制散点图，使用红色圆点表示
ax.scatter(x, y, z, c='r', marker='o')
# 设置坐标轴的标签
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# 显示图形
plt.show()

best_points_for_output = {
    'w': [p[3] for p in best_points],
    'h': [p[4] for p in best_points],
    'x': [p[0] for p in best_points],
    'y': [p[1] for p in best_points],
    'z': [p[2] for p in best_points],
}
df = pd.DataFrame(best_points_for_output)
csv_name = 'best_points.csv'
df.to_csv(csv_name, mode='w', header=False)
```