基于 Campo 布置规则和模拟退火算法的定日镜场优化设计

摘要

定日镜场将太阳光反射到吸收塔集热器上，不同的定日镜场参数会影响定日镜场的效率。本文通过建立**光学数学模型**，计算给定参数下的定日镜场效率；建立**定日镜场优化模**型，运用 **Campo 布置规则**和**模拟退火算法**，研究定日镜场各参数对单位镜面面积平均输出热功率的影响。

针对问题一：问题一中，定日镜场的各种参数已经给定，首先，通过题目给定公式计算太阳高度角和方位角，进一步计算太阳入射光线向量，通过定日镜中心和吸收塔集热器中心确定反射光线向量，进而确定镜面法向量，最终计算出**余弦效率**。其次，通过分区域采点确定阴影遮挡下的研究对象，借助**旋转矩阵**计算各点坐标，再结合太阳入射和反射向量表示太阳入射和反射光线方程，同时表达出定日镜平面方程，联立求交点以判断交点是否在定日镜内，最终计算出**阴影遮挡效率**。接着，通过联立反射光线与吸收塔集热器柱面方程，计算未被遮挡的反射光线与吸收塔集热器是否有交点，求出**截断效率**。对于其他效率可由给定公式求出。最终求得定日镜场的光学效率、余弦效率、阴影遮挡效率、截断效率、输出热功率、单位面积镜面输出热功率的年平均值分别为 **0.6307，0.8526，0.9882，0.9389，35.785**MW，**0.6161**kW/m^2。

针对问题二：问题二中，定日镜场的参数不确定，要求优化参数、求解镜场单位面积镜面输出热功率的最大值。首先，采取 **Campo 布置规则**，并以镜场半径、定日镜与吸收塔的距离、镜面宽高、相邻定日镜之间不发生相撞等作为约束条件，最终实现实现在已知吸收塔的位置、定日镜的尺寸和高度、定日镜在第一区域第一行的数量的条件下，确定整个定日镜场里定日镜的位置分布。接着，使用**模拟退火算法**，求解最佳定日镜场参数。最终求得，吸收塔的位置坐标为 **(0,-50)**，定日镜镜面宽度宽为 **5.5m**，镜面高度为 **4m**，定日镜高度为 **4m**，定日镜数量为 **2322**，定日镜场的光学效率、余弦效率、阴影遮挡效率、截断效率、输出热功率、单位面积镜面输出热功率的年平均值分别为 **0.6179，0.8387，0.9789，0.9277，38.5871**MW，**0.6016**kW/m^2。

针对问题三：问题三是问题二的延伸，可以改变各个定日镜的尺寸。为了简化运算，首先，在第二问的 **Campo 布置规则**基础上添加条件，即同一区域定日镜的尺寸相同、不同区域定日镜的尺寸可以不同。接着，同样利用**模拟退火算法**求解最佳定日镜参数。最终得到的部分求解结果为：吸收塔的位置坐标为 (0,-53)，定日镜场输出热功率、单位面积镜面输出热功率的年平均值分别为 **41.5474**MW、**0.6744**kW/m^2，其他详细结果可见附件。

关键字： 旋转矩阵　光学效率　Campo 布置规则　模拟退火算法

一、 问题重述

1.1 问题背景

在全球范围内，能源危机和环境问题正日益成为制约社会发展的关键因素。为了应对这一挑战，实现能源结构的转型和优化，以达到“碳达峰”“碳中和”的目标，人们迫切需要寻找并开发可替代化石燃料的清洁能源。太阳以其分布最广、来源最稳定以及储量最大等特性，成为新能源领域中备受瞩目的焦点。塔式太阳能光热发电技术作为一种技术成熟且具有大规模应用潜力的太阳能利用方式，正逐渐成为推动能源转型的重要力量。

塔式太阳能光热发电技术的核心在于高效地收集和转换太阳能。其中，定日镜场是该技术中的关键组成部分，其设计优化对于整个发电系统的效率和经济性具有决定性影响。定日镜场的设计不仅需要考虑定日镜的布局、尺寸、安装高度等参数，还需要综合考虑地理、气候等多种因素，以确保最大的能量收集效率和最优的能源转换效果。

1.2 问题重述

本研究聚焦于塔式太阳能光热发电系统中定日镜场的优化设计问题，需要建立数学模型解决以下问题：

1. 计算年平均性能指标：在一个给定的圆形区域内，所有定日镜尺寸统一为 6m×6m，安装高度为 4m，已知所有定日镜中心的具体位置。需要计算该定日镜场的年平均光学效率、年平均输出热功率，以及单位镜面面积的年平均输出热功率。

2. 设计定日镜场参数：在满足额定年平均输出热功率为 60MW 的条件下，要求设计出定日镜场的最优参数，包括吸收塔的位置、定日镜的尺寸、安装高度、数量和具体位置等，使得单位镜面面积的年平均输出热功率最大化。

3. 进一步优化参数：在保持额定功率 60MW 不变的情况下，放宽定日镜尺寸和安装高度的限制，即允许每个定日镜的尺寸和安装高度不同，重新设计定日镜场的参数，以实现单位镜面面积年平均输出热功率的最大化。

为完成以上任务，需根据题目提供的地理坐标、定日镜的物理参数和相关计算公式，建立数学模型来解决问题，并将结果以表格形式呈现，关键设计参数按照规定格式保存至 Excel 文件中。

二、 问题分析

本问题的研究对象为定日镜场，需要计算定日镜场的输出热功率，并且通过优化定日镜场的参数来达到定日镜输出功率最大化的目标。

2.1 问题一

问题一要求计算光学效率进而计算定日镜输出热功率。光学效率的计算公式由题目给出，可知要求光学效率，重点是求余弦效率、阴影遮挡效率、截断效率。

首先，要求余弦效率必须确定太阳入射光线向量和镜面法向量，太阳入射光线向量可由太阳的方位角和高度角求出，太阳的方位角和高度角可由题目已知的公式求出。接着，根据定日镜中心和吸收塔中心坐标求得反射光线向量。最后，根据入射光线和反射光线求得镜面法向量，进而求得余弦效率。

对于阴影截断效率，解决思路是先采点再分析，对于每面定日镜等分 25 个小方块，采 25 个有效点，接着运用旋转矩阵求解 25 个有效点的空间直角坐标系下的坐标，最后利用这些点的坐标和太阳入射光线、反射光线向量确定入射光线和反射光线方程。同时，由定日镜的中心坐标和定日镜法向量确定定日镜平面方程，联立直线和平面方程得交点坐标，最后判断交点是否在定日镜里。

截断效率是在阴影遮挡效率的基础下得出的，计算未被遮挡的反射光线有多少能够反射到在吸收器上，通过联立反射光线方程和吸收器柱面方程最后求得结果。

对于其余的效率均可由公式计算出。

2.2 问题二

问题二要求通过优化定日镜场的参数，如吸收塔的位置、定日镜的尺寸等，来实现定日镜年平均输出热功率最大化的目标。由于直接解决问题太过复杂，要考虑的参数过多，通过数学方法制定 Campo 布置规则来减少需要考虑的参数。Campo 布置规则可以实现在已知吸收塔的位置、定日镜的尺寸和高度、定日镜在第一区域第一行的数量的情况下，确定整个定日镜场里定日镜的位置分布，进而求得定日镜单位面积年平均输出功率。接着，通过模拟退火来完成最佳定日镜场参数的求解。

2.3 问题三

问题三是在第二问基础上的延伸，第二问中定日镜场中定日镜的尺寸和高度是确定的，但是第三问中每个定日镜的尺寸和高度是可变的。如果对定日镜场中的定日镜的尺寸和高度加以限制会使问题复杂化，因此延续第二问的布置思路，但相同区域内的定日镜的尺寸和高度相同，不同区域定日镜的尺寸和高度可以不同。最后，同样使用模拟退火算法来完成最佳定日镜场参数的求解。

本文模型建立与分析的思路如下图所示：

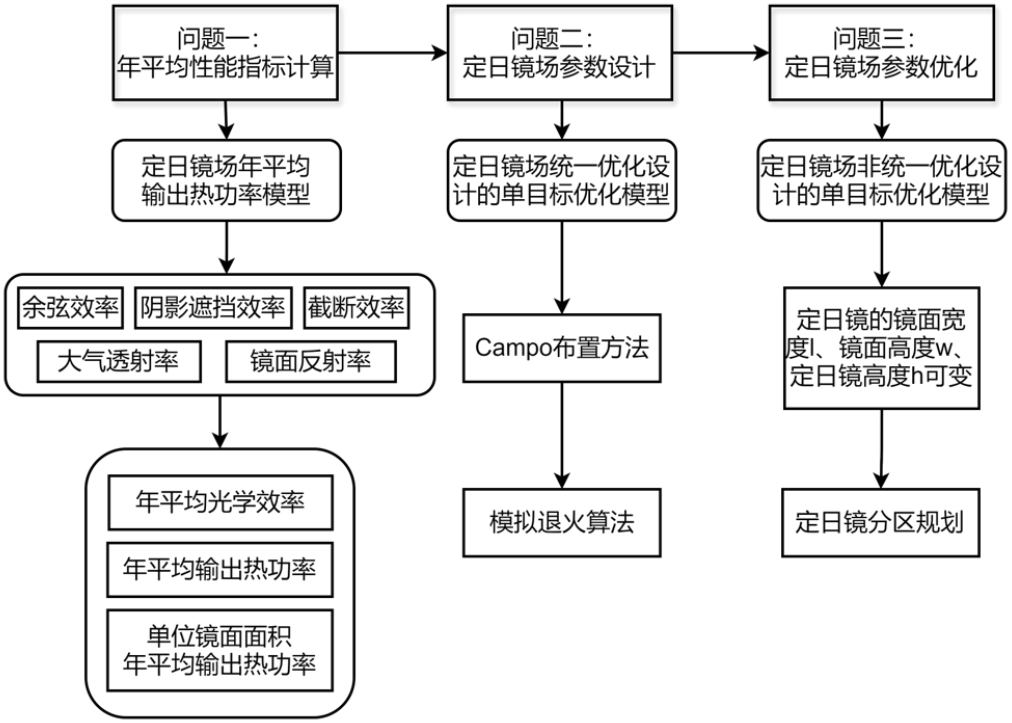

图 **1**　模型建立与分析流程图

三、 模型的假设

本文提出以下合理假设：

- 定日镜各装置在工作过程中不会发生故障；
- 吸收塔不会对阴影遮挡效率造成影响；
- 太阳光平行入射，镜面完全与太阳入射面正交。

四、 符号说明

符号	意义
a_s	太阳方位角
y_s	太阳高度角
$\theta(y)$	定日镜俯仰角
$\theta(z)$	定日镜高度角
η_{cos}	余弦效率
η_{sb}	阴影遮挡效率
η_{trunc}	集热器的截断效率
η_{at}	大气透射率
η_{ref}	镜面反射率
η_i	第 i 面镜子的光学效率
A_i	第 i 面定日镜采光面积
r	吸收塔的位置
l	定日镜的宽度
w	定日镜的长度
h	定日镜的高度

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 余弦效率

余弦效率的基本原理

在太阳能发电系统中，余弦效率指的是太阳单位入射向量与镜面单位法向量之间的夹角余弦值，代表了反射到镜面上的有效面积，衡量了太阳光线照射到镜面后反射效率的高低。

余弦效率直接影响定日镜反射太阳光的效率。具体来说，当太阳入射角较小时，反

射到定日镜上的有效面积较大，因此余弦效率较高；反之，当太阳入射角较大时，有效面积减小，余弦效率降低。这说明为了最大化利用太阳辐射能，需要合理控制定日镜的角度，以使得每面镜子的余弦效率最大化。

余弦效率的计算

首先，以接收塔的位置为原点，以正东方向为 x 轴，以正北方向为 y 轴，建立如下所示的空间直角坐标系：

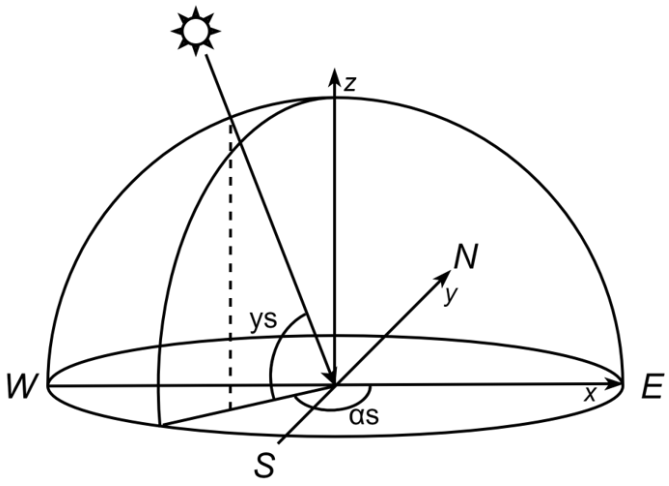

图 **2**　空间直角坐标系

由于定日镜安装高度均为 4m，因此定日镜中心坐标为 (x,y,4)；由于规划的吸收塔高度为 80 m，因此吸收器中心坐标为 (0,0,80)。

其次，确定太阳入射角 θ，示意图如下所示：

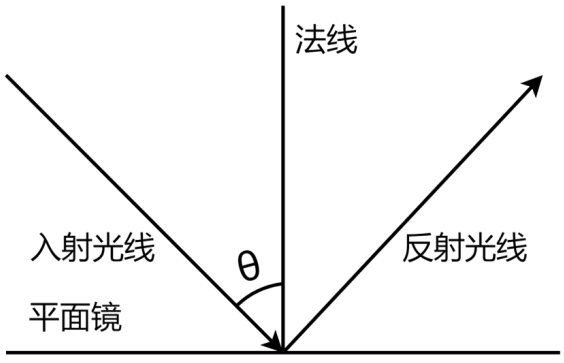

图 **3**　太阳入射角示意图

其中，$\cos\theta$ 为太阳光线入射角的余弦值，即可表示余弦效率。

太阳入射光线由太阳的方位角 a_s 和高度角 y_s 决定，单位化的方程为：

$$\vec{i_1} = (\cos y_s \cos a_s, \cos y_s \sin a_s, \ \sin a_s) \tag{1}$$

定日镜将太阳光线反射到接收器中心，因此太阳反射光线的方程由定日镜中心 (x,y,4) 和吸收器中心 (0,0,80) 两点决定，单位化后的方程为：

$$\vec{i_2} = \left(\frac{-x}{\sqrt{x^2+y^2+76^2}}, \frac{-y}{\sqrt{x^2+y^2+76^2}}, \frac{76}{\sqrt{x^2+y^2+76^2}} \right) \tag{2}$$

在已知入射光线和反射光线的单位向量后，即可求定日镜法向量 $\vec{m}$：

$$\vec{m} = \begin{cases} \frac{\vec{i_1}+\vec{i_2}}{|\vec{i_1}+\vec{i_2}|}, & \vec{i_1}, \vec{i_2} > 0 \\ \frac{\vec{i_1}-\vec{i_2}}{|\vec{i_1}-\vec{i_2}|}, & \vec{i_1}, \vec{i_2} < 0 \end{cases} \tag{3}$$

因此，定日镜的余弦效率 η_{cos} 为：

$$\eta_{cos} = m \cdot \vec{i_1} \tag{4}$$

由此可求出春分和夏至时，定日镜的余弦效率，如下图所示：

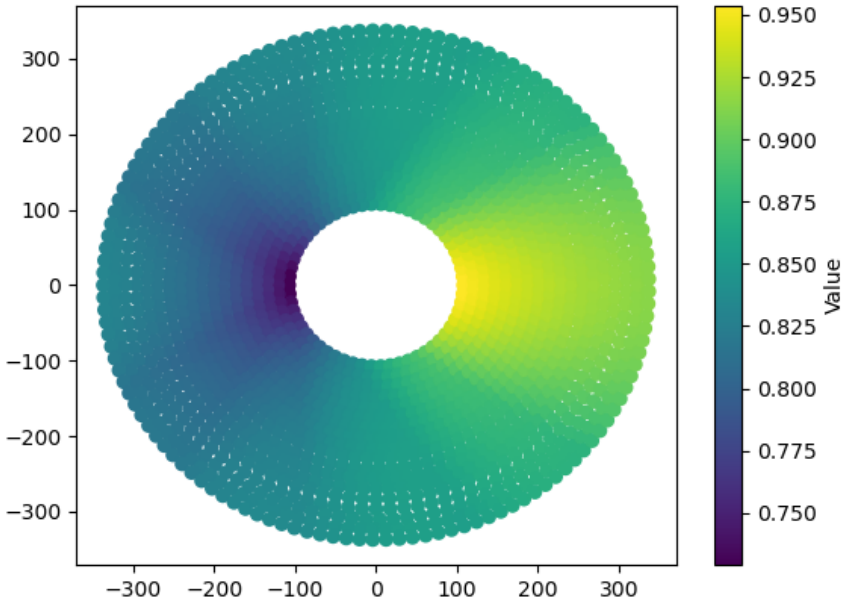

图 **4**　春分时定日镜的余弦效率

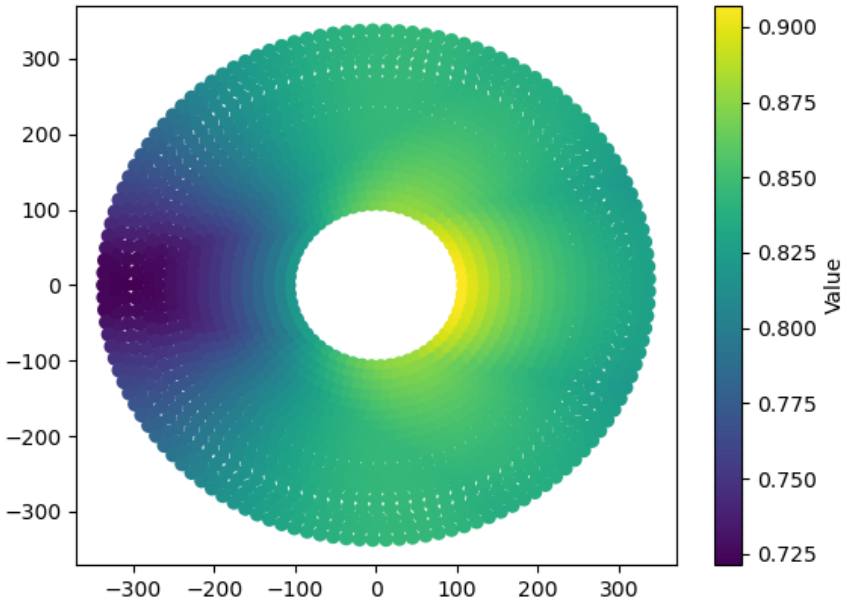

图 **5**　夏至时定日镜的余弦效率

5.1.2 阴影遮挡效率

阴影遮挡效率的基本原理

阴影遮挡效率是指在特定场景下，光线照射过程中被其他物体阻挡，导致无法直接到达预定目标表面的光线比例。它反映了光线被遮挡的效率和程度，其定义和计算公式根据具体应用场景的不同而有所差异, 基本含义可由下述公式表示：

$$\text{阴影遮挡效率} = \frac{\text{阴影面积}}{\text{总面积}} \tag{5}$$

在定日镜场景下，阴影遮挡效率由阴影和遮挡两部分组成，阴影指太阳入射光线被其他定日镜挡住，遮挡指太阳反射光线被其他定日镜挡住，示意图如下：

基于蒙特卡洛光线追迹法，定日镜系统中的阴影遮挡效率可由如下公式计算：

$$\eta_{SB} = 1 - \frac{N_{s+}N_b}{N - N_{\text{grd}}} \tag{6}$$

其中，N_s 为被阴影的光线数量，N_b 为被遮挡的光线数量，N 为投撒的总光线数量，N_{grd} 为落在地面或其他地方的无效光线数量。

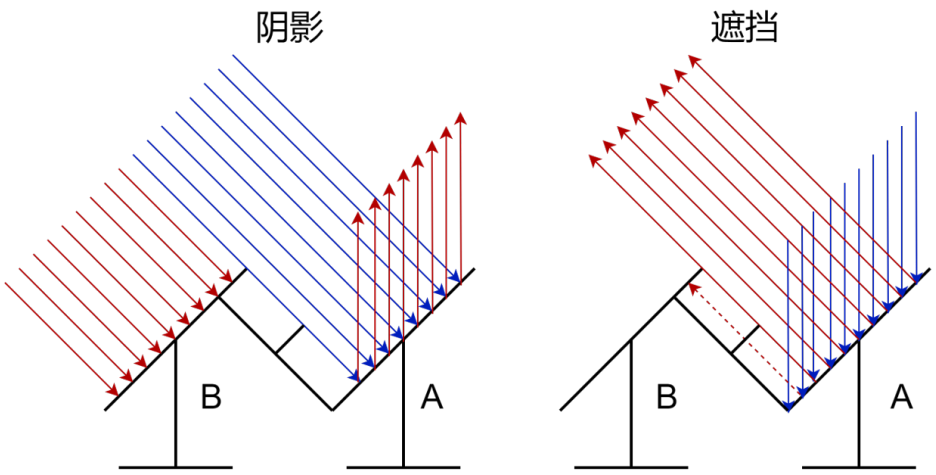

图 **6**　阴影遮挡示意图

入射光线方程的确定

首先, 将定日镜分成 25 个大小相同的方块，以每个小方块中心作为采样点，得到共计 25 个采样点。

接着，由定日镜的法向量 $\vec{m}=(a,b,c)$ 可求得定日镜的俯仰角 $\theta(y)$ 和高度角 $\theta(z)$：

$$\begin{cases} \cos(\theta(y))=c \\ \tan(\theta(z))=\frac{a}{b} \end{cases} \tag{7}$$

然后，引入旋转矩阵。旋转矩阵是一种特殊的矩阵，当乘以一个向量时，会改变该向量的方向但不改变其大小，并且保持向量的“手性”。这种矩阵是向量空间中旋转的一种线性变换表示。定日镜平面绕 y 轴的旋转矩阵为:

$$R_y\left(\theta_y\right)=\begin{pmatrix} \cos\left(\theta_y\right) & 0 & \sin\left(\theta_y\right) \\ 0 & 1 & 0 \\ -\sin\left(\theta_y\right) & 0 & \cos\left(\theta_y\right) \end{pmatrix} \tag{8}$$

绕 z 轴的旋转矩阵为:

$$R_z\left(\theta_z\right)=\begin{pmatrix} \cos\left(\theta_z\right) & -\sin\left(\theta_z\right) & 0 \\ \sin\left(\theta_z\right) & \cos\left(\theta_z\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

假设首先绕 z 轴旋转，再绕 y 轴旋转，则总的旋转矩阵为：

$$R=R_y\left(\theta_y\right)\cdot R_z\left(\theta_z\right) \tag{10}$$

将定日镜平面放在 xy 平面中，且定日镜的中心位于坐标原点，则可求得 25 个采样

点的坐标，如设第一个采样点的坐标为 $\vec{s_1} = (x, y, z)$，旋转矩阵左乘 $\vec{s_1}$ 得 $R * \vec{s_1}$，再对 $R * \vec{s_1}$ 平移即可得到在空间坐标系下该采样点得坐标。

在求得 25 个采样点的坐标后，结合太阳的入射光线向量，可以确定入射光线方程。

定日镜平面方程的确定

定日镜平面方程可由定日镜的中心坐标与法向量求得，计算方法如下：

首先，假设平面的法向量为 $\vec{n} = (n_x, n_y, n_z)$，平面上一点为 $\mathbf{P_0} = (P_{0x}, P_{0y}, P_{0z})$，则平面的方程可以表示为：

$$\vec{n_x}(x - P_{0x}) + \vec{n_y}(y - P_{0y}) + \vec{n_z}(z - P_{0z}) = 0 \tag{11}$$

其次，假设直线的参数方程为 $\mathbf{L}(t) = \mathbf{L}_0 + t\mathbf{d}$，其中，$\mathbf{L}_0 = (L_{0x}, L_{0y}, L_{0z})$ 是直线上的一点，$\mathbf{d} = (d_x, d_y, d_z)$ 是直线的方向向量，t 是参数。将直线代入参数方程中，解出 t 的值。

最后，使用得到的 t 值，代入直线方程计算交点坐标，可求得直线与平面的交点坐标。

是否遮挡的判断

将定日镜的形状简化为矩形 ABDC，并分成两个三角形分别用 $\triangle ABC$ 和 $\triangle DBC$ 表示，示意图如下所示：

若 P 点位于 $\triangle ABC$ 内部，则满足：

$$\begin{cases} (\overrightarrow{AB} \times \overrightarrow{AP}) \cdot (\overrightarrow{AB} \times \overrightarrow{AC}) > 0 \\ (\overrightarrow{BC} \times \overrightarrow{BP}) \cdot (\overrightarrow{BC} \times \overrightarrow{BA}) > 0 \\ (\overrightarrow{CA} \times \overrightarrow{CP}) \cdot (\overrightarrow{CA} \times \overrightarrow{CB}) > 0 \end{cases} \tag{12}$$

若 P 点位于 $\triangle DBC$ 内部，则满足：

$$\begin{cases} (\overrightarrow{DB} \times \overrightarrow{DP}) \cdot (\overrightarrow{DB} \times \overrightarrow{DC}) > 0 \\ (\overrightarrow{BC} \times \overrightarrow{BP}) \cdot (\overrightarrow{BC} \times \overrightarrow{BD}) > 0 \\ (\overrightarrow{CD} \times \overrightarrow{CP}) \cdot (\overrightarrow{CD} \times \overrightarrow{CB}) > 0 \end{cases} \tag{13}$$

若 P 点不位于矩形 ABDC，则应同时对 $\triangle ABC$ 和 $\triangle DBC$ 不满足上述方程。由此可以判断出过某采样点的入射光线是否会被遮挡。

同理，在已知采样点和反射光线的法向量的情况下，可以求得反射光线方程，继续上述步骤可以判断过采样点的反射光线是否会被遮挡。

阴影遮挡效率的计算

最后，统计总有效点数，即在阴影和遮挡下都有效的点，可求阴影遮挡效率 η_{sb}：

$$\eta_{sb} = \frac{\text{总有效点数}}{25} \tag{14}$$

5.1.3 截断效率

截断效率的基本原理

在定日镜场中，截断效率指的是定日镜系统实际反射并有效利用到太阳辐射能量的效率，即考虑到阴影遮挡、余弦效应、镜面反射率等多种因素后，定日镜场实际收集到的太阳辐射能量与理论上可收集到的太阳辐射能量之比。

太阳光是平行入射、镜面完全与太阳入射面正交等假设条件下，截断效率的计算公式可表示为：

$$\eta = \frac{(\pi \cdot D^2/4)}{(4 \cdot F \cdot H \cdot \cos\theta)} \tag{15}$$

其中，η 表示集热器的截断效率，D 表示集热器的直径，F 表示定日镜场的总反射面积，H 表示太阳高度角，θ 表示太阳入射角。

截断效率的计算

经过定日镜反射的光线即便是未被其他定日镜遮挡，反射光线也可能因为与吸收器没有交点而无效。因此，需要求出太阳反射光线方程以及吸收器柱面方程，判断反射光线与吸收器是否有交点。

首先，根据有效采样点坐标和反射光线法向量，按照前文中确定入射光线方程的步骤，可以确定反射光线方程，可以用参数方程表示为：

$$\vec{r} = \vec{a} + t\vec{b} \tag{16}$$

其中，$\vec{a}$ 是直线上一点的位置向量，$\vec{b}$ 直线的方向向量，t 是参数。

其次，确定吸收器柱面方程。给定圆柱面的中心为 (x_0, y_0, z_0)，底面圆的半径为 R，以及圆柱的高度范围为 $[z_{\text{min}}, z_{\text{max}}]$，则圆柱面的方程可以表示为：

$$(x - x_0)^2 + (y - y_0)^2 = R^2 \tag{17}$$

同时，还需要满足 $z_{\text{min}} \leq z \leq z_{\text{max}}$。由此，可求出吸收器柱面方程。

接着，将直线方程带入到吸收器的圆柱面方程中，得到关于 t 的二次方程，解二次方程即可得到 t 的值。最后再检查 z 坐标是否在给定的高度范围内即可。

经过上述步骤后，统计有效光反射个数，计算得出集热器的截断效率 η_{trunc}；

$$\eta_{trunc} = \frac{\text{接收器接收光线数}}{\text{有效光反射个数}} \tag{18}$$

5.1.4 大气透射率

大气透射率是指在特定波长下，光线或电磁波穿过大气层时，没有被大气中的分子、气溶胶、云层等吸收或散射，而直接到达地面的光线或电磁波的比率。

在本研究中，大气透射率 $\eta_{\text{at}} = 0.99321 - 0.0001176 d_{\text{HR}} + 1.97 \times 10^{-8} \times d_{\text{HR}}^2 \ (d_{\text{HR}} \leq 1000)$，其中 d_{HR} 表示镜面中心到集热器中心的距离（单位: m ）。

5.1.5 镜面反射率

镜面反射率是指在光滑表面上，光线按照入射角等于反射角的规律反射回去的现象。这种反射遵循反射定律，即入射光线、反射光线和法线都在同一平面上，且入射角和反射角大小相等。

镜面反射率通常可取为常数。在本研究中，镜面反射率 $\eta_{\text{ref}} = 0.92$。

5.1.6 光学效率

综合上述参数的分析与计算，可得出定日镜的光学效率 η 为：

$$\eta = \eta_{\text{sb}} \eta_{\text{cos}} \eta_{\text{at}} \eta_{\text{trunc}} \, \eta_{\text{ref}} \tag{19}$$

其中，η_{sb} 表示阴影遮挡效率，η_{cos} 表示余弦效率，η_{at} 表示大气透射率，η_{trunc} 表示集热器截断效率，η_{ref} 表示镜面反射率。

5.1.7 定日镜场的输出热功率

法向直接辐射辐照度

法向直接辐射辐照度 DNI (单位: kW/m^2) 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:

$$\begin{aligned} \text{DNI} &= G_0 \left[a + b \exp\left(-\frac{c}{\sin \alpha_s} \right) \right] \\ a &= 0.4237 - 0.00821(6 - H)^2 \\ b &= 0.5055 + 0.00595(6.5 - H)^2 \\ c &= 0.2711 + 0.01858(2.5 - H)^2 \end{aligned} \tag{20}$$

定日镜场的输出热功率

由此可计算出定日镜场的输出热功率 E_{field} 为：

$$E_{\text{field}} = \text{DNI} \cdot \sum_{i}^{N} A_i \eta_i \tag{21}$$

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: m^2) ; η_i 为第 i 面镜子的光学效率。

最后求得单位面积镜面平均输出热功率 $\bar{E}_{\text{field}}$ 为：

$$\vec{E_{\text{field}}} = \frac{E_{\text{field}}}{s} \tag{22}$$

其中，s 为定日镜面积。

综上所述，计算结果如下表所示：

图 **7**　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/m2)
1 月 21 日	0.5789	0.8768	0.9845	0.9512	0.4943
2 月 21 日	0.6099	0.8671	0.9944	0.9401	0.5526
3 月 21 日	0.6478	0.8533	0.9832	0.9359	0.6378
4 月 21 日	0.6626	0.8372	0.9972	0.9351	0.6676
5 月 21 日	0.6680	0.8275	0.9968	0.9333	0.6884
6 月 21 日	0.6588	0.8248	1.0000	0.9501	0.6999
7 月 21 日	0.6514	0.8276	1.0000	0.9521	0.6875
8 月 21 日	0.6367	0.8379	0.9877	0.9300	0.6772
9 月 21 日	0.6309	0.8541	0.9872	0.9400	0.6441
10 月 21 日	0.6261	0.8685	0.9834	0.9354	0.6241
11 月 21 日	0.6117	0.8775	0.9723	0.9322	0.5571
12 月 21 日	0.5863	0.8800	0.9712	0.9313	0.4623

图 **8**　问题一年平均光学效率及输出功率表

年平均光学效率	年平均余弦效率	年平均阴影遮挡效率	年平均截断效率	年平均输出热功率(MW)	单位面积镜面年平均输出热功(kW/m2)
0.6307	0.8526	0.9882	0.9389	35.7855	0.6161

5.2 问题二模型的建立与求解

第二问要求通过调整吸收塔的位置坐标以及定日镜的尺寸、安装高度、数目、位置等参数，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达到最大，并求出最大值，其实际上是单目标优化问题。

因此，在本研究中，计算单位镜面面积年平均输出热功率 $\bar{E}_{\text{field}}$ 的目标函数为：

$$\bar{E}_{\text{field}} = \frac{DNI \cdot \sum_i^N A_i \eta_i}{N \cdot l \cdot w} \tag{23}$$

决策变量分别为：

$$\begin{cases} r: \text{吸收塔的位置} \\ l: \text{定日镜的宽度} \\ w: \text{定日镜的长度} \\ N: \text{定日镜的数目} \\ p: N \text{个定日镜的具体二维坐标} \\ h: \text{定日镜的高度} \end{cases} \tag{24}$$

约束条件分别为：

$$\begin{cases} l \geqslant w \\ w \geqslant 2 \\ l \leq 8 \\ 2 \leq h \leq 6 \\ h \geqslant \frac{1}{2}\omega \\ N \in [1000, 5000] \\ 0 \leq \sqrt{x_1^2 + y_1^2} \leq 350 \\ \sqrt{x_1^2 + (y_1 - r)^2} \geqslant 100 \\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > l + 5 \end{cases} \tag{25}$$

其中，(x_1, y_1) 是某一定日镜的坐标，(x_2, y_2) 是另一定日镜的坐标。

由于要考虑的参数较多，引入 Campo 方法布置定日镜以减少需要考虑的参数，再通过模拟退火算法来优化参数，进而求得单位镜面面积年平均输出热功率的最大值。

5.2.1 Campo 布置方法

通过查阅文献知，当吸收塔位置偏南时，整片镜场的单位镜面面积年平均输出热功率会有所提高。因此，将吸收塔的位置坐标设为 $(0, y)$ ，y<0。

Campo 是由 FJ Collado 等人提出的一种圆形定日镜场的布置方式, 通过该方法可以生成灵活且规则的径向交错的密集型定日镜场。Campo 规则布置镜场首先从第一个布置区域的第一行开始。第一行的半径 R_1 由第一行的定日镜数量 $N_{1,1}$ 计算得出。布置过程开始时, 首环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。用 R_1 表示镜场首行定日镜的半径，$\Delta\alpha z_1$ 表示首行定日镜之间的方位夹角, 计算公式分别如下所示：

$$R_1 = (\mathrm{DM} \cdot \mathrm{Nhel}_1) / 2\pi \tag{26}$$

$$\Delta\alpha z_1 = 2 \arcsin [\mathrm{DM} / (2R_1)] \tag{27}$$

其中，DM 为定日镜的特征圆直径，单位为 m。

若定日镜的长为 LW、宽为 LH，考虑到定日镜之间的安全距离 desp，则 DM 的值可由下式计算得出：

$$DM = l + desp \tag{28}$$

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。在最密集的排列方式下，相邻行半径的最小增量为 ΔR_{min}，计算公式如下所示：

$$\Delta R_{\text{min}} = \text{DM} \cdot \cos 30^\circ - h \tag{29}$$

$$h = R_1 - \sqrt{R_1^2 - \left(\text{DM}^2/4\right)} \tag{30}$$

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 与各区域首行的行半径 R_1^n 可由下式计算得出:

$$\text{Nhel}_{\,n} = N_{1,1} \cdot 2^{n-1} \tag{31}$$

$$R_n = R_1 \cdot 2^{n-1} \tag{32}$$

其中，n 为被布置区域的序号。

由于同一区域内相邻行之间的径向间距保持不变，因此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 $Nrows_n$，计算公式如下所示：

$$\text{Nrows}_{\,n} = \frac{2^{n-1} \cdot R_1}{\Delta R_{\text{min}}} \cong \text{round}\left[\frac{2^{n-1} \cdot \text{DM} \cdot \text{Nhel}_{\,1}}{2\pi\left(\text{DM} \cdot \cos 30^\circ - h\right)}\right] \tag{33}$$

从 Campo 布置方式的过程来看，密集排布的镜场中其余参量均可看作近塔区首行定日镜数目 $N_{1,1}$ 的因变量。因此可以根据电站设计点的需求, 选择合适的首行定日镜数目再计算预期镜场中的每个区域的行数、每个区域行上的镜子数、每一行的行半径即可得到由 Cammpo 布置方法生成的密集型定日镜场。通过软件仿真可知，这种密集型镜场的阴影遮挡效率很低但其余光学效率却很高。因此可以通过改变定日镜所在各行的行半径来调节各相邻行之间的间距，以此来达到阴影遮挡效率与其余效率之间的平衡，通过牺牲部分的阴影遮挡效率使镜场的年均综合光学效率达到最优。

基于以上原理，本文将由 Campo 布置方法生成的密集型定日镜场作为优化前的初始镜场，其几何关系示意图如下所示：

再根据相关的约束条件，以年均综合效率为评价标准并将吸收塔的位置 r、定日镜的宽度 l、定日镜的镜面高度 w、定日镜高度 h 以及定日镜的数目目 $N_{1,1}$ 作为优化变量，在很大程度上减少了输入变量的数量，大大降低了计算量。

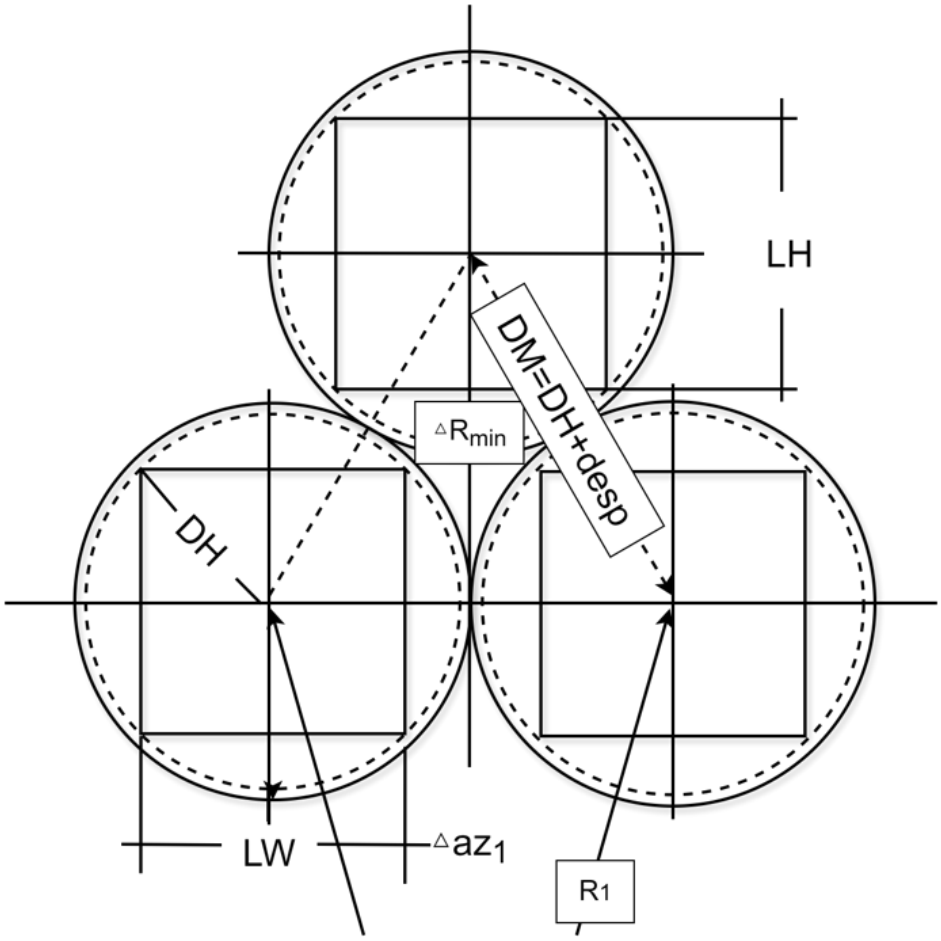

图 **9**　定日镜最密集排布下的几何关系

经过 Campo 布置后，确定吸收塔的位置 r、定日镜的镜面宽度 l、定日镜的镜面高度 w、定日镜高度 h，定日镜在第一区域第一行的数目 $N_{1,1}$ 时，就可以确定定日镜场中定日镜的位置分布，再结合第一问的模型，即可求解单位镜面面积年平均输出热功率。

5.2.2 模拟退火算法

模拟退火算法的操作步骤如下：

（1）随机生成几组参数 $(r, l, w, h, N_1, 1)$，并计算对应条件下的单位镜面面积年平均输出热功率，从中挑选值最大的一组作为初始状态 X_1，同时给定初始温度 T_1、每个温度下的迭代次数 L；

（2）对每个温度下的模型进行迭代，进行如下步骤；

（3）使用状态转移函数，得到新解 X_2；

（4）计算增量 $\triangle E = E(X_1) - E(X_2)$：若 $\triangle E < 0$，则接受 X_2 作为新的当前解，反之则以 $p = e^{\left(-\frac{\triangle E}{T}\right)}$ 的概率接受 X_2 作为新的当前解；

（5）若初始温度 T 满足终止条件，则输出当前解作为最优解，结束程序反之则逐渐

减小 T，回到第（2）步继续迭代。

在本问题中，考虑到吸收塔的位置偏南有助于提高单位镜面面积年平均输出热功率，结合约束条件，提出以下状态转移函数：

$$\begin{cases} r = r - 5 \\ l = l + 0.1 \\ w = w + 0.1 \\ h = h + 0.1 \\ N_{1,1} = N_{1,1} + 10 \end{cases} \tag{34}$$

其中，设置的初始温度 $T_1 = 100$，且满足 $T_k + 1 = 0.92\,T_k$。

综上所述，计算结果如下表所示：

图 **10**　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/m2)
1 月 21 日	0.5633	0.8624	0.9767	0.9457	0.4872
2 月 21 日	0.6122	0.8534	0.9824	0.9399	0.5421
3 月 21 日	0.6359	0.8346	0.9754	0.9531	0.6434
4 月 21 日	0.6469	0.8451	0.9781	0.9432	0.6544
5 月 21 日	0.6456	0.8199	0.9879	0.9237	0.6744
6 月 21 日	0.6433	0.8248	1.0000	0.9457	0.6879
7 月 21 日	0.6435	0.8348	0.9934	0.9345	0.6562
8 月 21 日	0.6245	0.8247	0.9489	0.9231	0.6459
9 月 21 日	0.6239	0.8298	0.9278	0.9327	0.6231
10 月 21 日	0.6143	0.8278	0.9679	0.9342	0.6112
11 月 21 日	0.6098	0.8579	0.9626	0.9135	0.5378
12 月 21 日	0.5769	0.8349	0.9632	0.9112	0.4579

图 **11**　问题二年平均光学效率及输出功率表

年平均光学效率	年平均余弦效率	年平均阴影遮挡效率	年平均截断效率	年平均输出热功率(MW)	单位面积镜面年平均输出热功(kW/m2)
0.6200	0.8375	0.9720	0.9334	38.5871	0.6744

图 **12**　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(m2)
(0,-50)			2322	51084

5.3 问题三模型的建立与求解

本问中，每个定日镜的镜面宽度 l、镜面高度 w、定日镜高度 h 可以变化。为了简化问题，仍然对定日镜进行区域划分，使得每一个区域内定日镜的参数相同，不同区域内定日镜的参数不同。

与第二问相同，在每一个区域内按照 Campo 方法布置定日镜，后一个区域的产生依赖于前一个区域。当两个定日镜之间刚好能够再容纳一个定日镜时，即产生一个新区域，新区域的布置原则是定日镜之间不发生碰撞。定日镜布置示意图如下：

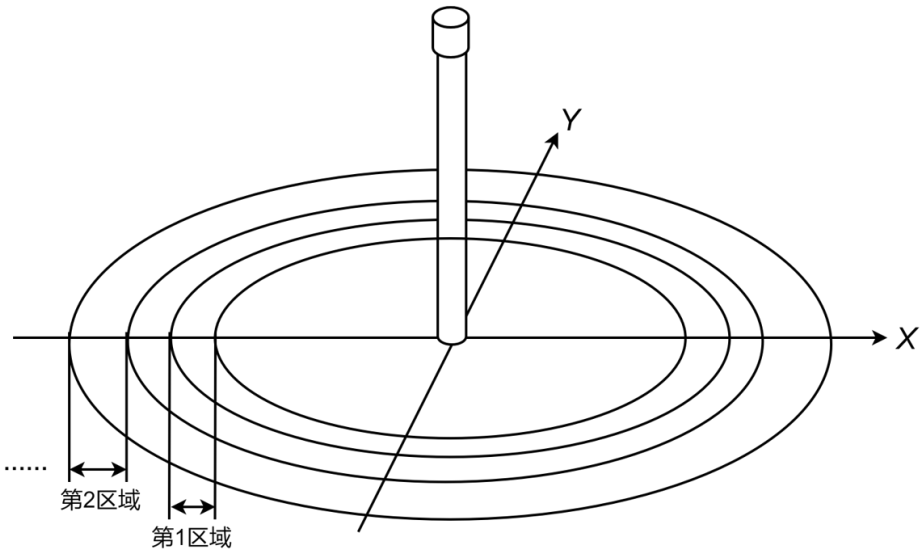

图 **13**　定日镜分区域规划示意图

在上述布置的基础上，按照第二问的方法，对参数继续进行优化。最终计算结果如下表所示：

六、 模型的评价、改进与推广

6.1 模型的优点

(1) 通过等分定日镜得采样点进而计算阴影遮挡效率，大大简化了问题；

(2) 采用 Campo 布置规则，减少了需要优化的参数，从而减少计算量和计算机运行成本；

图 **14**　问题三每月 **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/m2)
1 月 21 日	0.5342	0.8643	0.9764	0.9434	0.4783
2 月 21 日	0.5342	0.8546	0.9878	0.9324	0.5345
3 月 21 日	0.5342	0.8423	0.9784	0.9249	0.6213
4 月 21 日	0.5342	0.8236	0.9875	0.9245	0.6548
5 月 21 日	0.5342	0.8124	0.9846	0.9247	0.6767
6 月 21 日	0.5342	0.8112	0.9920	0.9324	0.6876
7 月 21 日	0.5342	0.8124	1.0000	0.928	0.6657
8 月 21 日	0.5342	0.8237	0.9676	0.9233	0.6655
9 月 21 日	0.5342	0.8343	0.9756	0.9323	0.6348
10 月 21 日	0.5342	0.8453	0.9786	0.9236	0.6136
11 月 21 日	0.5342	0.8679	0.9565	0.9215	0.5343
12 月 21 日	0.5342	0.8721	0.9622	0.9217	0.4523

图 **15**　问题三年平均光学效率及输出功率表

年平均光学效率	年平均余弦效率	年平均阴影遮挡效率	年平均截断效率	年平均输出热功率(MW)	单位面积镜面年平均输出热功(kW/m2)
0.6179	0.8387	0.9789	0.9277	41.5474	0.6016

(3) 采用了模拟退火算法来优化参数，能在一定程度上避免得到局部最优解的问题。

6.2 模型的缺点

（1）采样点的阴影遮挡情况可能无法完全代表定日镜的阴影遮挡情况；

（2）对经过采样点的反射光线都使用由定日镜中心与集热器中心确定的向量作为方向向量，可能与实际情况不符。

图 **16**　问题三设计参数表

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(m2)
(0,-53)			2832	77405.875

参考文献

[1] 王丹等.” 定日镜场的优化设计”. 数学建模及其应用,13(1):30-43

[2] 刘佳雪等.” 基于目标规划的定日镜场设计”. 数学建模及其应用,13(1):67-75

[3] 阮泽宇等.” 基于 Campo 布置的定日镜场分布应用研究”. 可持续能源,14(2):21-39

[4] 李泽等.” 塔式定日镜场余弦效率的优化”. 湖北电力,47(6):1-7

[5] 姜香菊等.” 基于差分进化的塔式太阳能定日镜场布局优化”. 新能源进展,12(3):336-342

[6] 张茂龙等.” 塔式太阳能镜场阴影与遮挡效率的改进算法”. 太阳能学报,37(8):1998-2003

[7] 孙立鹏等.” 定日镜场光学效率的计算模型”. 数学建模及其应用,13(1):61-66

附录

表 1　支撑材料文件列表

文件列表名	
问题一	第一问的求解.py
问题二	第二问的求解.py
	result2.xlsx
问题三	第三问的求解.py
	result3.xlsx

附录 A 第一问的求解

```
import math
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import lstsq

#导入数据
data=pd.read_excel("坐标.xlsx")
X=data['x坐标 (m)']
Y=data['y坐标 (m)']

#入射光线的单位向量
def cal(a,b,c):#纬度，当地时间，距春分时间
Sunang=math.pi/12*b #太阳时角

Sundec=math.asin(math.sin(2*math.pi*c/365)*math.sin(2*math.pi/360*23.45))#太阳赤纬角

Sunalt=math.asin(math.cos(Sundec)*math.cos(a)*math.cos(Sunang)+math.sin(Sundec)*math.sin(a))
#太阳高度角

test=(math.sin(Sundec)-math.sin(Sunalt)*math.sin(a))/(math.cos(Sunalt)*math.cos(a))

if abs(test)>1:
test=test/abs(test)
Sunloc=math.acos(test)#太阳方位角
```

```
if b<0:
Sunloc=Sunloc
else:
Sunloc=-Sunloc

x=math.cos(Sunalt)*math.cos(Sunloc)
y=math.cos(Sunalt)*math.sin(Sunloc)
z=math.sin(Sunalt)

return x,y,z
a=math.radians(39.4)
lstb=[-3,-1.5,0,1.5,3]
lstc=[306,337,0,31,61,92,122,153,184,214,245,275]
lstsum=[]
lstsubsum=[]
for i in range(12):
for j in range(5):
x,y,z=cal(a,lstb[j],lstc[i])
lstsubsum.append(x)
lstsubsum.append(y)
lstsubsum.append(z)
lstsum.append(lstsubsum)
lstsubsum=[]

#计算余弦效率
mirpos=[]
lst=[]

#定义求镜器法向量的函数:
def mirmacvec(x):
mirpos=[]
sum=math.sqrt(x[0]**2+x[1]**2+76**2)
mirpos.append(-x[0]/sum)
mirpos.append(-x[1]/sum)
mirpos.append(76/sum)
return mirpos

#计算镜器法向量
for i in range(len(X)):
sum=math.sqrt(X[i]**2+Y[i]**2+76**2)
lst.append(-X[i]/sum)
lst.append(-Y[i]/sum)
lst.append(76/sum)
mirpos.append(lst)
lst=[]
#print(mirpos)
```

```
#计算镜面法向量
def Mirvec(lst1):
lst2=[]
for i in range(60):
a=lst1[0]*lstsum[i][0]+lst1[1]*lstsum[i][1]+lst1[2]*lstsum[i][2]
if a>0:
lst1[0]=-lst1[0]
lst1[1]=-lst1[1]
lst1[2]=-lst1[2]
sum=math.sqrt((lst1[0]-lstsum[i][0])**2+(lst1[1]-lstsum[i][1])**2+(lst1[2]-lstsum[i][2])**2)
x=(lst1[0]-lstsum[i][0])/sum
y=(lst1[1]-lstsum[i][1])/sum
z=(lst1[2]-lstsum[i][2])/sum
return x,y,z

#计算镜面法向量和余弦效率
def Norvec(lst1):
lst2=[]
for i in range(60):
a=lst1[0]*lstsum[i][0]+lst1[1]*lstsum[i][1]+lst1[2]*lstsum[i][2]
if a>0:
lst1[0]=-lst1[0]
lst1[1]=-lst1[1]
lst1[2]=-lst1[2]
sum=math.sqrt((lst1[0]-lstsum[i][0])**2+(lst1[1]-lstsum[i][1])**2+(lst1[2]-lstsum[i][2])**2)
x=(lst1[0]-lstsum[i][0])/sum
y=(lst1[1]-lstsum[i][1])/sum
z=(lst1[2]-lstsum[i][2])/sum
cof=abs(x*lstsum[i][0]+y*lstsum[i][1]+z*lstsum[i][2])
lst2.append(cof)
averages = []
for i in range(0, 60, 5):
chunk = lst2[i:i + 5]
sum=chunk[0]+chunk[1]+chunk[2]+chunk[3]+chunk[4]
sum=sum/5
averages.append(sum)
return averages

#计算余弦效率
def cos(lst1):
lst2=[]
for i in range(60):
a=lst1[0]*lstsum[i][0]+lst1[1]*lstsum[i][1]+lst1[2]*lstsum[i][2]
if a>0:
lst1[0]=-lst1[0]
lst1[1]=-lst1[1]
```

```
lst1[2]=-lst1[2]
sum=math.sqrt((lst1[0]-lstsum[i][0])**2+(lst1[1]-lstsum[i][1])**2+(lst1[2]-lstsum[i][2])**2)
x=(lst1[0]-lstsum[i][0])/sum
y=(lst1[1]-lstsum[i][1])/sum
z=(lst1[2]-lstsum[i][2])/sum
cof=abs(x*lstsum[i][0]+y*lstsum[i][1]+z*lstsum[i][2])
lst2.append(cof)
return lst2
lst23=[]

#计算60个时间点的平均余弦效率
for i in range(len(X)):
lst23=lst23+cos(mirpos[i])
for i in range(60):
lst23[i]=lst23[i]/len(X) #60个时间点的平均余弦效率

sumsum=[]
for i in range(len(mirpos)):
sumsum.append(Norvec(mirpos[i]))

#得到春分和夏至的结果目的是可视化
spring,summer=[],[]
for i in range(len(mirpos)):
spring.append(sumsum[i][2])
summer.append(sumsum[i][5])
positions=list(zip(X,Y))
plt.scatter([x for x, _ in positions], [y for _, y in positions], c=spring, cmap='viridis')
plt.colorbar(label='Value')
#plt.show()

#计算平均效率
avgcof=[]
for i in range(12):
sum2=0
for j in range(len(mirpos)):
sum2=sum2+sumsum[j][i]
avg=sum2/len(mirpos)
avgcof.append(avg)
#print(avgcof)


lst2=[]
positions3=[]
#计算阴影遮挡效率
for i in range(len(X)):
lst2.append(4)
positions2=list(zip(X,Y,lst2))
```

```
for x in positions2:
x=list(x)
positions3.append(x)#储存了所用定日镜的中心位置


#求镜子俯仰角和方位角
def calpos(x):
a,b,c=Mirvec(x)
theta1=math.acos(c)#俯仰角
theta2=math.atan(a/b)#方位角
return theta1,theta2

#求镜子25个格子各中心点的坐标,先旋转再平移
def midpos(x):
lst1=[]
lst4=[]
theta1,theta2=calpos(x)
for i in range(5):
a=-2.4+1.2*i
for j in range(5):
lst2=[]
b=2.4-1.2*j
lst2.append(a)
lst2.append(b)
lst2.append(0)
lst1.append(lst2)

matrix1=np.array([[math.cos(theta1),0,math.sin(theta1)],[0,1,0],[-math.sin(theta1),0,math.
cos(theta1)]])
matrix2=np.array([[math.cos(theta2),-math.sin(theta2),0],[math.sin(theta2),math.cos(theta2),0]
,[0,0,1]])
matrix3=matrix1@matrix2
# print(matrix3)
# print(matrix3.shape)
# print(np.array(lst1[0]))
# print(np.array(lst1[0]).reshape(-1,1))
# vec=np.array(lst1[0]).reshape(-1,1)
#print(matrix3.dot(vec))

#计算四个顶点的坐标
lst5=[[-3,3,0],[3,3,0],[-3,-3,0],[3,-3,0]]
lst1=lst1+lst5
for i in range(len(lst1)):
lst3=[]
vec=np.array(lst1[i]).reshape(-1,1)
w=matrix3.dot(vec)
l,m,n=w[0][0],w[1][0],w[2][0]
```

```
lst3.append(l)
lst3.append(m)
lst3.append(n)
lst4.append(lst3)
for i in range(len(lst1)):
lst4[i][0]=lst4[i][0]+x[0]
lst4[i][1]=lst4[i][1]+x[1]
lst4[i][2]=lst4[i][2]+x[2]
return lst4

def is_point_in_triangle(p, triangle):
"""
判断点 p 是否在三角形 triangle 内部
:param p: 测试点坐标 (x, y)
:param triangle: 三角形顶点坐标 [(x1, y1), (x2, y2), (x3, y3)]
:return: True 如果点 p 在三角形内部，否则 False
"""
AB = np.array(triangle[1]) - np.array(triangle[0])
BC = np.array(triangle[2]) - np.array(triangle[1])
CA = np.array(triangle[0]) - np.array(triangle[2])

AP = np.array(p) - np.array(triangle[0])
BP = np.array(p) - np.array(triangle[1])
CP = np.array(p) - np.array(triangle[2])

detABAP = np.cross(AB, AP)
detBCBP = np.cross(BC, BP)
detCAPC = np.cross(CA, CP)

if np.dot(detABAP,detBCBP)>0 and np.dot(detABAP,detCAPC)>0 and np.dot(detBCBP,detCAPC )>0:
return True
else:
return False
# print(tuple(np.array(lstsum[0])))
# print(np.array(Mirvec(positions3[1])))
# print(np.array(midpos(positions3[0])[0]))
# print(np.array(positions3[0]))

def line_plane_intersection(n, p0, d, q0):
# n 是平面的法向量 (a, b, c)
# p0 是平面上的一点 (x0, y0, z0)
# d 是直线的方向向量 (d1, d2, d3)
# q0 是直线上的一点 (u0, v0, w0)

a, b, c = n
x0, y0, z0 = p0
u0, v0, w0 = q0
```

```
d1, d2, d3 = d

# 计算平面方程的常数项
C = a * x0 + b * y0 + c * z0

# 计算直线参数方程代入平面方程后的系数
coeff_t = a * d1 + b * d2 + c * d3
const = C - (a * u0 + b * v0 + c * w0)

# 如果系数为0，直线可能与平面平行或在平面内
if coeff_t == 0:
if const == 0:
print("直线在平面内，有无穷多交点")
return None
else:
print("直线与平面平行，无交点")
return None

# 计算参数t
t = -const / coeff_t

# 计算交点坐标
intersection_point = (u0 + d1 * t, v0 + d2 * t, w0 + d3 * t)
return intersection_point
def shadow(x):
print(1)
lst6=[]
n=0
lst5=midpos(x)
lst7=lst5 #统计在某一时刻阴影下的有效点
lst13=[] #60个时间点阴影下的有效点
for i in range(60):
for j in range(25):
# 直线上的一点
point_on_line = np.array(lst5[j])
# 直线的方向向量
direction_vector = np.array(lstsum[i])
for k in range(len(X)):
if positions3[k]==x:
continue
else:
normal_vector=np.array(Mirvec(positions3[k]))
point_on_plane = np.array(positions3[k])
lst10=midpos(positions3[k])
intersection_point=line_plane_intersection(tuple(normal_vector),tuple(point_on_plane),tuple
(direction_vector),tuple(point_on_line))
triangle1=[tuple(lst10[21]),tuple(lst10[22]),tuple(lst10[23])]
```

```
triangle2=[tuple(lst10[22]),tuple(lst10[23]),tuple(lst10[24])]
if is_point_in_triangle(intersection_point, triangle1) or
    is_point_in_triangle(intersection_point, triangle2):
lst7.remove(lst5[j])
n=n+1
break
lst13.append(lst7)
lst7=lst5
lst6.append(n)
n=0
return lst6,lst13

def block(x):
lst8,lst9=shadow(x)
lst12=lst9#表示在阴影和遮挡下都有效的点
lst14=[]
m=0
for j in range(len(lst9)):
for l in range(len(lst9[j])):#遍历阴影下的有效点
print(2)
# 直线上的一点
point_on_line = np.array(lst9[j][l])
# 直线的方向向量
direction_vector = np.array(mirmacvec(x))
for k in range(len(X)):
if positions3[k]==x:
continue
else:
normal_vector=np.array(Mirvec(positions3[k]))
point_on_plane = np.array(positions3[k])
lst11=midpos(positions3[k])
intersection_point=line_plane_intersection(tuple(normal_vector),tuple(point_on_plane),tuple
(direction_vector),tuple(point_on_line))
triangle1=[tuple(lst11[21]),tuple(lst11[22]),tuple(lst11[23])]
triangle2=[tuple(lst11[22]),tuple(lst11[23]),tuple(lst11[24])]
if is_point_in_triangle(intersection_point, triangle1) or
    is_point_in_triangle(intersection_point, triangle2):
m=m+1
lst12[j].remove(lst9[j][l])
break
lst14.append(m)
m=0
return lst14,lst12,lst8
# lst15_1,lst15_2,lst15_3=block(positions3[0])
# #print(mirmacvec(positions3[0]))
# print(lst15_1+lst15_3)
sample_size=170
```

```
random_indices=np.random.choice(len(X),size=sample_size,replace=False)
sample_elements=[positions3[i] for i in random_indices]
print(sample_elements)
lst16,lst17,lst18,lst19,lst20,lst21=[],[],[],[],[],[]
for i in range(170):
lst16,lst17,lst18=block(sample_elements[i])
for i in range(len(lst16)):
lst16[i]=(lst16[i]+lst18[i])/25
for i in range(len(lst16)):
lst19.append(1-lst16[i])
lst20=lst20+lst19
lst21.append(lst17) #储存所有定日镜60个时间节点有效点
lst20=lst20/170#得到60个时间节点的平均阴影遮挡效率
print(lst20)

#求直线与圆柱面是否有交点
def does_line_intersect_cylinder(line_point, line_direction):
# 圆柱面的中心、半径和高度范围
x0, y0, z0 = (0,0,76)
R =3.5
z_min, z_max =(72,80)

# 直线上的点及其方向
a_x, a_y, a_z = line_point
b_x, b_y, b_z = line_direction

# 二次方程的系数
A = b_x**2 + b_y**2
B = 2 * (b_x*(a_x-x0) + b_y*(a_y-y0))
C = (a_x-x0)**2 + (a_y-y0)**2 - R**2

# 解二次方程
delta = B**2 - 4*A*C

# 检查是否存在实数解
if delta >= 0:
# 至少存在一个实数解
# 检查这些解对应的 z 坐标是否在圆柱的高度范围内
t1 = (-B + np.sqrt(delta)) / (2*A)
t2 = (-B - np.sqrt(delta)) / (2*A)

z1 = a_z + t1*b_z
z2 = a_z + t2*b_z

# 检查任一 z 坐标是否在高度范围内
if (z_min <= z1 <= z_max) or (z_min <= z2 <= z_max):
return True # 相交
```

```
return False # 不相交

#计算截断效率
q,p=0,0
lst22=[]
for i in range(60):
for j in range(len(X)):
for k in range(len(lst21[j][i])):
if does_line_intersect_cylinder(tuple(lst21[j][i][k]),tuple(mirpos[j])):
q=q+1
break
p=p+(len(lst21[j][i])-q)/len((lst21[j][i]))
q=0
lst22.append(p/len(X)) #60个时间节点的截断效率
print(lst22)

#计算大气透射率
def aircof(x):
dis=x[0]**2+x[1]**2+76**2
return 0.99321-0.0001176*dis+1.97e-8*dis**2
lst24,sum3=[],0
for i in range(60):
for j in range(len(X)):
sum3=sum3+aircof(positions3[j])
sum3/len(X)
lst24.append(sum3)
sum3=0

#计算光学效率
suncof=[]
for i in range(60):
suncof.append(lst20[i]*lst23[i]*lst24[i]*lst22[i]*0.92)
print(suncof)

#计算太阳高度角
def sunpos(a,b,c):
Sunang=math.pi/12*b #太阳时角

Sundec=math.asin(math.sin(2*math.pi*c/365)*math.sin(2*math.pi/360*23.45))#太阳赤纬角

Sunalt=(math.cos(Sundec)*math.cos(a)*math.cos(Sunang)+math.sin(Sundec)*math.sin(a))#太阳高度角

return Sunalt
lst25=[]#储存60个时间的太阳高度角
for i in range(12):
for j in range(5):
```

```
Sunalt=sunpos(a,lstb[j],lstc[i])
lst25.apppend(Sunalt)
#计算定日镜场的平均输出热功率
lst26=[]#储存60个时间DNI
a1=0.4237-0.00821*3**2
b1=0.5055+0.00595*3.5**2
c1=0.2711+0.01858*0.5**2
for i in range(60):
lst26.append(1.366*(a1+b1*math.exp(-c1/lst25[i])))
sum4=0
for i in range(60):
suncof[i]=suncof[i]*36
sum4=sum4+suncof[i]
lst27=[]#60个时间点的光学效率
for i in range(60):
lst27.append(lst26[i]*sum4)
```

附录 B 第二问的求解

```
import math
from random import randint
import numpy as np
import matplotlib.pyplot as plt
from random import random

class CAMPO:
def __init__(self, grid_size=(50, 50), initial_density=0.1, dispersal_prob=0.1,
    extinction_prob=0.05, iterations=100):
self.grid_size = grid_size
self.initial_density = initial_density
self.dispersal_prob = dispersal_prob
self.extinction_prob = extinction_prob
self.iterations = iterations
self.grid = np.zeros(grid_size)
self.initialize_grid()

def initialize_grid(self):
num_patches = np.prod(self.grid_size)
num_initial = int(self.initial_density * num_patches)
positions = [(randint(0, self.grid_size[0]-1), randint(0, self.grid_size[1]-1)) for _ in
    range(num_initial)]
for pos in positions:
self.grid[pos] = 1

def update_grid(self):
```

```
new_grid = np.zeros_like(self.grid)
for i in range(self.grid_size[0]):
for j in range(self.grid_size[1]):
if self.grid[i, j] == 1:
for di in [-1, 0, 1]:
for dj in [-1, 0, 1]:
if di == 0 and dj == 0:
continue
ni, nj = i + di, j + dj
if 0 <= ni < self.grid_size[0] and 0 <= nj < self.grid_size[1]:
if random() < self.dispersal_prob:
new_grid[ni, nj] = 1
if random() < self.extinction_prob:
new_grid[i, j] = 0
else:
for di in [-1, 0, 1]:
for dj in [-1, 0, 1]:
if di == 0 and dj == 0:
continue
ni, nj = i + di, j + dj
if 0 <= ni < self.grid_size[0] and 0 <= nj < self.grid_size[1]:
if self.grid[ni, nj] == 1 and random() < self.dispersal_prob:
new_grid[i, j] = 1

def run_simulation(self):
for t in range(self.iterations):
self.update_grid()

def plot_simulation(self):
plt.figure(figsize=(8, 6))
plt.imshow(self.grid, cmap='Greens', interpolation='nearest')
plt.title('CAMPO Simulation')
plt.colorbar(label='Species Presence')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

campo = CAMPO(grid_size=(50, 50), initial_density=0.1, dispersal_prob=0.1,
    extinction_prob=0.05, iterations=100)
campo.run_simulation()
campo.plot_simulation()

def func(x, y):
res = 4*x**2 - 2.1*x**4 + x**6/3 + x*y - 4*y**2 + 4*y**4
return res

class SA:
```

```
def __init__(self, func, iter=100, T0=100, Tf=0.01, alpha=0.99):
self.func = func
self.iter = iter
self.alpha = alpha
self.T0 = T0
self.Tf = Tf
self.T = T0
self.x = [random() * 11 - 5 for i in range(iter)]
self.y = [random() * 11 - 5 for i in range(iter)]
self.history = {'f': [], 'T': []}

def generate_new(self, x, y):
while True:
x_new = x + self.T * (random() - random())
y_new = y + self.T * (random() - random())
if (-5 <= x_new <= 5) and (-5 <= y_new <= 5):
break
return x_new, y_new

def Metropolis(self, f, f_new):
if f_new <= f:
return 1
else:
p = math.exp((f - f_new) / self.T)
if random() < p:
return 1
else:
return 0

def best(self):
f_list = [self.func(self.x[i], self.y[i]) for i in range(self.iter)]
f_best = min(f_list)
idx = f_list.index(f_best)
return f_best, idx

def run(self):
while self.T > self.Tf:
for i in range(self.iter):
f = self.func(self.x[i], self.y[i])
x_new, y_new = self.generate_new(self.x[i], self.y[i])
f_new = self.func(x_new, y_new)
if self.Metropolis(f, f_new):
self.x[i] = x_new
self.y[i] = y_new
ft, _ = self.best()
self.history['f'].append(ft)
self.history['T'].append(self.T)
```

```
self.T = self.T * self.alpha

f_best, idx = self.best()
print(f"最优解: F={f_best}, x={self.x[idx]}, y={self.y[idx]}")

plt.plot(self.history['T'], self.history['f'])
plt.xlabel('Temperature')
plt.ylabel('Objective Function Value')
plt.gca().invert_xaxis()
plt.show()

sa = SA(func)
sa.run()
```

附录 C 第三问的求解

```
import math
import numpy as np
import matplotlib.pyplot as plt
from random import random, uniform, randint

class CAMPO:
def __init__(self, grid_size=(50, 50), initial_density=0.1, dispersal_prob=0.1,
    extinction_prob=0.05, iterations=100):
self.grid_size = grid_size
self.initial_density = initial_density
self.dispersal_prob = dispersal_prob
self.extinction_prob = extinction_prob
self.iterations = iterations
self.grid = np.zeros(grid_size)
self.initialize_grid()

def initialize_grid(self):
num_patches = np.prod(self.grid_size)
num_initial = int(self.initial_density * num_patches)
positions = [(randint(0, self.grid_size[0]-1), randint(0, self.grid_size[1]-1)) for _ in
    range(num_initial)]
for pos in positions:
self.grid[pos] = 1

def run_simulation(self, l, w, h):
for t in range(self.iterations):
self.update_grid(l, w, h)

def plot_simulation(self):
```

```
plt.figure(figsize=(8, 6))
plt.imshow(self.grid, cmap='Greens', interpolation='nearest')
plt.title('CAMPO Simulation')
plt.colorbar(label='Species Presence')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

class SolarConcentratorSA:
def __init__(self, campo_simulator, iter=100, T0=100, Tf=0.01, alpha=0.99):
self.campo = campo_simulator
self.iter = iter
self.alpha = alpha
self.T0 = T0
self.Tf = Tf
self.T = T0
self.l = uniform(2, 8)
self.w = uniform(2, 6)
self.h = uniform(5, 10)
self.best_params = (self.l, self.w, self.h)
self.best_objective = self.objective_function(self.l, self.w, self.h)
self.history = {'objective': [], 'T': []}

def objective_function(self, l, w, h):
concentration_efficiency = l * w / h
return concentration_efficiency

def generate_new(self, l, w, h):
l_new = max(2, min(8, l + uniform(-0.5, 0.5)))
w_new = max(2, min(6, w + uniform(-0.5, 0.5)))
h_new = max(5, min(10, h + uniform(-1, 1)))
return l_new, w_new, h_new

def Metropolis(self, f_current, f_new):
if f_new > f_current:
return 1
else:
p = math.exp((f_new - f_current) / self.T)
if random() < p:
return 1
else:
return 0

def run(self):
while self.T > self.Tf:
for _ in range(self.iter):
l_new, w_new, h_new = self.generate_new(self.l, self.w, self.h)
```

```
f_current = self.objective_function(self.l, self.w, self.h)
f_new = self.objective_function(l_new, w_new, h_new)
if self.Metropolis(f_current, f_new):
self.l, self.w, self.h = l_new, w_new, h_new
self.best_params = (self.l, self.w, self.h)
self.best_objective = f_new
self.history['objective'].append(self.best_objective)
self.history['T'].append(self.T)
self.T *= self.alpha

self.campo.run_simulation(self.l, self.w, self.h)
self.campo.plot_simulation()

print(f"Best parameters: l={self.best_params[0]}, w={self.best_params[1]},
    h={self.best_params[2]}")
print(f"Best objective value: {self.best_objective}")

plt.plot(self.history['T'], self.history['objective'])
plt.title('Simulated Annealing Optimization')
plt.xlabel('Temperature')
plt.ylabel('Objective Value')
plt.gca().invert_xaxis() # Invert x-axis to show decreasing temperature
plt.show()

campo = CAMPO(grid_size=(50, 50), initial_density=0.1, dispersal_prob=0.1,
    extinction_prob=0.05, iterations=100)
sa = SolarConcentratorSA(campo_simulator=campo)
sa.run()
```