基于 X 模型的定日镜场优化设计问题的研究

摘要

本文针对定日镜场优化设计问题，通过建立镜面离散化模型利用梯度下降优化算法，实现对给定参数和分布的定日镜场的相关数据求解以及给定部分参数的优化设计。

针对问题一，建立镜面离散化模型，根据已给出的有关定日镜场的参数计算出太阳角等位置参数，进一步计算出法向直接辐射辐照度 (DNI)、定日镜的阴影遮挡效率、余弦效率、大气透射率及集热器截断效率，得到定日镜的光学效率。再根据法向直接辐射辐照度、定日镜面积及光学效率计算定日镜场的输出热功率。最后将定日镜场的输出热功率除以定日镜总面积即可求得单位面积镜面平均输出热功率。

针对问题二，考虑到决策变量、目标函数和约束条件都很明确，我们考虑径向交错的辐射网络状圆形布局，由间距得出其他参数并尽可能实现交错，得到定日镜布局。再使用梯度下降法优化模型，将问题一的计算过程简化作为目标函数，通过改变初始参数不断迭代最终求解出在给定初始值时的最优解，并在此基础上控制变量进行灵敏度分析，最终确定最优解。

针对问题三，考虑到问题的复杂性，采用与问题二不同的思路，建立经验模型，以减小阴影遮挡的影响，最大化利用太阳光能量。根据问题二得到的经验，吸收塔在沿 y 轴负方向偏移 250 米时单位位镜面面积年平均输出热功率最高，据此，以吸收塔中心所在位置为抛物面中心建立抛物面分布模型，求解出在该模型下的相关数据，发现仅在阴影遮挡效率上有所提升。

最后，对本文所建的模型进行了误差分析，评价了模型的优缺点，并提出一些改进方案。

关键字：梯度下降法　灵敏度分析　径向交错网格状圆形布局　抛物面模型

目录

一、 问题的重述

1.1 问题的背景

定日镜底座由纵向转轴和水平转轴组成，轴线与地面垂直的纵向转轴控制反射镜的方位角，轴线与地面平行的水平转轴控制反射镜的俯仰角。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，加热集热器中的导热介质，并将太阳能以热能形式储存起来，再经过热交换实现由热能向电能的转化。现需要根据已知地理位置信息，设计定日镜场的位置分布，高度和尺寸分布，从而优化使单位镜面面积年平均输出热功率更大。

1.2 相关数据

表 1　定日镜场参数

参数	参数值
经纬度	(39.4°N,98.5°E)
海拔	3000m
半径	350m
吸收塔高度	80m
圆柱形集热器	高 8m，直径 7m
镜面边长	2∼8m(镜面宽度 ⩾ 镜面高度)
安装高度	2∼6m(安装高度 ⩾ 镜面高度/2)
相邻定日镜底座中心之间的距离	⩾ 镜面宽度 +5m

注: 为简化计算，所有 “年均” 指标的计算时点均取当地时间每月 21 日 9:00、10:30、12:00、13:30、15:00。

1.3 需要解决的问题

问题一

已知定日镜尺寸 (6m×6m) 和高度 (4m) 以及位置分布数据，直接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。

问题二

在定日镜场的额定年平均输出热功率达到 60MW 且所有定日镜尺寸及安装高度相同的前提下，设计定日镜场的吸收塔的位置坐标、定日镜尺寸、安装高度、定日镜数目和定日镜位置，使单位镜面面积年平均输出热功率最大。

问题三

在问题二的基础上，更改条件为定日镜尺寸和安装高度可以不同，重新设计定日镜场的各个参数，使定日镜场在达到额定功率至少为 60MW 的条件下，单位镜面面积年平均输出热功率最大。

二、 模型的假设

1. 定日镜场为圆形，假定镜场中的定日镜中心与镜面支撑点重合；
2. 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始终指向集热器中心；
3. 对于太阳光线的形状，在计算阴影和遮挡效率时，由于考虑的距离范围很小，可视为平行光；而计算截断效率时应将太阳光视为具有一定锥形角的一束锥形光线，太阳入射光线经定日镜任意一点的反射光线也是一束锥形光线；
4. 假设太阳照射不考虑云或其他物质的影响，各种已归结在大气透射率中；
5. 假设镜面整洁，镜面反射率不会改变。

三、 符号的说明

本模型所用的各主要变量符号及意义见表 2

表 2　主要变量符号及意义

符号	意义
ϕ	纬度 (北纬为正)(单位:°)
ω	太阳时角 (单位:°)
ST	当地时间 (单位:h)
D	以春分 (3 月 21 日) 作为第 0 天起算的天数
δ	太阳赤纬角 (单位:°)
α_s	太阳高度角 (单位:°)
γ_s	太阳方位角 (单位:°)
DNI	法向直接辐射辐照度 (单位:kW/m^2)
G_0	太阳常数 (取 1.366kW/m^2)
H	海拔高度 (单位:km)
γ	太阳光日射角 (单位:°)
$\hat{\mathbf{e}}_s$	指向太阳的入射光矢量

$\hat{\mathbf{n}}_h$	定日镜中心处的法向量 (有 $\hat{\mathbf{n}}_{h,x}$、$\hat{\mathbf{n}}_{h,y}$、$\hat{\mathbf{n}}_{h,z}$ 三个地平坐标系下的分量)
α_h	定日镜姿态高度角 (单位:°)
γ_h	定日镜姿态方位角 (单位:°)
C_{L0}	定日镜 A 自身坐标系
C_{L1}	定日镜 B 自身坐标系
C_G	全局 (地平) 坐标系
α_A	定日镜 A 的高度角 (单位:°)
γ_A	定日镜 A 的方位角 (单位:°)
α_B	定日镜 B 的高度角 (单位:°)
γ_B	定日镜 B 的方位角 (单位:°)
η	定日镜的光学效率
η_{sb}	阴影遮挡效率
η_{cos}	余弦效率
η_{at}	大气投射率
η_{trunc}	集热器截断效率
η_{ref}	镜面反射率
HR	镜面中心到集热器中心的距离 ($HR \leqslant 1000$)(单位:m)
E_{field}	定日镜场的输出热功率
N	定日镜总数 (单位: 面)
A	定日镜采光面积 (单位:m^2)
(x, y)	位置坐标 (单位:m)
m	定日镜高度 (单位:m)
n	定日镜宽度 (单位:m)
h	定日镜安装高度 (单位:m)
d	相邻定日镜底座中心之间的距离 (单位:m)

四、 模型建立与求解

4.1 问题一的分析与求解

4.1.1 模型的分析

有关定日镜场的各种参数已给出，首先我们在程序中定义定日镜的尺寸、中心位置、安装高度、吸收塔的高度、定日镜场的中心坐标、太阳常数等。然后根据地理位置和日期时间，通过公式计算太阳的赤纬角、高度角和方位角等位置参数。再利用所得到

的太阳高度角等数据，计算法向直接辐射辐照度 (DNI)。再利用给定的公式，计算定日镜的阴影遮挡效率 (考虑定日镜间的相互遮挡，吸收塔影子对定日镜的遮挡)、余弦效率 (太阳光线入射角与定日镜法线向量的夹角余弦值)、大气透射率 (与镜面中心到集热器中心的距离有关)、集热器截断效率 (集热器截获的能量占镜场汇聚能量的百分比)、镜面反射率 (取常数 0.92)，相乘得到定日镜的光学效率。根据法向直接辐射辐照度、定日镜面积及定日镜的光学效率计算定日镜场的输出热功率。最后将定日镜场的输出热功率除以定日镜总面积即可求得单位面积镜面平均输出热功率。

4.1.2 模型的建立与求解

一. 计算太阳位置参数

定日镜场中心位于北纬 39.4°，即 $\varphi = 39.4$ (北纬为正);

太阳时角

$$\omega = \frac{\pi}{12}(ST - 12) \tag{4-1-1}$$

其中，ST 为当地时间，为简化计算，“年均”指标取当地时间每月 21 日 9:00、10：30、12：00、13：30、15：00;

太阳赤纬角

$$sin\delta = sin\frac{2\pi D}{365}sin(\frac{2\pi}{360}23.45) \tag{4-1-2}$$

其中，D 为以春分 (3 月 21 日) 作为第 0 天起算的天数;

由此求得太阳高度角 α_s

$$sin\alpha_s = cos\delta cos\varphi cos\omega + sin\delta sin\varphi \tag{4-1-3}$$

太阳方位角 γ_s

$$cos\gamma_s = \frac{sin\delta - sin\alpha_s sin\varphi}{cos\alpha_s cos\varphi} \tag{4-1-4}$$

二. 计算法向直接辐射辐照度

地球上垂直于太阳光线的平面单位面积单位时间内接收到的太阳辐射能量为法向直接辐射辐照度 DNI(单位：kW/m^2)，近似计算:

$$\begin{cases} DNI = G_0[a + b\,exp(-\dfrac{c}{sin\alpha_s})] \\ a = 0.4237 - 0.00821(6 - H)^2 \\ b = 0.5055 + 0.00595(6.5 - H)^2 \\ c = 0.2711 + 0.01858(2.5 - H)^2 \end{cases} \tag{4-1-5}$$

其中，太阳常数 G_0 取 1.366kW/m^2,H 为海拔高度 (单位：km)。

三. 计算定日镜的光学效率

余弦效率 定日镜的余弦损失指由于太阳光入射方向与镜面采光口法线方向不平行引起的接收能量损失 [1]。余弦效率是定日镜上太阳入射角的余弦值。由定义得余弦效率：

$$\eta_{cos} = cos\gamma = \hat{\mathbf{e}}_s \cdot \hat{\mathbf{n}}_h \tag{4-1-6}$$

如图所示：γ 为太阳光入射角，$\hat{\mathbf{e}}_s$ 为指向太阳的入射光矢量，$\hat{\mathbf{n}}_h$ 为定日镜中心处的法向量。

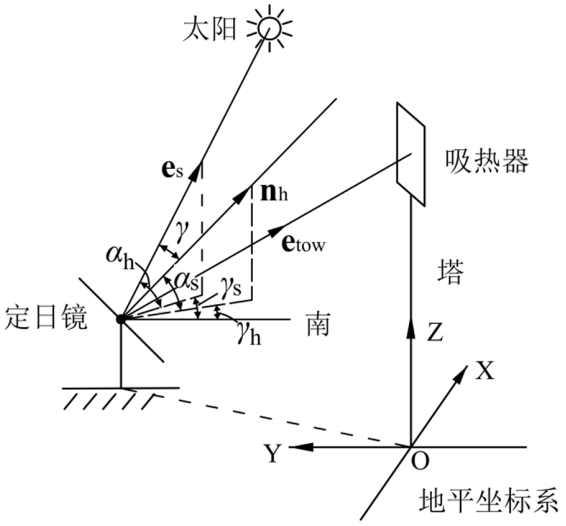

图 1 光径示意图 [3]

在以正东方向为 x 轴，正北方向为 y 轴，z 轴指向天顶的地平坐标系下，入射光矢量 $\hat{\mathbf{e}}_s$ 为：

$$\hat{\mathbf{e}}_s = (sin\gamma_s cos\alpha_s, cos\gamma_s cos\alpha_s, sin\alpha_s)^T \tag{4-1-7}$$

由反射定律得定日镜中心点的法向量 $\hat{\mathbf{n}}_h$：

$$\hat{\mathbf{n}}_h = \frac{\hat{\mathbf{e}}_s + \hat{\mathbf{e}}_{tow}}{\|\hat{\mathbf{e}}_s + \hat{\mathbf{e}}_{tow}\|} \tag{4-1-8}$$

由几何关系得：

$$\begin{cases} sin\alpha_h = \hat{\mathbf{n}}_{h,z} \\ cos\gamma_h = \dfrac{\hat{\mathbf{n}}_{h,y}}{cos\alpha_h} \end{cases} \tag{4-1-9}$$

其中，γ_h 为定日镜姿态方位角，α_h 为定日镜姿态高度角，$\hat{\mathbf{n}}_{h,x}$、$\hat{\mathbf{n}}_{h,y}$、$\hat{\mathbf{n}}_{h,z}$ 分别为矢量 $\hat{\mathbf{n}}_h$ 在地平坐标系下的三个坐标分量。

阴影和遮挡效率 阴影遮挡效率 $\eta_{sb}=1-$ 阴影遮挡损失；

采用数值计算方法，对定日镜镜面进行离散化以生成镜面离散点，如图所示 [3]：

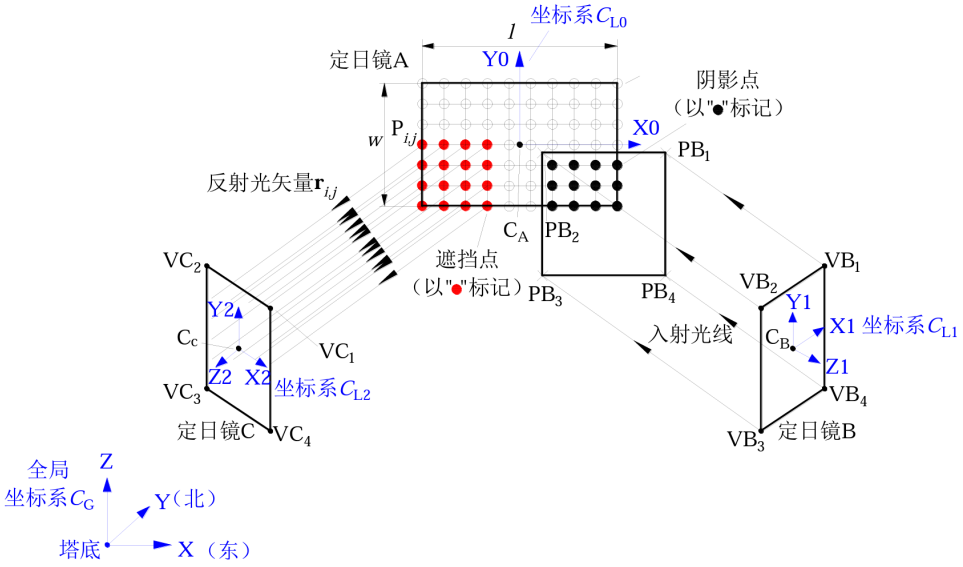

图 2 阴影和遮挡效率的数值计算

每个离散点代表以该元为中心的一块微元面，该点的阴影或遮挡则表示微元的阴影或遮挡，这样通过统计镜面中被阴影和遮挡的点的数量来实现阴影和遮挡效率的计算。

（1）建立离散化镜面模型

设矩形定日镜尺寸为 l(长)×w(宽)，将镜面划分成 I(行数)×J(列数) 个离散点。在定日镜 A 自身坐标系 C_{L0}（定日镜中心 C_A 为坐标系原点，中心点法线为 Z_0 轴且指向上方，X_0 轴平行于俯仰轴）下，定日镜 A 中点 $P_{i,j}$（第 i 行第 j 列）的坐标 $P_{i,j}^{C_{L0}}$ 表示为：

$$\begin{pmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{pmatrix} = \begin{pmatrix} -l/2+(j-1/2)\times l/J \\ w/2-(i-1/2)\times w/I \\ 0 \end{pmatrix} \tag{4-1-10}$$

定日镜镜面面型为平面型，因此镜面中各离散点处的单位法向量 $n_{i,j}$，其坐标 $n_{i,j}^{C_{L0}}$ 均为：

$$n_{i,j}^{C_{L0}}=(0,0,1) \tag{4-1-11}$$

（2）阴影分析

邻近定日镜若有阴影，则在相邻镜面上的投影为四边形，因此只需求被遮挡部分四个顶点的投影位置就可以确定阴影的位置，形状和面积等信息。邻近定日镜 B 上第 n 个

顶点 VB_n 在以自身为参考的坐标系 CL_1 下的坐标可表示为：

$$VB_n^{C_{L1}} = \begin{cases} VB_1^{C_{L1}} : (l/2, w/2, 0)^T & n = 1 \\ VB_2^{C_{L1}} : (-l/2, w/2, 0)^T & n = 2 \\ VB_3^{C_{L1}} : (-l/2, -w/2, 0)^T & n = 3 \\ VB_4^{C_{L1}} : (l/2, -w/2, 0)^T & n = 4 \end{cases} \tag{4-1-12}$$

设 α_B、γ_B 分别为定日镜 B 的高度角和方位角，将坐标系 C_{L1} 沿 X_1 轴顺时针旋转 $\pi/2 - \alpha_B$，再沿 Z_1 轴逆时针旋转 γ_B，再将定日镜 B 中心平移至塔底，得到全局坐标系 C_G，顶点在该坐标系下的坐标 $VB_n^{C_G}$ 为：

$$VB_n^{C_G} = \begin{pmatrix} cos\gamma_B & sin\gamma_B sin\alpha_B & -sin\gamma_B cos\alpha_B \\ -sin\gamma_B & cos\gamma_B sin\alpha_B & -cos\gamma_B cos\alpha_B \\ 0 & cos\alpha_B & sin\alpha_B \end{pmatrix} VB_n^{C_{L1}} + C_B^{C_G} \tag{4-1-13}$$

其中，$C_B^{C_G}$ 为定日镜 B 中心点 C_B 的全局坐标。

由经过定日镜 B 顶点的入射光线方程和定日镜 A 的平面方程联立得 VB_n 投影点 PB_n 的全局坐标 $PB_n^{C_G}$：

$$PB_n^{C_{L0}} = \begin{pmatrix} cos\gamma_A & -sin\gamma_A & 0 \\ sin\alpha_A sin\gamma_A & sin\alpha_A cos\gamma_A & cos\alpha_A \\ -cos\alpha_A sin\gamma_A & -cos\alpha_A cos\gamma_A & sin\alpha_A \end{pmatrix} (PB_n^{C_G} - C_A^{C_G}) \tag{4-1-14}$$

其中，α_A，γ_A 分别为定日镜 A 的高度角和方位角。

（3）遮挡分析

可以发现，被遮挡的光线是反射光线，假如将该反射光线看做“入射光线”，则求遮挡的过程即转换为求阴影的过程。设遮挡 A 镜反射光线的定日镜为 C 镜，根据入射光线向量（太阳位置求出）求出反射光线向量，则将定日镜 C 的参数和反射光线向量带入求阴影过程，得到在镜 A 坐标系 C_{L0} 上的四个顶点 $PC_n^{C_L0}$。

（4）综合计算

搜索镜 A 临近定日镜，分别计算阴影和遮挡，将所得 $PB_n^{C_{L0}}$、$PC_n^{C_L0}$... 存入数组 L，遍历镜 A 坐标系 C_{L0} 上所有离散点，统计不在 L 中顶点所围成矩形中的点的数目，记为 N_{shadow}，则定日镜 A 的阴影效率 η_{shadow} 为：

$$\eta_{shadow} = 1 - \frac{N_{shadow}}{I \times J} \tag{4-1-15}$$

大气透射率 大气投射率 $\eta_{at}=0.99321-0.0001176d_{HR}+1.97\times10^{-8}\times d_{HR}^2(d_{HR}\leqslant 1000)$，其中，$d_{HR}$ 表示镜面中心到集热器中心的距离 (单位:m)；

集热器截断效率 截断效率为吸热器截获的能量占镜场汇聚能量的百分比。其中，镜场汇聚的能量是整个镜场能够反射出的太阳光能量，要除去已经被阴影或挡光、衰减等损失的太阳光能量 [7]。

集热器截断效率 $\eta_{trunc}=\dfrac{\text{集热器接收能量}}{\text{镜面全反射能量-阴影遮挡损失能量}}$；

建立一个光锥坐标系 $O_S-X_SY_SZ_S$(Z_S 沿主光线方向，并朝向太阳圆盘中心;X_S 轴始终与地面平行;Y_S 轴垂直于 X_S 和 Z_S 轴)。设光锥中任一光线与来自太阳中心的主光线的夹角为 σ，与 X_S 轴的夹角为 τ，则该光线表达式为:

$$\vec{V_s}=(sin\sigma cos\tau,sin\sigma sin\tau,os\sigma)=(a,b,c) \tag{4-1-16}$$

一束光锥由无数根光线组成，计算截断效率或者计算吸热器上的能流密度，即累积计算许多光线在经过定日镜的反射后到达吸热器上时的落点，根据落入。以一束光锥中某一根光线为例。整个计算过程如下:

光锥坐标系到地面坐标系的矩阵转换关系式为:

$$T=\begin{pmatrix} sin\gamma & -sin\alpha cos\gamma & cos\alpha cos\gamma \\ -cos\gamma & -sin\alpha sin\gamma & cos\alpha sin\gamma \\ 0 & cos\alpha & sin\alpha \end{pmatrix} \tag{4-1-17}$$

将光锥坐标系中的 $\vec{V_s}$ 向量转化到地面坐标系中，得:

$$\vec{V_{sl}}=T\cdot\vec{V_s}=(a_1,b_1,c_1) \tag{4-1-18}$$

设定日镜的法线单位向量 $\vec{V_N}=(u_0,v_0,w_0)$，光线通过定日镜反射后的反射光线向量为 $\vec{V_R}$，光线向量 $\vec{V_{Sl}}$ 与镜面法向向量 $\vec{V_N}$ 的夹角 θ 的余弦值为:

$$cos\theta=\vec{V_{Sl}}\cdot\vec{V_N} \tag{4-1-19}$$

反射光线的方程表示为:

$$\frac{x-x_1}{m}=\frac{y-y_1}{n}=\frac{z}{l} \tag{4-1-20}$$

圆柱式集热器的方程表示为：

$$\begin{cases} x^2+y^2=R^2 \\ z\in[-\frac{h}{2},\frac{h}{2}] \end{cases} \tag{4-1-21}$$

由于 $x^2+y^2=R^2$，转换到极坐标系中得 $x=Rcos\theta, y=Rsin\theta$，与方程 $\frac{x-x_1}{m}=\frac{y-y_1}{n}$ 联立得到一元二次方程：

$$(n^2+m^2)cos^2\theta-\frac{2n(nx_1-my_1)}{R}cos\theta+(\frac{nx_1-my_1}{R})^2-m^2=0 \tag{4-1-22}$$

并由 $z\in[-\frac{h}{2},\frac{h}{2}]$ 与 $\frac{x-x_1}{m}=\frac{y-y_1}{n}=\frac{z}{l}$ 联立得到两个约束条件：

$$\begin{cases} \frac{l(Rcos\theta-x_1)}{m}\in[-\frac{h}{2},\frac{h}{2}] \\ \\ \frac{l(Rsin\theta-y_1)}{n}\in[-\frac{h}{2},\frac{h}{2}] \end{cases} \tag{4-1-23}$$

再由公式 $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ 得到方程的两个解 x_1,x_2，若满足约束条件，即为最后 $cos\theta$ 的结果，由此即可得到光线到达吸热器上时的落点坐标 (x,y)。

通过随机均匀模拟光锥中的光线，计算统计落点位于集热器柱面上的数目占比，作为截断效率的值。

镜面反射率 镜面反射率 η_{ref} 取为常数 0.92。

因此，定日镜的光学效率 η 为：

$$\eta=\eta_{sb}\eta_{cos}\eta_{at}\eta_{trunc}\eta_{ref} \tag{4-1-24}$$

四. 计算定日镜场的输出热功率

定日镜场的输出热功率 E_{field} 为：

$$E_{field}=DNI\cdot\sum_{i}^{N}A_i\eta_i \tag{4-1-25}$$

其中，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

定日镜的光学效率。

五. 计算单位面积镜面平均输出热功率

将定日镜场的输出热功率除以定日镜总面积即可求得单位面积镜面平均输出热功率:

$$E_{aver} = \frac{E_{field}}{\sum_i^N A_i} \tag{4-1-26}$$

4.1.3 模型的最终输出结果

表 3 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	0.583358	0.719235	0.951581	0.959912	0.516249
2 月 21 日	0.606576	0.739994	0.960297	0.961311	0.583307
3 月 21 日	0.629799	0.760922	0.970191	0.960765	0.642450
4 月 21 日	0.657515	0.779306	0.991656	0.958186	0.698711
5 月 21 日	0.668927	0.789378	0.998143	0.956123	0.724566
6 月 21 日	0.671936	0.792448	0.999060	0.955825	0.732043
7 月 21 日	0.668873	0.789272	0.998101	0.956214	0.724404
8 月 21 日	0.656472	0.778596	0.990891	0.958276	0.696620
9 月 21 日	0.628420	0.759863	0.969304	0.960877	0.639365
10 月 21 日	0.603819	0.737358	0.959416	0.961244	0.575488
11 月 21 日	0.580891	0.717472	0.950448	0.959343	0.509677
12 月 21 日	0.570323	0.710260	0.944890	0.957050	0.481106

表 4 问题一：年平均光学效率及输出功率表

年平均光学效率	年平均余弦效率	年平均阴影遮挡效率	年平均截断效率	年平均输出热功率 (MW)	单位面积镜面平均输出热功率 (kW/m^2)
0.62724241	0.756175395	0.973664863	0.958760451	39.38806991	0.626998884

4.2 问题二的分析与求解

4.2.1 模型的分析

问题 2 为数学规划问题。

决策变量:

• 每个定日镜 i 的位置坐标（x_i, y_i）
• 吸收塔位置坐标（x', y'）
• 定日镜尺寸 (高度 m 和宽度 n)
• 定日镜安装高度 h
• 定日镜数目 N
• 相邻定日镜底座中心之间的距离 d

目标函数：
• 单位面积镜面年平均输出热功率尽量大

约束条件：
• 年平均输出热功率达到 60MW
• 吸收塔周围 100 m 范围内不安装定日镜，且所有定日镜的位置坐标必须在半径 350m 的圆形区域内建设
• 定日镜高度在 2m 到 8 米之间
• 定日镜宽度在 2m 到 8 米之间且不小于定日镜高度
• 定日镜安装高度在 2m 到 6 米之间且不小于定日镜高度的一半 (确保镜面在绕水平转轴旋转时不会触及地面)
• 相邻定日镜底座中心之间的距离比镜面宽度多 5m 以上

因此，可以使用数学优化方法来求解这个问题，以找到最优的定日镜场设计参数。

4.2.2 模型的建立与求解

建立模型：

$$max \sum_{i=1}^{N} DNI \times \eta_i$$

$$s.t. \begin{cases} \sum_{i=1}^{N} DNI \times m \times n \times \eta_i \geqslant 60MW \\ |x_i|, |y_i| \leqslant 350,\ i \in [1, N] \\ (x_i - x')^2 + (y_i - y')^2 \geqslant 100^2,\ i \in [1, N] \\ m \in [2, 8] \\ n \in [m, 8] \\ h \in [max(2, \frac{m}{2}), 6] \\ d \geqslant n + 5 \end{cases} \tag{4-2-1}$$

（1）设计定日镜分布

如图所示，国内外的镜场布局多采用径向交错的辐射网格状圆形布局方式 [8]：

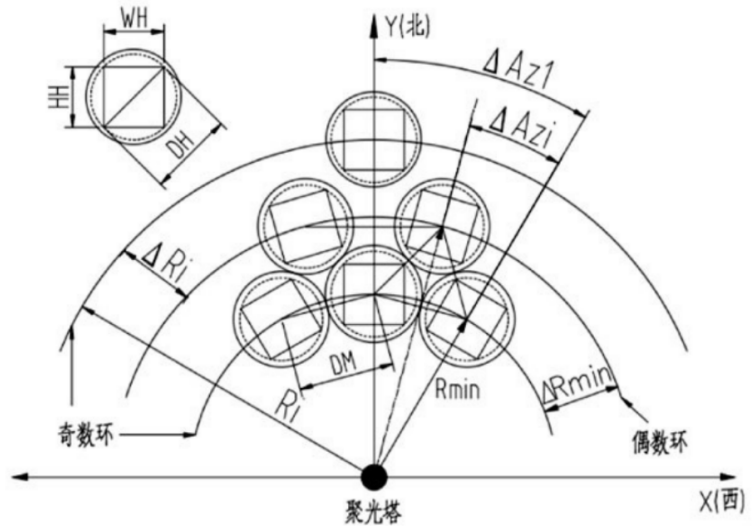

图 3　径向交错的辐射网格状圆形布局

受此启发，我们选择设计让定日镜径向交错以减小阴影遮挡，增大受光面积。

编写输出二维圆形点阵的算法：该算法在实现图 3 交错布局的同时还能保证点的密集布局。

首先根据输入的最小间距 d(满足与相邻定日镜底座中心之间的距离比镜面宽度多 5m)，利用该算法计算出圈数、每圈的直径、每圈的定日镜数量和每圈的定日镜间距，再计算某一环中的镜子夹角，尽可能使下一环的镜子位于上一环镜子中心连线的垂直平分线上，即可确定下一环镜子的布局位置。

（2）简化设计目标函数

由于问题一的模拟计算函数采用镜面离散化模型和光线追踪算法，需要逐点逐光线计算，复杂性过大，无法直接作为问题二优化模型的目标函数。因此我们将该函数利用随机方法进行简化，依据是梯度下降算法的求解过程只需要利用函数的梯度变化；同时将采集数据的月份从 12 个月减为取 3，6，9，12 月，依据是它们代表了一年春分，夏至，秋分，冬至四个太阳位置的关键节点。

（3）运行优化模型

利用梯度下降法模型，固定安装高度初始参数为 4，镜高/宽初始参数为 6，选择不同吸收塔 Y 轴偏移初始参数运行梯度下降优化方法 5 次，得到如下数据：

表 5　问题二梯度下降优化结果

初始参数	安装高度	镜高/宽	吸收塔偏移	镜面数	年平均功率	单位输出	效率
-50	3.966239	5.2225592	-47.7415397	3398	60731.13001	0.655271	0.643163
-100	3.933475	5.6525332	-85.3709916	3131	65181.46418	0.651561	0.639309]
-150	4.014058	6.0074831	-149.432461	2943	69233.19663	0.651837	0.639161
-200	3.796355	5.5927100	-200.513084	3168	68413.06684	0.690413	0.677010
-250	4.598296	5.2825124	-251.476478	3354	66070.74637	0.705936	0.692396

由此得：当吸收塔位置偏移约为-250 时，可以在年平均输出热功率达到 60MW 的前提下得到最大单位面积镜面年平均输出热功率。

固定吸收塔位置偏移和安装高度，以镜高/宽为变量进行灵敏度分析：

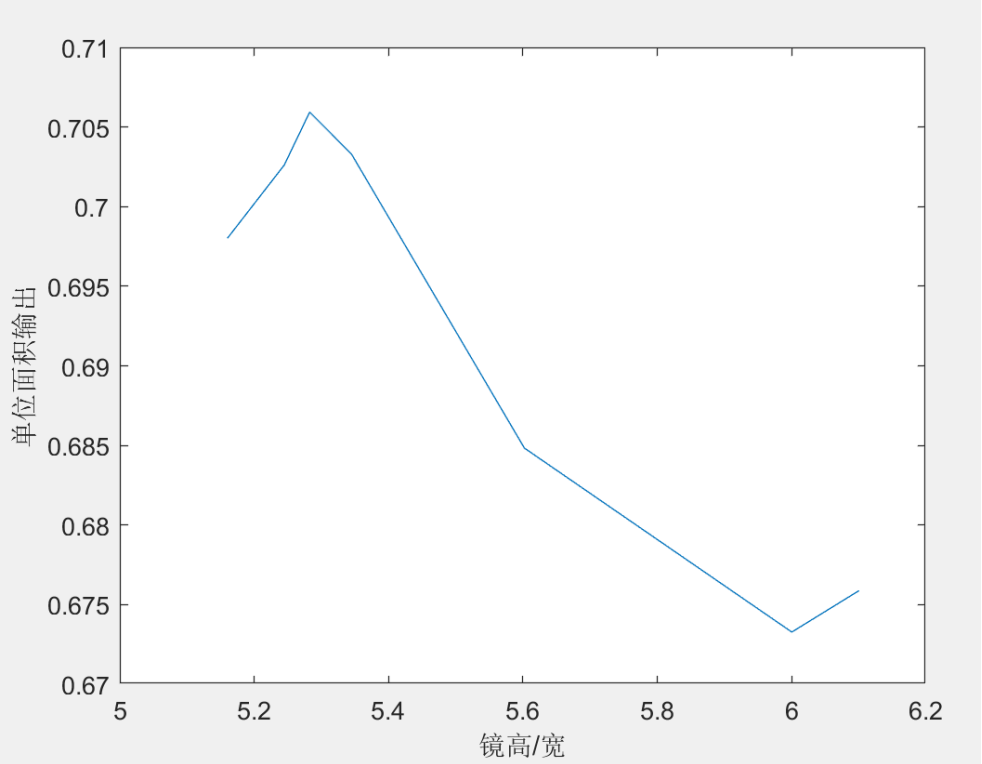

图 4　镜高/宽灵敏度分析

固定吸收塔位置偏移和镜高/宽，以定日镜安装高度为变量进行灵敏度分析：

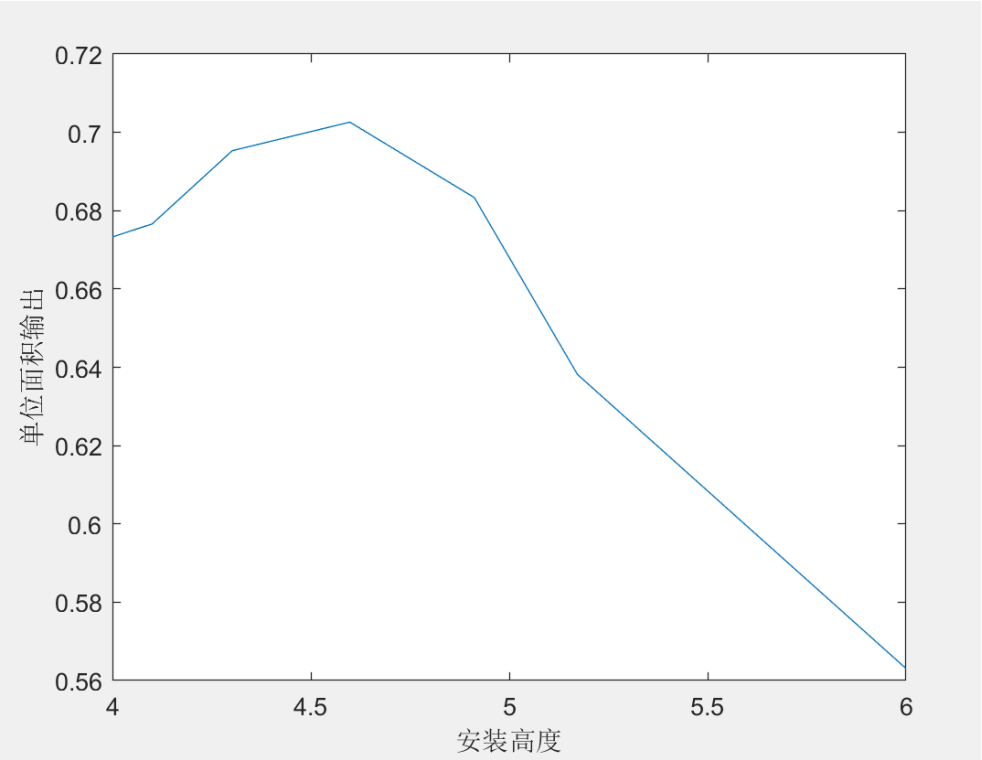

图 5　安装高度灵敏度分析

最后确定各最优参数进行计算得到输出结果。将得到的定日镜位置与高度数据作图，得到如图所示模型示意图：

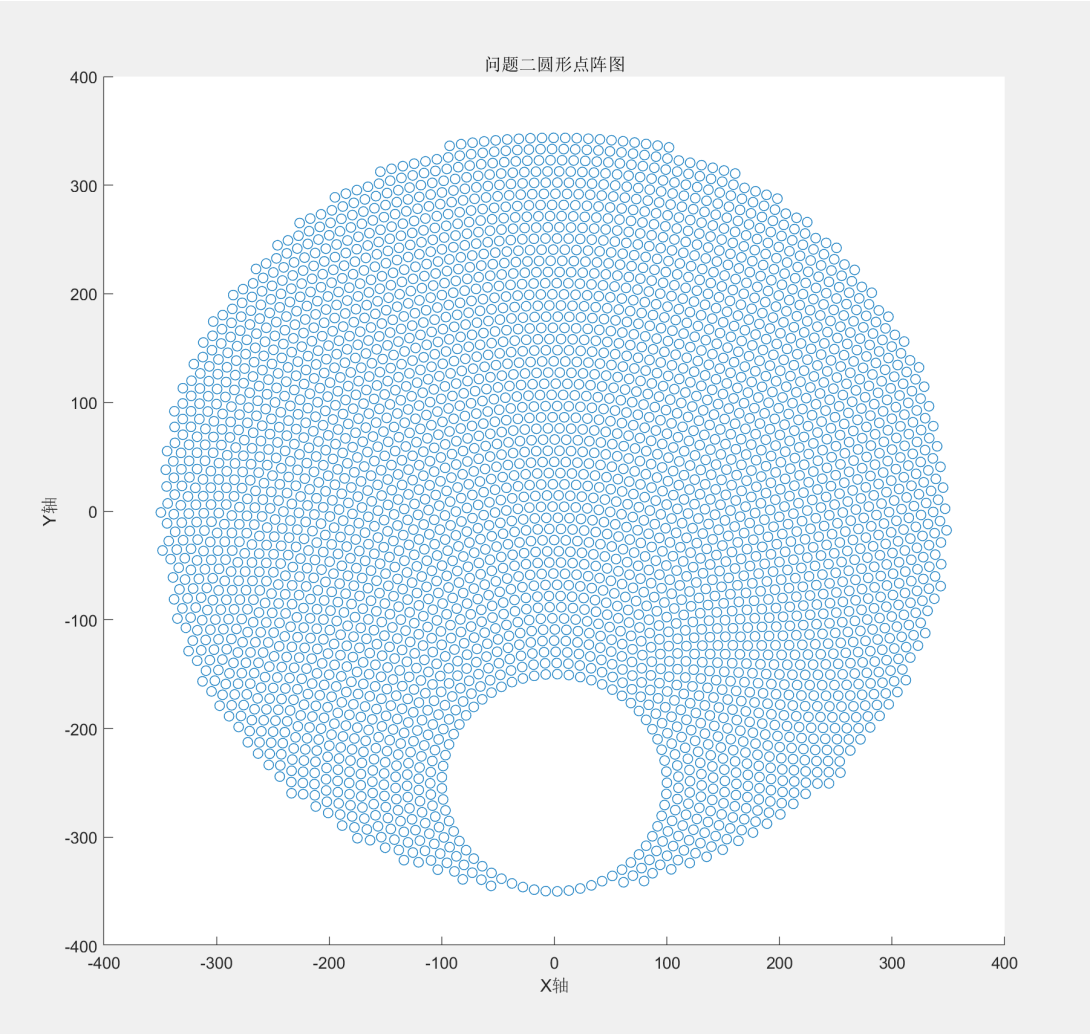

图 6　问题二优化得到的定日镜场模型

4.2.3 模型的最终输出结果

表 6 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	0.69614668	0.8986786	0.93166253	0.935945347	0.620992291
2 月 21 日	0.702377634	0.88936382	0.949151886	0.93663043	0.679344936
3 月 21 日	0.700888567	0.87296776	0.964468151	0.937077786	0.717967736
4 月 21 日	0.693576506	0.84828044	0.982135075	0.937122709	0.738515
5 月 21 日	0.682144337	0.82606975	0.992344157	0.936720457	0.739817426
6 月 21 日	0.675762488	0.81668643	0.994529578	0.936556106	0.737097587
7 月 21 日	0.682316609	0.82636382	0.992277922	0.936686116	0.739850273
8 月 21 日	0.693715687	0.84952634	0.981285991	0.936745963	0.737697961
9 月 21 日	0.701177934	0.87402945	0.963685656	0.937086199	0.716472802
10 月 21 日	0.702279065	0.89086642	0.947238453	0.936807968	0.673405065
11 月 21 日	0.694917526	0.89923203	0.929981274	0.935405795	0.614604917
12 月 21 日	0.690881956	0.90116207	0.923199529	0.934798775	0.58790579

表 7 问题二：年平均光学效率及输出功率表

年平均 光学效率	年平均 余弦效率	年平均阴影 遮挡效率	年平均 截断效率	年平均输出热 功率 (MW)	单位面积镜面平均 输出热功率 (kW/m^2)
0.693015416	0.866102244	0.96266335	0.936465304	64.86013475	0.691972649

表 8 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,-250)	5.2825×5.2825	4.598	3359	93732.2442

4.3 问题三的分析与求解

4.3.1 模型的分析

问题 3 在问题 2 的基础上，更改条件为定日镜的尺寸和安装高度可以不同，重新设计定日镜场的参数，使得定日镜场在达到要求的额定功率 60MW 的条件下，单位镜面面积年平均输出热功率尽量大。

决策变量：

• 每个定日镜的位置坐标（x_i, y_i）

• 吸收塔位置坐标（x', y'）

• 每个定日镜的尺寸 (高度 m_i 和宽度 n_i)

• 每个定日镜的安装高度 h_i

• 定日镜数目 N

• 相邻定日镜底座中心之间的距离 d

目标函数：

• 单位面积镜面年平均输出热功率尽量大

约束条件：

• 年平均输出热功率达到 60MW

• 吸收塔周围 100 m 范围内不安装定日镜，且所有定日镜的位置坐标必须在半径 350m 的圆形区域内建设

• 定日镜高度在 2m 到 8 米之间

• 定日镜宽度在 2m 到 8 米之间且不小于该定日镜高度

• 定日镜安装高度在 2m 到 6 米之间且不小于该定日镜高度的一半 (确保镜面在绕水平转轴旋转时不会触及地面)

• 相邻定日镜底座中心之间的距离比两镜面平均宽度多 5m 以上

4.3.2 模型的建立与求解

建立模型：

$$max \sum_{i=1}^{N} DNI \times \eta_i$$

$$s.t. \begin{cases} \sum_{i=1}^{N} DNI \times m_i \times n_i \times \eta_i \geqslant 60MW \\ |x_i|, |y_i| \leqslant 350,\ i \in [1, N] \\ (x_i - x')^2 + (y_i - y')^2 \geqslant 100^2,\ i \in [1, N] \\ m \in [2, 8] \\ n_i \in [m_i, 8],\ i \in [1, N] \\ h_i \in [max(2, \frac{m_i}{2}), 6],\ i \in [1, N] \\ d_i \geqslant \frac{n_i + n_{\text{相邻}}}{2} + 5,\ i \in [1, N] \end{cases} \tag{4-3-1}$$

我们发现这是在问题二的基础上新增了两个决策变量，这就导致变量的维度过高，直接使用优化算法将不能得出结果，于是我们在问题二所得最优解基础上，设计定日镜的较优生成模型

（1）高度生成模型

考虑到密集型定日镜塔吸收到的总太阳辐射能量高，且阴影遮挡对密集型定日镜塔影响较大，为了减小定日镜之间的相互遮挡与吸收塔对定日镜的遮挡，我们想到利用曲面来减少阴影遮挡 (越远离吸收塔，定日镜的安装高度越高)。

考虑到抛物面的光学性质，我们选择建立抛物面的物理模型，以求减少阴影和遮挡。

由第二问的经验知：吸收塔在偏移 y 轴-250 处，即 $Y_t = -250$ 时可以获得最大单位面积镜面平均输出热功率。先考虑二维平面，将吸收塔中心取为二维平面原点，建立抛物线方程：

$$z = a(y - \frac{Y_t}{2}) + b \tag{4-3-2}$$

由于定日镜的安装高度范围为 2m 到 6m，为了最大化的利用，我们取点 $(Y_t + 100, 2)$ 和点 $(350, 6)$。代入抛物线方程可求得 $a = 0.000017, b = 1.9995556$。

接下来将其转换为三维抛物面。首先将抛物线沿 y 轴正方向平移 Y_t, 再绕 z 轴进行旋转，再将整个抛物面平移回去，最终得到抛物面方程：

$$z = 0.000017[(x^2 + (y + 125)^2] + 1.9995556 \tag{4-3-3}$$

由于采用径向交错的辐射网络状圆形布局，沿径向越远，定日镜分布越稀疏。为最大化利用阳光，我们选择让定日镜的面积与定日镜安装位置到吸收塔中心的平面直线距离成正比。

（2）边长和位置

按照问题二的思路编写程序，生成立体点阵。与问题二不同的是，问题三中的镜面尺寸和安装高度是可变的。对于镜面尺寸，在假定镜子的边长和镜子到吸收器的平面直线距离成正比后，就可以按比例生成总圈数和每一圈的半径。在确定半径后，即可得出每圈的周长，按照镜面大小计算出每一圈能容纳的镜子个数。最后按照问题二的中的插空法生成镜子的 x 和 y 坐标。确定 x 和 y 坐标后，对于镜面高度，将相关参数带入式 (4-3-3) 即可求出镜面高度。

将得到的定日镜位置与高度数据作图，得到如图所示三维模型示意图：

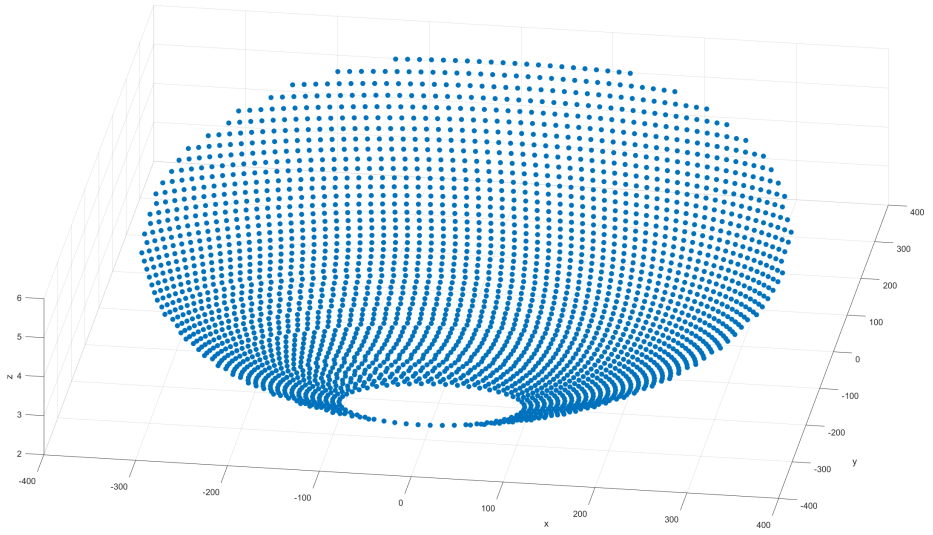

图 7　问题三优化得到的定日镜场三维模型

4.3.3 模型的最终输出结果

表 9　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/m^2)
1 月 21 日	0.682547013	0.89513548	0.953209514	0.900287573	0.60559597
2 月 21 日	0.686572675	0.8870304	0.966011541	0.90176115	0.66134979
3 月 21 日	0.683345794	0.87211718	0.976430624	0.903129684	0.69855876
4 月 21 日	0.673759856	0.84912564	0.988584793	0.903327175	0.71711503
5 月 21 日	0.662459516	0.82817799	0.995102588	0.904677164	0.71877125
6 月 21 日	0.656486338	0.8192802	0.996560799	0.904930569	0.71660527
7 月 21 日	0.662471004	0.82845644	0.995023676	0.904460503	0.718632
8 月 21 日	0.674568604	0.85029482	0.988023618	0.903680872	0.71699276
9 月 21 日	0.683697336	0.873095	0.976029099	0.902953625	0.69706504
10 月 21 日	0.68651295	0.88836754	0.964648552	0.901597629	0.6555136
11 月 21 日	0.681473075	0.89559489	0.951917001	0.899629807	0.59937985
12 月 21 日	0.677629904	0.89715242	0.946180225	0.898417677	0.57329438

表 10　问题三：年平均光学效率及输出功率表

年平均光学效率	年平均余弦效率	年平均阴影遮挡效率	年平均截断效率	年平均输出热功率 (MW)	单位面积镜面平均输出热功率 (kW/m^2)
0.675960339	0.865319	0.974810169	0.902404452	57.54715494	0.673239475

表 11　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 m^2
(0,-250)			2539	50720479.18

注：“定日镜尺寸”及“定日镜安装高度”见 result3.xlsx 文件。

五、 模型的误差分析

1. 使用梯度下降法优化模型时，由于初始参数的选取，模型本身的精度等问题，所得到的解可能与实际最优解有一定的误差。以梯度下降法得到的最优解再次进行单变量灵敏度分析时，控制变量选择最优解，最后的组合可能与多变量一起考虑的实际最优解有一定的误差。
2. 在计算阴影遮挡时，将太阳光视作平行光进行计算，没有考虑太阳光的实际锥形对结果的影响，这将会导致在计算阴影遮挡时有微小的误差存在。
3. 在计算具体数据时，由于计算机本身对浮点数的精度问题，得到的数据会有一定的误差；在利用优化模型得到较优的参数时，为了方便计算，在进一步验证计算参数时将数据控制在了五位小数范围内，因此会有一定的计算误差。
4. 在确定定日镜的位置时，基于“圆形定日镜场”进行分析，令定日镜的分布严格在圆周曲线上，而在实际情况中，无法精确按照圆周安置定日镜，且实际效率最高的定日镜塔分布方式不一定的圆形 (经查阅资料知可能是六边形)，因此本文所设计的方法在应用到实际时还需考虑更多因素。
5. 在建立物理模型时，有些是根据经验分析得到的方法，这将会导致存在一定的决策误差。

六、 模型的评价与改进

6.1 模型的优点

采用的镜面离散化模型和光线追踪模拟算法进行计算能保证较高的精准度。

模型优化时所用的梯度下降法有可广泛应用 (可以应用于各种机器学习和深度学习任务，包括线性回归、逻辑回归、神经网络等)、可解释 (数学原理清晰)、高效 (在大规模数据集上表现良好，可以通过小批量随机梯度下降（Mini-Batch Stochastic Gradient Descent）等技术来高效地处理大量数据)、有良好的收敛性、可扩展性、可并行化、易找到全局最优解等优点。

使用恰当的经验进行条件限制，简化了模型的建立。同时，抛物面模型比较有创新性，在与实际契合的前提下大大简化了运算。

6.2 模型的缺点

在编写程序求解问题一时，发现采用的镜面离散化模型使得计算复杂性过高，得出结果的时间使得直接优化变得不可能，于是在问题 2 中我们不得不采用随机方法简化计算量，这可能导致得出结果的精准度问题。

模型优化时所用的梯度下降法有收敛速度慢 (可能需要许多迭代才能收敛到最优解)、对超参数敏感 (需要调整学习率等超参数，不同的超参数选择可能会影响优化的效果)、内存占用大 (需要存储大量的参数和梯度信息，可能导致内存占用问题)、不适用于非光滑损失函数 (要求损失函数是可微的，对于非光滑或不可微的损失函数，需要使用其他优化方法) 等问题。

计算阴影遮挡时所用的平行光模型、优化模型参数的选择及浮点数计算的精度，都造成一定的误差。

6.3 模型的改进

问题二的定日镜排布算法可以尝试其他更多类型。

可以进一步增加参数，使定日镜的长和宽不同，可能会得到更好的结果。

可以进一步改进梯度下降优化算法，例如进行学习率调度、进行预训练和迁移学习等。

参考文献

[1] 张平, 奚正稳, 华文瀚等. 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(06):5-8.

[2] 杜宇航, 刘向民, 王兴平等. 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,2020,40(05):426-432.DOI:10.19805/j.cnki.jcspe.2020.05.012.

[3] 胡叶广. 塔式太阳能热发电系统的多级反射式聚光镜场的研究 [D]. 哈尔滨工业大学,2019.

[4] 胡中正. 俯仰—方位式双轴定日镜的自动校准方法研究 [D]. 北方工业大学,2019.

[5] 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,2023.DOI:10.27205/d.cnki.gltec.2022.001089.

[6] 程小龙. 基于光学效率的塔式电站镜场布局优化设计研究 [D]. 合肥工业大学,2019.

[7] 胡亮, 张平, 奚正稳等. 塔式光热发电聚光场截断效率的光线追迹和人工神经网络计算方法 [J]. 东方电气评论,2020,34(03):64-69+75.DOI:10.13661/j.cnki.issn1001-9006.2020.03.014.

[8] 程小龙, 尹延国, 马少波. 塔式电站定日镜场布局的优化设计研究 [J]. 能源与环境,2018(02):64-66+70.

附录 A 求解相关结果函数

```
import math
import numpy as np
import openpyxl
from matplotlib import pyplot as plt

def sin(x):
    return np.sin(x)

def cos(x):
    return np.cos(x)

D = {1:306, 2:337, 3:0, 4:31, 5:61, 6:92, 7:122, 8:153, 9:184,
    10:214, 11:245, 12:275} # key为月份, value为D的值
ST = {1:9, 2:10.5, 3:12, 4:13.5, 5:15} #
    key为时段代号, value为具体时间
latitude = math.pi * 39.4 / 180 # latitude纬度

def get_angle(sin, cos):
    angle = (np.angle(cos + 1j * sin) + 2 * math.pi) % (2 * math.pi)
    return angle

def get_sun_angle(time, month):
    sin_declination = math.sin(2 * math.pi * D[month] / 365) *
        math.sin(2 * math.pi * 23.45 / 360) # declination太阳赤纬角
    cos_declination = math.sqrt(1 - sin_declination ** 2)
    # cos_declination = math.cos(math.asin(sin_declination))
    hour = math.pi * (ST[time] - 12) / 12 # hour时角
    sin_altitude = cos_declination * math.cos(latitude) *
        math.cos(hour) + sin_declination * math.sin(
        latitude) # altitude太阳高度角

    cos_altitude = math.sqrt(1 - sin_altitude ** 2)
```

```
        # cos_altitude = math.cos(math.asin(sin_altitude))
        cos_orientation = round((sin_declination - sin_altitude *
            math.sin(latitude)) / (cos_altitude * math.cos(latitude)),
                                12) # orientation太阳方位角
        if ST[time] <= 12:
            sin_orientation = math.sqrt(1 - cos_orientation ** 2)
        else:
            sin_orientation = -math.sqrt(1 - cos_orientation ** 2)
        # sin_orientation = math.sin(math.acos(cos_orientation))
        e_sun = np.array([sin_orientation * cos_altitude, cos_orientation
            * cos_altitude, sin_altitude])

        return e_sun, get_angle(sin_altitude, cos_altitude),
            get_angle(sin_orientation, cos_orientation)


    def generate_points(Y):
        minsize = 4 # 镜子最小尺寸
        maxsize = 8 # 镜子最大尺寸
        sizerange = maxsize - minsize
        minh = 2 # 镜子最小安装高度
        maxh = 6 # 镜子最大安装高度
        hrange = maxh - minh
        furtest = abs(Y) + 350
        deltaY = float(Y) # 吸收塔Y偏移量
        deltaX = 0 # 吸收塔X偏移量
        center = np.array([deltaX, deltaY])
        # 计算圆周率
        pi = np.pi
        # 生成点列表
        points = []
        xlist = []
        ylist = []
        zlist = []
        sizelist = [] # 每个镜子的大小
```

```
    maxradius = furtest
    minradius = 100
    radius = [] # 每一圈的半径列表
    amount = [] # 每一圈的镜面数目
    gap = [] # 每一圈的间隔
    # 假设镜子的边长和距离成正比
    d = minsize + 5 # 圈与圈之间的间距
    dis = 100
    # 确定圈数和半径
    while dis <= maxradius:
        d = (dis - 100) * (sizerange / (maxradius - 100)) + minsize +
            5
        dis += d
        gap.append(d)
    # 确定每一圈的半径
    for i in range(0, len(gap)):
        gapsum = 0
        for j in range(0, i):
            gapsum += gap[j]
        distance = gapsum + minradius
        radius.append(distance)
    # 确定每一圈的镜面数目
    for c in range(0, len(radius)):
        length = 2 * pi * radius[c] # 周长
        # 已知弦长和半径，求弧长
        arclen = radius[c] * (2 * math.asin(d / (2 * radius[c])))
        num = int(length / arclen)
        amount.append(num)
    alpha = 0 # 每一圈的偏移量，用于插空
    for c in range(0, len(radius)):
        for i in range(0, amount[c]):
            x = center[0] + radius[c] * np.cos(2 * pi * i / amount[c]
                + alpha)
            y = center[1] + radius[c] * np.sin(2 * pi * i / amount[c]
                + alpha)
```

```
            size = 0
            if math.sqrt(x ** 2 + y ** 2) <= 350:
                xlist.append(x)
                ylist.append(y)
                z = 0.000017 * (x ** 2 + (y + 125) ** 2) + 1.9995556
                zlist.append(z)
                if gap[c] <= 13:
                    size = gap[c] - 5
                else:
                    size = 8
                sizelist.append(size)
                points.append((x, y, z, size))
        alpha += math.asin(d / (2 * radius[c]))
    return points

def get_mirror_angle(x, y, z, e_sun):
    e_rec = np.array([X_tower - x, Y_tower - y, H - z])
    e_rec = e_rec / np.linalg.norm(e_sun + e_rec)
    n_h = (e_sun + e_rec) / np.linalg.norm(e_sun + e_rec)
    test = np.dot(n_h, e_rec.reshape(-1, 1))
    sin_a = round(n_h[2], 12)
    cos_a = math.sqrt(1 - sin_a ** 2)
    sin_o = round(n_h[1] / cos_a, 6)
    cos_o = math.sqrt(1 - sin_o ** 2)
    return get_angle(sin_a, cos_a), get_angle(sin_o, cos_o), n_h #
        高度，方位


def get_DNI(h, sin_altitude):
    H0 = 3
    a = 0.4273 - 0.00821 * (6 - (H0 + h)) ** 2
    b = 0.5055 + 0.00595 * (6.5 - (H0 + h)) ** 2
    c = 0.2711 + 0.01858 * (2.5 - (H0 + h)) ** 2
    DNI = G0 * (a + b * math.exp(-c / sin_altitude))
    return DNI
```

```
def get_cos_loss(e_sun, n):
    return np.dot(e_sun, n.reshape(-1, 1))

def get_shadow_points(e_sun, x, y, z, altitude, orientation, e, x_b,
    y_b, z_b, altitude_b, orientation_b, L_b, W_b):
    C_A_CG = np.array([x, y, z]).reshape(-1, 1)
    C_B_CG = np.array([x_b, y_b, z_b]).reshape(-1, 1)
    # C_l0 坐标系下

    # C_l1坐标系
    VB_Cl1 = [np.array([L_b / 2, W_b / 2, 0]).reshape(-1, 1),
             np.array([-L_b / 2, W_b / 2, 0]).reshape(-1, 1),
             np.array([-L_b / 2, -W_b / 2, 0]).reshape(-1, 1),
             np.array([L_b / 2, -W_b / 2, 0]).reshape(-1, 1)]
    """T1 = np.array([[cos(orientation_b),sin(altitude_b)*
    sin(orientation_b),-sin(orientation_b)*cos(altitude_b)],
                 [-sin(orientation_b),cos(orientation_b)*
                 sin(altitude_b),-cos(orientation_b)*cos(altitude_b)],
                 [0,cos(altitude_b),sin(altitude_b)]])"""
    T1 = np.array([[math.sin(orientation_b), -math.sin(altitude_b) *
       math.cos(orientation_b),
                  math.cos(altitude_b) * math.cos(orientation_b)],
                 [-math.cos(orientation_b), -math.sin(altitude_b) *
                    math.sin(orientation_b),
                  math.cos(altitude_b) * math.sin(orientation_b)],
                 [0, math.cos(altitude_b), math.sin(altitude_b)]])
    VB_CG = [0, 0, 0, 0]
    for i in range(len(VB_Cl1)):
        VB_CG[i] = np.matmul(T1, VB_Cl1[i]) + C_B_CG
    PB_CG = [0, 0, 0, 0]
    e0 = e.reshape(-1, 1)
    for i in range(4):
        PB_CG[i] = VB_CG[i] + (np.dot(e, C_A_CG - VB_CG[i]) /
```

```
            np.dot(e_sun, e0)) * e_sun.reshape(-1, 1)
    '''T2 = np.array([[cos(orientation),-sin(orientation),0],
                  [sin(altitude)*sin(orientation),sin(altitude)*
                  cos(orientation),cos(altitude)],
                  [-cos(altitude)*sin(orientation),cos(altitude)*
                  cos(orientation),sin(altitude)]])'''
    T2 = np.array([[math.sin(orientation), -math.sin(altitude) *
       math.cos(orientation),
                   math.cos(altitude) * math.cos(orientation)],
                  [-math.cos(orientation), -math.sin(altitude) *
                     math.sin(orientation),
                   math.cos(altitude) * math.sin(orientation)],
                  [0, math.cos(altitude), math.sin(altitude)]])
    PB_CL0 = [0, 0, 0, 0]
    for i in range(4):
        PB_CL0[i] = np.matmul(T2, PB_CG[i] - C_A_CG)
    return PB_CL0
    # other_code

def get_shadow_loss(L, W, e_sun, x, y, z, altitude, orientation, e,
    id):
    I = int(W / point_distance)
    J = int(L / point_distance)
    Points = np.zeros((I, J, 3), dtype=float)
    for i in range(I):
        for j in range(J):
            Points[i][j][0] = -(L + point_distance) / 2 + j *
               point_distance
            Points[i][j][1] = -(W + point_distance) / 2 + i *
               point_distance
            Points[i][j][2] = 0

    list_around = []
    for i in range(id):
        if (data[i][0] - data[id][0]) * (data[i][0] - data[id][0]) +
```

```
            (data[i][1] - data[id][1]) * (
                data[i][1] - data[id][1]) <= MAX_DISTANCE:
            list_around.append(id)
    if (id > 0):
        list_around.append(id - 1)
    if (id < len(data) - 1):
        list_around.append(id + 1)
    P1 = []
    P2 = []
    P3 = []
    P4 = []
    for b in list_around:
        if mirror_angles[b] is None:
            mirror_angles[b] = get_mirror_angle(data[b][0],
                data[b][1], data[b][2], e_sun)
        PB = get_shadow_points(e_sun, x, y, z, altitude, orientation,
            e, data[b][0], data[b][1], data[b][2],
                            mirror_angles[b][0], mirror_angles[b][1],
                                W, L)
        P1.append(PB[0]);
        P2.append(PB[1]);
        P3.append(PB[2]);
        P4.append(PB[3])
        # 反射光线
        e_reflect = 2 * np.dot(e_sun, e.reshape(-1, 1)) * e - e_sun
        PB = get_shadow_points(e_reflect, x, y, z, altitude,
            orientation, e, data[b][0], data[b][1], data[b][2],
                            mirror_angles[b][0], mirror_angles[b][1],
                                W, L)
        P1.append(PB[0])
        P2.append(PB[1])
        P3.append(PB[2])
        P4.append(PB[3])
        percent, matrix = shadow_percent(Points, P1, P2, P3, P4)
    return percent, matrix, Points
```

```
# 先写一个函数来计算一个点 (A) 朝向一线段 (BC) 的张角
def cacul_angle(ax, ay, bx, by, cx, cy): # 接收三个点的坐标
    # ↑cacul_angle()函数用余弦定理来计算一个点对其他两个点的张角
    a = math.sqrt((bx - cx) ** 2 + (by - cy) ** 2)
    # ↑余弦定理需要三边长，如图，这是 ABC中和点A相对的那条边BC的长度a
    b = math.sqrt((cx - ax) ** 2 + (cy - ay) ** 2)
    # ↑这是 ABC中和点B相对的那条边AC的长度b, math.sqrt(x)是x的平方根
    c = math.sqrt((bx - ax) ** 2 + (by - ay) ** 2)
    # ↑这是 ABC中和点C相对的那条边AB的长度c
    alpha = math.acos(round((b ** 2 + c ** 2 - a ** 2) / (2 * b * c),
        12))
    # ↑余弦定理公式求出角
    return alpha # 返回 , 是一个弧度制的值

def if_cover(a_x, a_y, b_x, b_y, c_x, c_y, d_x, d_y, e_x, e_y):
    # ↑接收五个点，其中各个点的名字对应图上的各点
    # if_cover()函数用余弦定理判断给定点A点是否在凸四边形BCDE内
    """
     (beta)是点A对四边形四条边的张角，如果四个张角和为360°则显然A在四边
    形BCDE之内，如果张角和不为360°则A在四边形BCDE之外。
    """
    beta = cacul_angle(a_x, a_y, b_x, b_y, c_x, c_y) +
        cacul_angle(a_x, a_y, c_x, c_y, d_x, d_y) + \
            cacul_angle(a_x, a_y, d_x, d_y, e_x, e_y) +
                cacul_angle(a_x, a_y, e_x, e_y, b_x, b_y)

    if round(beta, 5) == round(math.pi * 2, 5): # 如果 等于2
        return True # 则返回True
    else:
        return False # 否则返回False

def shadow_percent(Points, p1, p2, p3, p4): #
    需要传入点矩阵 (numpy对象) , 四个顶点数组 (二维向量) 对象
    # 解析传入的参数
```

```
    x1 = []
    y1 = []
    x2 = []
    y2 = []
    x3 = []
    y3 = []
    x4 = []
    y4 = []
    length = len(p1)
    for n in range(0, length):
        x1.append(p1[n][0])
        y1.append(p1[n][1])
        x2.append(p2[n][0])
        y2.append(p2[n][1])
        x3.append(p3[n][0])
        y3.append(p3[n][1])
        x4.append(p4[n][0])
        y4.append(p4[n][1])
    rows = Points.shape[0] # 行数
    cols = Points.shape[1] # 列数
    shadows = 0
    # 创建01矩阵
    matrix = np.zeros((rows, cols))
    # 遍历point矩阵中的所有点
    for n in range(0, length):
        for i in range(0, rows):
            for j in range(0, cols):
                if if_cover(Points[i][j][0], Points[i][j][1], x1[n],
                    y1[n], x2[n], y2[n], x3[n], y3[n], x4[n],
                            y4[n]) and \
                        matrix[i][j] == 0:
                    matrix[i][j] = 1
    # 统计阴影部分占比
    for i in range(0, rows):
        for j in range(0, cols):
```

```
                if matrix[i][j] == 1:
                    shadows = shadows + 1
        percent = 1 - (shadows / (rows * cols))
        return percent, matrix

def get_break_effect(id, z, a, y, matrix, Points):
    sum_on = 0
    for i in range(RAND_NUM):
        o = 0.00465
        r = np.random.uniform(low=0.0, high=2 * math.pi, size=None)
        count = 0
        for i in range(len(Points)):
            for j in range(len(Points[0])):
                if matrix[i][j] == 0:
                    count += 1
                    if effect(id, z, o, r, Points[i][j], a, y,
                        mirror_angles[id][2]) > 0:
                        sum_on += 1
    if count == 0:
        return 0
    else:
        return sum_on / (RAND_NUM * count)

    def effect(id, z, o, r, H_1, a, y, V_n): #
        o为任意光线在光锥中与来自太阳中心的主光线的夹角,r为光线与Xs(光锥坐标
    系中, Xs轴与地面平行, 且垂直于沿着主光线方向朝向太阳圆盘中心的Zs轴)的夹角
        # V_s为该光线在光锥坐标系中的向量表示,V_n为定日镜的法线单位向量,a,
        y为光锥坐标系高度角和方向角
        R = 3.5 # 单位为米
        h = 8
        altitude, orientation = mirror_angles[id][0], mirror_angles[id][1]
        # orientation += math.pi/2
        y = math.pi / 2 - y
        V_s = np.array([math.sin(o) * math.cos(r), math.sin(o) *
            math.sin(r), math.cos(o)])
```

```
    T = np.array([[math.sin(y), -math.sin(a) * math.cos(y),
       math.cos(a) * math.cos(y)],
                  [-math.cos(y), -math.sin(a) * math.sin(y),
                     math.cos(a) * math.sin(y)],
                  [0, math.cos(a), math.sin(a)]])
    '''T= np.array([[math.cos(y), math.sin(a) * math.sin(y),
       -math.sin(y) * math.cos(a)],
                  [-math.sin(y), math.cos(y) * math.sin(a),
                     -math.cos(y) * math.cos(a)],
                  [0, math.cos(a), math.sin(a)]])'''
    '''T1 = np.array([[math.cos(orientation), math.sin(altitude) *
       math.sin(orientation), -math.sin(orientation) *
       math.cos(altitude)],
                  [-math.sin(orientation), math.cos(orientation) *
                     math.sin(altitude), -math.cos(orientation) *
                     math.cos(altitude)],
                  [0, math.cos(altitude), math.sin(altitude)]])'''
    T1 = np.array([[math.sin(orientation), -math.sin(altitude) *
       math.cos(orientation),
                   math.cos(altitude) * math.cos(orientation)],
                  [-math.cos(orientation), -math.sin(altitude) *
                     math.sin(orientation),
                   math.cos(altitude) * math.sin(orientation)],
                  [0, math.cos(altitude), math.sin(altitude)]])
    # 镜A到地面
    H_L = np.dot(T1, H_1) + np.array([data[id][0], data[id][1], z])
    H_L[0] -= X_tower
    H_L[1] -= Y_tower
    V_sl = np.dot(T, V_s)
    cos = np.dot(V_sl, V_n)
    V_R = 2 * cos * V_n - V_sl
    a = (V_R[0] ** 2 + V_R[1] ** 2)
    xx = V_R[1] * H_L[0] - V_R[0] * H_L[1]
    b = -2 * V_R[1] * (xx) / R
    c = (xx / R) ** 2 - V_R[0] ** 2
```

```
        derta = b ** 2 - 4 * a * c
        if derta < 0:
            return 0
        cos1 = round((-b + math.sqrt(derta)) / (2 * a), 12)
        cos2 = round((-b - math.sqrt(derta)) / (2 * a), 12)
        sin1 = (V_R[1] * (R * cos1 - H_L[0]) / V_R[0] + H_L[1]) / R
        sin2 = (V_R[1] * (R * cos2 - H_L[0]) / V_R[0] + H_L[1]) / R
        point = []
        z1 = V_R[2] * (R * cos1 - H_L[0]) / V_R[0]
        z2 = V_R[2] * (R * cos2 - H_L[0]) / V_R[0]
        if (z1 >= H - H_L[2] - h / 2) & (z1 <= H - H_L[2] + h / 2) & (z1
            >= H - H_L[2] - h / 2) & (
                z1 <= H - H_L[2] + h / 2):
            point.append((R * cos1, R * sin1))
        if (z2 >= H - H_L[2] - h / 2) & (z2 <= H - H_L[2] + h / 2) & (z2
            >= H - H_L[2] - h / 2) & (
                z2 <= H - H_L[2] + h / 2):
            point.append((R * cos2, R * sin2))
        return len(point)

    def get_air_effect(id, h):
        d_hr = min(math.sqrt(data[id][0] ** 2 + data[id][1] ** 2 + (H -
            h) ** 2), 1000)
        return 0.99321 - 0.0001176 * d_hr + (1.97 * 1e-8) * (d_hr ** 2)

    end
```