塔式定日镜场光学效率计算及布局优化

摘要

在近年低碳背景下，塔式太阳能光热电站受到广泛关注。为了最大化镜场的光学效率，合理的镜场布局至关重要。基于光学效率计算的镜场布局优化是这类电站设计的重要研究领域。

针对问题一，本文首先利用**光的反射定律**构建**定日镜追踪模型**。通过太阳高度角及方位角计算出入射光线的方向向量，由定日镜中心与集热器中心计算出反射光线的方向向量，从而得出定日镜每时刻的镜面法向量。通过构建镜面坐标系到地面坐标系的**旋转矩阵**，可以很方便地计算镜面坐标系下的点在地面的坐标。其次，本文根据定日镜场光学效率的定义式，建立**定日镜的光学效率模型**，分别计算大气透射率、余弦效率、阴影遮挡效率和集热器截断效率。其中余弦效率由定日镜法向量与入射光线单位向量点积取绝对值得到；阴影遮挡效率的计算采用**蒙德卡洛光线追迹法**，将镜面等距分为 36 个面元，计算入射与反射光线被塔和其他定日镜遮挡的面元的比例；截断效率同样采用光线追迹法，使用**角均分法**将太阳光锥分为若干子光线，并建立**光锥坐标系**到地面坐标系的变换矩阵，计算光锥中的子光线的直线方程与圆柱体集热器的相交比例得到截断效率。通过对 12 个典型日中的 5 个典型时算得**年平均光学效率约为 61.3%**，**余弦效率约为 75.6%**，**阴影遮挡效率约为 95.7%**，**截断效率约为 94.4%**，**输出功率约为 37.5MW**，**单位面积镜面输出热功率约为 59.6%**，并且用全年平均光学效率热力图进行结果的可视化分析。

针对问题二，本文提出**镜场布局参数优化模型**。先利用余弦效率和大气透射率估算吸收塔半径 700m 范围内的光学效率分布情况，再基于**定步长搜索**圈出半径为 350m 的范围光学效率最高的部分从而确定了吸收塔的最佳位置，得到了**吸收塔的最佳位置是位于偏心圆的底部**，然后选取阴影遮挡效率较高的 **Eliminate Blocking 布局**确定出各定日镜的位置。接着，本文采用**粒子群**的启发式算法确定全体定日镜高度和尺寸的最优参数。最终得到的**单位镜面面积年平均输出热功率为 1.06kW/m2**

针对问题三，考虑到高度差可以改变定日镜的阴影遮挡效率，本文在可变的定日镜高度和定日镜尺寸上提出几种调整方式：分别以等差高度或者等比数列调整定日镜每一排之间的高度差得到定日镜场径向最优的高度分布情况；通过计算**镜场中光学效率分布热力图**，通过将光学效率小的定日镜长宽乘以比例因子适当缩小的方法，增大单位面积镜面的年平均输出热功率。最优方案得到的**单位镜面面积年平均输出热功率为 1.153kW/m2**

关键字：**光学效率 蒙德卡洛光线追迹法 Eliminate Blocking 布局 粒子群算法**

目录

一　问题重述

1.1　问题背景

在全球能源供应清洁化、低碳化的背景下，太阳能热发电因兼具环保性、稳定性、可调节性和易于并网等特点近年来发展迅速。其中，塔式太阳能热发电站因具有聚光比大、热损耗小、系统容量大、发电效率高等特点，受到重点关注并实现了大规模的商业运营。塔式太阳能光热电站通过大量的定日镜聚集太阳光能，使低密度的太阳光能量反射到位于高塔顶部的集热器上聚集成为高密度的太阳光能量。这类光热电站的设计中，镜场的规划和布局至关重要，因为它直接决定了整个系统的能源利用效率。镜场的光学效率是一个关键性能指标，它定义为集热器接收到的光通量与镜场反射镜能够接收的最大光通量之比。

由于太阳在全年的不同时间和不同位置太阳高度角和方位角不同，定日镜需要不断调整反射姿态，从而保证其反射的太阳光能汇聚到集热器中心。此外，太阳光线为具有一定的锥形角度的锥形光线，当定日镜与集热器之间的距离较远时，镜场难以高效聚焦。如果将定日镜紧密排列以减小定日镜与集热器之间的距离，可能会增加定日镜之间的干涉，从而浪费一部分可用光线。因此，为了最大化光学效率，合理的镜场布局至关重要。基于光学效率计算的镜场布局优化是这类电站设计的重要研究领域。

1.2　问题重述

本题介绍了塔式太阳能光热发电站的布局设计、布局约束，以及传统基于方位角-高度角跟踪方式的定日镜基本工作原理。现计划在一个具体圆形区域建设圆形定日镜场，并给出了相关公式、简化条件和基本位置参数。本题将“年均”指标指定为每个月 21 日的五个具体时间点，共 60 个数据。

在问题一中，吸收塔位于定日镜场中心，规划的吸收塔高度为 $80m$，集热器采用高 $8m$、直径 $7m$ 的圆柱形外表受光式集热器。定日镜的尺寸、高度也被固定为 $6m \times 6m$，高度为 $4m$，要求根据公式算出镜场的年平均化学效率、年平均输出热功率，以及单位镜面面积年平均输出热功率。

问题二则只设定了镜场的额定年平均输出热功率，假定所有定日镜尺寸及高度参数一致，可以改变吸收塔位置、定日镜尺寸位置，要求设定定日镜场的其他参数，使得定日镜场在满足条件约束的前提下实现单位镜面面积的年平均输出功率最大化。其中，条件约束包括地理约束及功率约束：达到额定功率 $60MW$ 为功率约束，吸收塔周围 $100m$ 范围内不安装定日镜，定日镜的镜面宽度不小于镜面高度。镜面边长在 $2m$ 至 $8m$ 之间，安装高度在 $2m$ 至 $6m$ 之间，安装高度必须保证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。

问题三进一步去掉了问题二“定日镜尺寸及高度参数一致”的限制，需要在同样的前提下实现单位镜面面积的年平均输出功率最大化。

二　问题分析

本题给出定日镜的反射原理及计算定日镜场的效率公式、输出功率公式，几何模型清晰，计算目标明确。对于定日镜效率模型，分为五个分效率进行具体计算。其中涉及太阳高度角、方位角以确定阳光方向；涉及定日镜在镜场中的位置信息及镜面方向以考虑镜子之间的阴影遮挡效率，计算相互效应；涉及基于光锥的反射投影模型，以计算截断效率。

2.1　问题一分析

首先，在坐标系的建立上，本题为我们提供了以吸热塔为中心的镜场坐标系，对应在地面上直接观测太阳光线传播的情况；而定日镜在垂直、水平两个坐标轴下均有旋转，且光的反射定律及光线模拟在镜面坐标系下表示较为方便；基于远距离传播的光线锥体模型在光锥坐标系下便于计算截断效率，故本题需要我们在镜场系的基础上，建立定日镜的镜面坐标系及光锥坐标系进行辅助建模；由于在镜场坐标系中，太阳光射入定日镜时，光线向量的竖直分量恒为负，反射光线时竖直分量恒为正，故通过竖直分量可以明确区分入射、出射光线。

其次，本题将“全年平均”的连续性时间概念离散化，简化为共 60 个时间点（每月的 21 日中 5 个具体时间段）进行定日镜输出效率估计，故利于定日镜追踪及镜场效率的静态建模。根据公式可以获得每个时间点的太阳高度角、方位角。再利用定日镜中心反射太阳光线至集热器中心的原理，结合定日镜的地理位置信息可以得出每面镜子在 60 个时间点下的镜面法向量。于是，在固定时间点、固定地理位置的每面镜子的出入射光方向（主光方向）及镜面法向量都是唯一确定的，这利于我们针对离散的数据进行建模，确定全年平均的镜场功率数据。

接着，针对太阳光的锥形光线假设，在计算阴影损失时，阴影及遮挡面积远大于锥形光线的投影面积，故太阳光在计算遮挡和阴影时可以近似为平行光主光；而在计算截断效率时，按照光的反射定律，入射到定日镜的光锥角度为 $9.3mrad$，那么经定日镜反射出的光线锥角也应当为 $9.3mrad$。假设定日镜距离吸热面距离为 $100m$ 且垂直于反射光线的中轴，这样镜面上某一点反射到吸热面的光斑直径应约为 $0.94m$。这样的大小已经不能忽略了，所以在进行吸热面能量分布仿真时应当使用非平行光模型。

故需要假设固定的光锥夹角以计算截断效率。因此，针对不同的定日镜分效率计算，我们采取不同的光源假设。因此，针对各个分效率的计算，需要建立两个效率计算模型，分别计算基于平行光假设的阴影遮挡损失和基于光锥假设的截断损失。

综合上述分析，问题一需要建立镜面、锥形两个辅助坐标系，以离散时间点的具体方向数据进行静态几何建模。最后，利用题目给出的定日镜场输出热功率公式进行计算。

2.2 问题二、问题三分析

在问题一中，定日镜的尺寸及方位数目均为固定数值，我们只需要建立单个定日镜的效率计算模型即可代数得出结果；而在问题二中，结合镜场布局限制及额定功率条件，吸收塔的位置坐标、定日镜尺寸、安装高度、定日镜数目、定日镜位置均成为变量，需建立单位镜面面积年平均输出热功率有约束优化模型。注意到变量之间的关系：这就是说，本题需要我们寻找一个最优化的定日镜场布局模型。

由于变量间有相互依赖关系，本文采取先后确立的方式确定布局：首先需要确定吸收塔在镜场中的坐标；然后确定镜场内具体布局；最后确定其余尺寸、高度参数。其中，由于变量众多，直接确定新镜场布局不切实际，本文可以先调研已有的较高效的镜场布局，基于本题最优化指标选取一个最佳布局。

在问题三中，增加了定日镜镜面尺寸、高度为变量，此时本题成为了一个多变量非线性优化问题。我们针对问题二的结果进行定日镜高度、尺寸的调整，选用几个参数进行试验，挑选光学效率最高的一个作为结果。

综合上述分析，本文需要建立镜面、锥形两个辅助坐标系，以离散时间点的具体方向数据进行静态几何建模，利用逐级深入的问题结构求解约束优化问题。

三　模型假设

1. 定日镜的反射面近似为理想光滑的水平反射面：

   由于小正方形块之间相对于镜面的边长有较小的间隙，定日镜的反射面被认为是一个理想光滑的水平反射面。

2. 假设镜面中心与镜面支撑点重合：

3. 假设目标定日镜的阴影遮挡效率只受到距离最近的 8 台定日镜的影响：

   由于距离越近越容易产生遮挡，为简化计算，本文只考虑与自身相邻的八台定日镜的阴影遮挡影响。

4. 假设固定太阳高度角时辐射强度 DNI 不变：

   即不考虑天气影响，单位面积的辐射强度只与太阳高度角相关。

5. 在余弦效率、遮挡阴影效率计算中，本文假设太阳光线为平行光；在计算截断效率时，本文将入射到镜面的太阳光束近似成一个 $9.3mrad$ 的太阳光锥：

   在计算余弦效率及阴影损失时，由于可能产生遮挡的镜面距离较近，定日镜之间的间距远小于整个镜场的规模。经查阅，对于相同的定日镜，在投影到周围定日镜平面时，由光线发散导致的边缘增加量仅为 1mm 左右，远小于镜面尺寸，因此可以忽略太阳光的发散性；而在计算截断效率时，多考虑光从镜面到吸收塔的远距离传播，此时光线锥形角度产生的光斑被放大，其投影点及溢出状态是讨论截

断效率的首要因素。由于太阳辐射是以光锥的形式到达地表。根据日地距离 L 为 1.5×10^8 km，太阳的半径为 6.96×10^5 km，由于二者相差甚远, 所以光雉角度特别小。根据数学的估算原则, 在角度极小时, 正弦值无限接近角度值, 所以此时可计算太阳光雉角度为: $2\alpha_s = 2\sin\alpha_s = 2 * \frac{6.96\times10^5}{1.5\times10^8} = 9.3\text{mrad}$ ， 因此此时可近似太阳光锥角度为 $9.3mrad$。

6. 所有被追迹的光线所带的能量相等。

7. 吸收塔视为与集热器底面半径相同的圆柱体

8. 不考虑定日镜的精度或晃动可能造成光斑的偏移或溢出

四　符号说明

表 1: 符号说明

符号	描述	单位
ω	太阳时角	弧度
δ	太阳赤纬角	弧度
α_s	太阳高度角	弧度
γ_s	太阳方位角	弧度
H_z	集热器中心高度	m
h_0	镜面中心相对于镜场平面的高度为	m
O_A	定日镜中心坐标	无
O	集热器中心坐标	无
$S_{r,A}$	定日镜镜面中心指向集热器的光线的单位法向量	无
$S_{n,A}$	定日镜法向量	无
θ_z	定日镜俯仰角	弧度
θ_s	定日镜方位角	弧度
R_{mirror}	镜面坐标系至地面坐标系的旋转矩阵	无
η	定日镜场光学效率	无
η_{ref}	镜面反射率	无
η_{cos}	余弦效率	无
η_{at}	大气透射率	无
η_{trunc}	截断效率	无
η_{sb}	阴影遮挡效率	无
$\vec{i}$	入射光线反方向的单位向量	无
$\vec{V_H}$	一束光线在镜面坐标系中的向量	无
$\vec{V_0}$	在地面坐标系中的向量	无
R_{tower}	集热塔平面坐标系到地面平面坐标系的二维旋转矩阵	无
R_{cone}	锥形坐标系到镜场坐标系的旋转矩阵	无

五　模型的建立与求解

按照题目给出的镜场效率公式，分为五个分效率进行具体计算。其中涉及太阳高度角、方位角以确定阳光方向；涉及定日镜在镜场中的位置信息及镜面方向以考虑镜子之间的阴影遮挡效率，计算相互效应；涉及基于光锥的反射投影模型，以计算截断效率。

5.1 定日镜反射模型

塔式太阳能热发电站中，定日镜追踪太阳的运动轨迹，并将太阳光反射汇聚至高塔的集热装置上。对于题中所示的圆形定日镜场，处于不同位置的定日镜在同一时刻的镜面法向量与俯仰角均不相同。随着太阳位置的变化，镜场中的每一面定日镜都需要依靠底座的纵向转轴和水平转轴实时调整自身的俯仰角和方位角，这就要求掌握太阳实时的位置信息，对定日镜的跟踪反射过程进行建模。本文先根据题目给出的太阳高度角和方位角计算公式，计算出太阳光在每年内的不同时间、每天内的不同时刻的入射方向，再利用反射定律计算出定日镜的镜面法向量，依据法向量计算出定日镜的俯仰角与方位角。

5.1.1 太阳高度角与方位角的计算

首先计算出太阳时角 ω:

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

其中 ST 为当地时间。再计算出太阳赤纬角 δ:

$$\delta = \arcsin\left(\sin(\frac{2\pi D}{365})\sin(\frac{2\pi}{360}23.45)\right) \tag{2}$$

上式中 D 为以春分作为第 0 天起算的天数，本文假设 3 月 21 日为春分日。设 φ 为当地纬度，太阳高度角 $\alpha_s \in \left[0, \frac{\pi}{2}\right]$ 且满足

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

太阳方位角 γ_s 满足

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

对于北半球，上午太阳方位角 $\gamma_s \in (\frac{\pi}{2}, \pi)$，下午 $\gamma_s \in (\pi, \frac{3\pi}{2})$ 因此对(4)式求反余弦函数得到太阳方位角 γ_s 时需结合当地时间 ST 判断此时角度与 π 的大小关系。

5.1.2 定日镜镜面法向量及俯仰角的计算

因为定日镜场的综合效率并不是各个单一定日镜效率的简单叠加或相乘，尤其在阴影遮挡效率的计算过程中还需要将目标定日镜向其他定日镜所在的平面进行投影来确定是否发生阴影或遮挡，因此建立定日镜场效率模型的第一步就是计算定日镜的相关参数及顶点坐标。

1. 建立镜场坐标系

   由题目信息，塔式光热电站的定日镜场以吸热塔为中心进行周向布置。以圆形区域中心为原点，正东方向为 x 轴正向，北方为 y 轴正向，垂直于地面向上方 z 轴正向建立坐标系，称为镜场坐标系。设集热器中心的高度为 H_z，镜面中心相对于

镜场平面的高度为 h_0，定日镜镜面中心点位于基座的中轴线上，则集热器与定日镜的中心坐标分别可表示为 $O(0,0,H_z)$ 和 $O_A(x_{o,A},y_{o,A},z_{o,A})$。其中,$x_{o,A}$ 与 $y_{o,A}$ 是定日镜在镜场平面内的坐标，$z_{o,A}$ 表示镜面中心相对于镜场平面的高度，如图1所示。

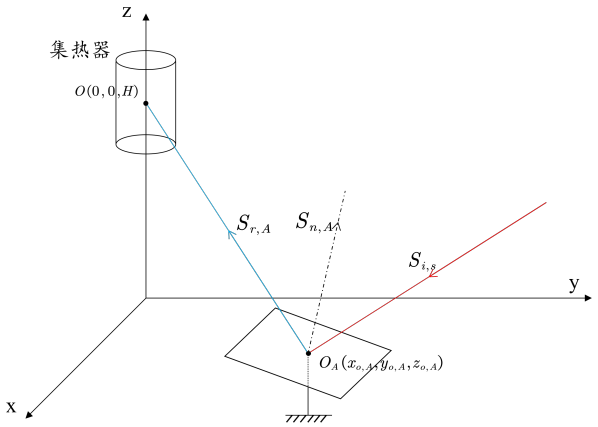

图 1: 定日镜与集热器

2. 利用反射定律计算定日镜的俯仰角

对于任意一台定日镜来讲，来自太阳的光线经过镜面反射后必然指向集热器中心。因此定日镜镜面中心指向集热器的光线的单位法向量 $S_{r,A}$ 可表示为:

$$S_{r,A}=\frac{O-O_A}{|O-O_A|}=\frac{(-x_{o,A},-y_{o,A},H_z-z_{o,A})}{\sqrt{x_{o,A}^2+y_{o,A}^2+(H_z-z_{o,A})^2}} \tag{5}$$

通过天文公式和定日镜经纬度可以求出太阳的高度角和方位角，假设某时刻太阳的高度角为 α_s 太阳的方位角为 γ_s。可知在确定的时间点上，太阳光线的单位法向量也是确定的。

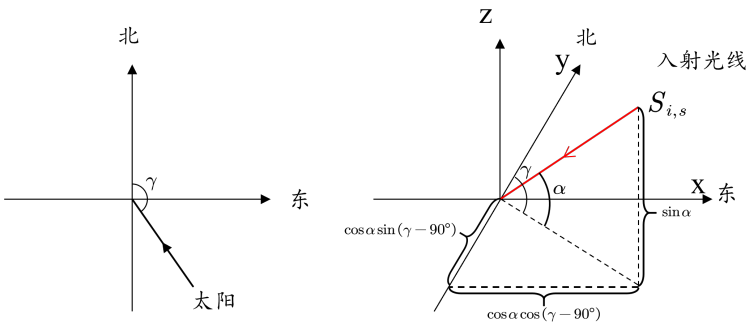

图 2: 由太阳高度角与方位角计算入射光的方向向量

如图2所示，若入射光线的单位向量表示为 $\vec{S_{i,s}}=(x_i,y_i,z_i)$，则 $\vec{S_{i,s}}$ 在镜场坐标系中的计算公式为:

$$\begin{cases} x_i = -\cos(\alpha_s)\cos(90^\circ - \gamma_s) \\ y_i = \cos(\alpha_s)\sin(90^\circ - \gamma_s) \\ z_i = -\sin(\alpha_s) \end{cases} \tag{6}$$

根据光反射定律，镜面法向量与入射、反射法向量共面且平分入射光线和反射光线之间的夹角。因此可由入射光线法向量和反射光线法向量推导出定日镜的镜面法向量 $S_{n,A}(x_n, y_n, z_n)$

$$S_{n,A} = \frac{S_{r,A} - S_{i,s}}{|S_{r,A} - S_{i,s}|} \tag{7}$$

由定日镜的镜面法向量 $S_{n,A}(x_n, y_n, z_n)$ 进而可计算得出定日镜的俯仰角 θ_z 和方位角 θ_s：

$$\begin{cases} \tan\theta_z = \frac{\sqrt{x_n^2+y_n^2}}{z_n} \\ \sin\theta_s = \frac{x}{\sqrt{x_n^2+y_n^2}} \end{cases} \tag{8}$$

5.1.3 旋转矩阵的计算

在运动学中的欧拉旋转定理指出，若一个刚体运动时，其内部至少有一点固定不动，则这个运动可被认为是围绕这个固定点对应固定轴的旋转运动。旋转过程中，向量的方向发生变化，而向量的大小不会改变，两个旋转矩阵相乘的结果仍为旋转矩阵。

在三维空间中，绕 X 轴旋转矩阵表达式为：

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{9}$$

绕 Y 轴旋转矩阵表达式为：

$$R_y(\alpha) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{10}$$

绕 Z 轴旋转矩阵表达式为：

$$R_z(\alpha) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

根据上文求出的定日镜的俯仰角 θ_z 和方位角 θ_s ，我们可以得到镜面坐标系至地面坐标系的旋转矩阵 R_{mirror}：

$$R_{mirror} = R_z(B) \times R_y(A) = \begin{bmatrix} cos\theta_s cos\theta_t & -sin\theta_t & cos\theta_t sin\theta_s \\ cos\theta_s sin\theta_t & cos\theta_t & sin\theta_s sin\theta_t \\ -sin\theta_s & 0 & cos\theta_s \end{bmatrix} \tag{12}$$

以法向量的旋转变换为例，镜面的朝向可看做原本水平朝上的正放定日镜先绕 Y 轴（朝向正北方向）旋转 θ_s（方位角），再绕 Z 轴旋转 θ_z，将旋转矩阵左乘设镜面法向量为 $\vec{n}$, 我们取正上方单位向量为 $\vec{z} = (0, 0, 1)$，此时定日镜镜面转动后的法向量为:

$$\vec{n} = R_z(B) \times R_y(A) \times \vec{z} \tag{13}$$

代入到旋转矩阵后得到:

$$\vec{n} = \begin{bmatrix} \sin\theta_s \cos\theta_t \\ \sin\theta_s \sin\theta_t \\ \cos\theta_s \end{bmatrix} \tag{14}$$

5.2 定日镜的光学效率模型

太阳光经由定日镜场一次反射后传播到集热器表面或吸热腔口，低能量密度的太阳光被定日镜场汇集为高能量密度的光束后在集热器表面由太阳辐射能转化为热能。其中，能被集热器接受到的能量与入射光线的总能量的比值称为定日镜场的光学效率。定日镜场的光学效率受到太阳位置、定日镜排布规则、大气条件、定日镜材质、发电站地理位置、集热器形状等因素的影响，是衡量一个定日镜场聚光能力的重要标准。题目给出的计算公式如下:

$$\eta = \eta_{\text{sb}} \eta_{\text{cos}} \eta_{\text{at}} \eta_{\text{trunc}} \eta_{\text{ref}} \tag{15}$$

在上式中，定日镜场的光学效率等于以下各部分的乘积:

1. 镜面反射率: 镜面反射率 η_{ref} 代表反射镜的反射率以及镜面脏污程度的综合效率，本文将 η_{ref} 取为常数 0.92。

2. 余弦损失: 余弦效率 $\eta_{\text{cos}} = 1-$ 余弦损失, 是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。

3. 大气衰减损失: 太阳光在空气传播的过程中，由于空气中的粉尘、颗粒等对光线造成了一定的衰减，因此从镜面反射到集热器的光路径中，因镜面与反射目标点的距离，其损失也略有不同。题目已给出大气透射率 η_{at} 的计算公式。

4. 阴影挡光损失: 阴影遮挡效率 $\eta_{\text{sb}} = 1-$ 阴影遮挡损失, 代表在镜场中定日镜相互遮挡造成的镜场接收到的太阳光线损失。

5. 溢出损失：也叫截断损失。当镜场反射的光斑落入集热器接受面时，由于光斑的特征以及定日镜的精度或晃动可能造成光斑的偏移或溢出，导致有部分光照射在集热器之外，这部分溢出光斑就形成了溢出损失。集热器截断效率 $\eta_{\text{trunc}} = \frac{\text{集热器接收能量}}{\text{镜面全反射能量} - \text{阴影遮挡损失能量}}$ 主要针对除去阴影挡光部分进行研究讨论。

5.2.1 大气透射率的计算

塔式系统主要依靠 DNI（直接辐射）进行发电，是电站选址的重要因素，大气透射衰减效率是影响其大小的直接因素。太阳光经由定日镜聚焦反射后传播到集热器的表面或采光口的过程中，太阳辐射能会因大气散射或被空气中的杂质颗粒物阻挡而造成能量损失。大气衰减效率 η_{at} 的值与反射距离 d_{HR} 以及当地的空气净度还有气压、空气温湿度等有关，因此大气衰减效率的值与距离并非线性相关。本文按照如下公式计算：

$$\eta_{\text{at}} = 0.99321 - 0.0001176 d_{\text{HR}} + 1.97 \times 10^{-8} \times d_{\text{HR}}^2 \quad (d_{\text{HR}} \leq 1000) \tag{16}$$

其中，d_{HR} 为反射距离，由 $\sqrt{x^2 + y^2 + (H - h_0)^2}$

5.2.2 余弦效率的计算

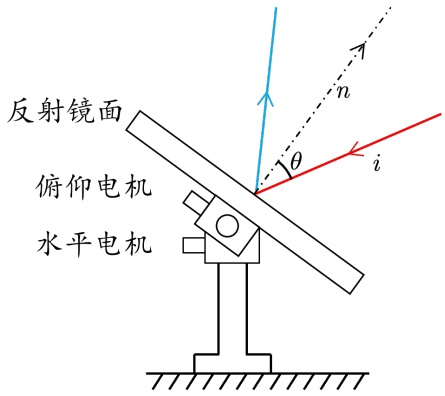

图 3: 余弦效率阐释图

如图3所示，由于要保证来自太阳的光线经过镜面反射后必然指向集热器中心，入射光线和镜面反射点的法线方向之间必然会存在一个夹角 θ，该夹角的余弦值 $cos(\theta)$ 即为该点的余弦效率值。假设整个镜面为一个理想的平面，此时在镜面上的每一个反射点拥有相同的镜面法向量。因此单一定日镜的余弦效率可以通过入射光线的单位向量与反射光线的单位向量的点积计算：

$$\eta_{\cos} = \cos\theta = |\vec{i} \cdot \vec{n}| \tag{17}$$

上式中 $\vec{i}$ 为入射光线单位向量 $S_{i,s} = (x_i, y_i, z_i)$，由(6)已计算得到。$\vec{n}$ 为镜面法向量，已在(7)中计算得出。

对单一定日镜来说，根据光反射定律及共面原理，入射光线与镜面法向量之间的夹角越小余弦效率越高。当且仅当定日镜中心、集热器中心和太阳三点构成一条直线时，入射光线、反射光线以及镜面法向量三者共线，定日镜的余弦效率值才会达到理论最大值。

5.2.3 阴影遮挡效率建模

由阴影遮挡效率公式，本文需要计算阴影遮挡损失。根据问题分析一节，在计算阴影损失时，由于可能产生遮挡的镜面距离较近，不涉及光线的远距离传播，阴影及遮挡面积远大于锥形光线产生的光斑面积，故太阳光在计算遮挡和阴影时可以近似为平行光。阴影损失由三部分组成，塔对镜场造成的阴影损失；后排定日镜接收的太阳光被前方定日镜所阻挡造成的阴影损失；后排定日镜在反射太阳光时被前方定日镜阻挡而未到达集热器上的挡光损失。

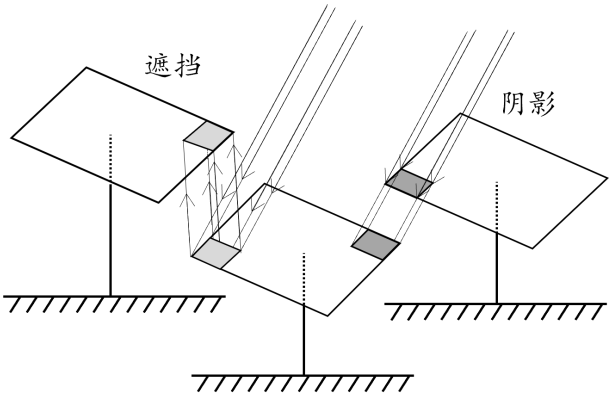

图 4: 遮挡阴影损失

图4展示了定日镜之间遮挡和阴影的影响：在镜场中定日镜的阴影落入另一个定日镜的镜面，或者反射光线照射到另一台定日镜的背面等，无可避免地镜场接收到的太阳光线有所损失。本文将前者单独考虑，后两种损失合并为“镜间损失”考虑。对于镜间损失，本文定义阴影损失为前排镜子的阴影落入后排镜子镜面产生的损失；遮挡损失为后排镜子反射光线时受到前排镜子遮挡，光线无法传播至集热器。

本文利用蒙德卡洛模拟法，将一块定日镜分成 6×6 的 36 个区域，取每个区域的中心点作为光线反射点，进行阴影损失的估计。

1. 定日镜间互相遮挡造成的阴影损失:

   根据在计算遮挡阴影时太阳光近似平行的假设，如图，以定日镜中心为原点，建立镜面坐标系。由上节的旋转矩阵，我们得到镜面坐标系到地面坐标系的旋转矩阵为 R_{mirror}, 一束光线在镜面坐标系中的向量表示为 $\vec{V_H}$，在地面坐标系中的向量表示为 $\vec{V_0}$, 则有 $\vec{V_0} = R_{mirror} \cdot \vec{V_H}$

   计算镜间阴影挡光损失的过程，实际上就是计算 A 镜的任意一点顺着入射光线或反射光线的相反方向是否会落入 B 镜区域内，并求出其在 B 镜的镜面坐标值。

设 A 镜的某一点 $H1(x_1, y_1)$, 要计算 $H2(x_2\ y_2)$ 的值，分为以下几个步骤:

(I) 先将 $H1$ 转换到地面坐标系下的坐标，并表示为 $H'1$

(II) 再将地面坐标系下的 $H'1$ 转换到 B 镜坐标系下，表示为 $H''1$。

(III) 将地面坐标系下的光线 (入射或反射) 转换到 B 镜坐标系下。

(IV) 在 B 镜坐标系下，根据两点一线的原理，计算光线与 B 镜交点 $H2$。

(V) 利用 B 镜判断 $H2$ 是否在镜内。

2. 塔对镜场造成的阴影损失

要计算塔的阴影损失，由于固定时刻的太阳光入射方向一定，首先需要计算出在 60 个时间点阳光的方向下，在水平面方向上塔在当下时间内影响的镜场区域，再在其中进行阴影判断。本文将此损失计算分为以下三个步骤:

(I) 基于太阳高度角 α_s 的阴影面积确定

(II) 基于太阳方位角的塔平面建系;

(III) 利用光线在塔平面的焦点判断阴影状况

定日镜之间的反射光线的遮挡问题可通过增加间距解决；对于入射光线遮挡的问题，太阳高度角是主导因素，在日出日落前后尤为明显。此外，定日镜的控制角度、尺寸及坐标等参数也是影响其效率的关键因素。

5.2.4 阴影遮挡损失计算

我们利用蒙德卡洛模拟法，将一块定日镜分成 6×6 的 36 个区域，取每个区域的中心点作为光线反射点，进行阴影遮挡损失百分比的计算。

1. 镜间相互损失计算

利用已求出的旋转矩阵，依照上述过程分步计算:

(a) 将 $H_1\,(x_1, y_1, 0)$ 转换为 H_1' :

$$H_1' = \begin{pmatrix} l_x & l_y & l_x \\ m_x & m_{y'} & m_x \\ n_x & n_y & n_x \end{pmatrix} \cdot H_1 + O_b = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \tag{18}$$

其中，O_b 为 A 镜坐标系原点在地面坐标系的坐标F, 表示为 (x_b, y_b, z_b)

(b) $H_1'\,(x_1', y_1', z_1')$ 转换为 H_1'' :

$$H_1'' = \begin{pmatrix} l_x & l_y & l_x \\ m_x & m_y & m_x \\ n_x & n_y & n_x \end{pmatrix}^T \cdot (H_1' - O_B) = \begin{pmatrix} x'' \\ y''_1 \\ z''_1 \end{pmatrix} \tag{19}$$

其中，O_B 为 B 镜坐标系原点在地面坐标系的坐标值，表示为 (x_B, y_B, z_B)

(c) 将光线在地面坐标系下的向量 $\overrightarrow{V_0}$ 转换到 B 镜坐标下，表示为 $\overrightarrow{V_H}$：

$$\overrightarrow{V_H} = \begin{pmatrix} l_x & l_y & l_x \\ m_x & m_y & m_x \\ n_x & n_y & n_x \end{pmatrix}^T \cdot \overrightarrow{V_0} = (a, b, c) \tag{20}$$

已知 B 镜坐标柔下 $H''_1 = (x''_1, y''_1, z''_1)$ 点、光线向量 $\overrightarrow{V_H} = (a, b, c)$，

求 $H_2 = (x_2, y_2, 0)$

解:

$$\frac{x_2 - x''_1}{a} = \frac{y_2 - y''_1}{b} = \frac{-z''}{c}$$

$$\text{得:} \begin{cases} x_2 = \frac{cx''_1 - az''}{c} \\ y_2 = \frac{c''_1 - bz''}{c} \end{cases} \tag{21}$$

(d) 判断 H_2 是否落入镜面范围内。

2. 塔对镜场造成的阴影损失计算

(a) 基于太阳高度角 α_s 的阴影面积确定

取圆柱集热器顶端水平圆形表面垂直于太阳光入射的直径，沿太阳光入射方向投影至镜场地面，圆柱集热器中心 $O(0, 0, H_z)$，半径为 R，故结合高度角 α_s，影响到矩形区域长为 $\frac{H_z}{tan(\alpha_s)}$，宽为 $2R$，角度沿塔中心逆时针旋转 $180° - \gamma_s$，如平面图5所示。

(b) 基于太阳方位角的塔平面建系

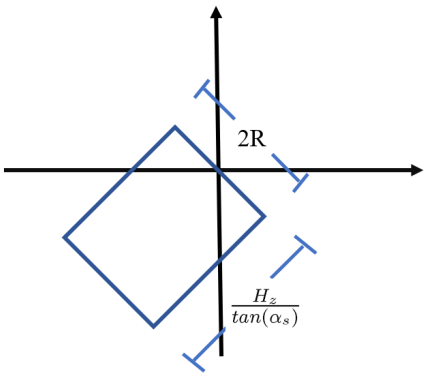

图 5: 集热器平面旋转示意图

基于太阳方位角 γ_s，以吸收塔中心为原点，考虑水平面方向上的旋转变换：将镜场坐标系的正北方向 Y 轴旋转为以太阳光入射方向为 Y 轴，建立旋转后的塔平面二维坐标系，即构建沿塔中心逆时针旋转 $180° - \gamma_s$ 的塔平面坐标系，其旋转矩阵如下：

$$R_{tower} = \begin{bmatrix} \cos\gamma_s & -\sin\gamma_s \\ \sin\gamma_s & -\cos\gamma_s \end{bmatrix} \tag{22}$$

(c) 利用光线在塔平面的焦点判断阴影状况同理，利用三点一线原理，计算在镜面系下，射向定日镜上每一点的光与塔平面的交点, 判断是否在塔的区域内。

5.2.5 截断效率模型

截断效率定义为: 集热器截获的能量占镜场汇聚能量的百分比。其中，所谓“镜场汇聚的能量”应是整个镜场能够反射出的太阳光能量，这就要除去在前一小节讨论过的被阴影或遮挡等损失的太阳光能量。由于太阳光线并非平行光线，而是一束半展角为4.65mrad 的锥形光线，由反射定律可知，锥形太阳光照到定日镜后反射的光线仍为锥形光线，随着定日镜到吸收塔的距离的增大，光线打到集热器形成的光斑面积也会增大，这会导致部分光线溢出了集热器所能吸收的范围，从而产生了截断效率的问题。

设锥形光线的光锥轴线为锥形光线的主光线。在不同时刻，经过定日镜中心的反射光线的主光线一定经过集热器中心，而集热器的位置与每个定日镜的位置坐标都是固定的，因此同一面镜子所有面元上反射光线主光线的方向都是确定的且互相平行，所以反射光线的方向只与定日镜的地理位置相关，与时刻无关。因此，计算截断效率也就是分别计算镜子上每个面元在固定的方向上的光锥产生的光斑情况。沿用5.2.3的分割方法，将一块定日镜的长宽等间隔分成 6 份，形成 6×6 的 36 个面元，分别计算每个面元的反射光锥与圆柱体集热塔的相交情况，进行截断损失的估计。

对于每一个面元反射出的锥形光线，采用角均分法将一束光锥等分为若干光线，将光锥周向方向与半角展宽方向以均匀角度的步长进行划分，图6中，右侧图为最终划分出的光线示意图，将光锥顶点与锥体底面面元的连线即为光锥中的一束光线。

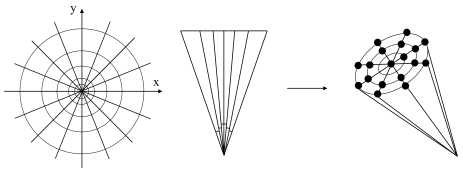

图 6: 角等分法将光锥分成若干光线

为研究投影在集热器的光斑是否溢出，本文对定日镜上的每一个面元、每一个面元反射出的每一条锥形光线使用蒙德卡洛光线追迹法计算溢出的光线占所有光线的比例。光线追迹法是一种统计学的方法，它主要用来追迹太阳入射的光线经镜面的反射后最终到达集热器计算区域光线的数目，且其统计精度主要决定于追迹光线的数量。对于某一个位置的定日镜，反射光锥主光线方向单位向量为定日镜中心到集热器中心的单位向量，整个光锥近似为一个全角为 $9.3mrad$ 的圆锥进行计算。计算光锥中的一束光线是否能照射到圆柱体集热器分为以下三个步骤:

1. 光锥坐标系的建立

为了求光锥中某一根光线的表达式，建立一个光锥坐标系 $O_S\ X_SY_SZ_S$：

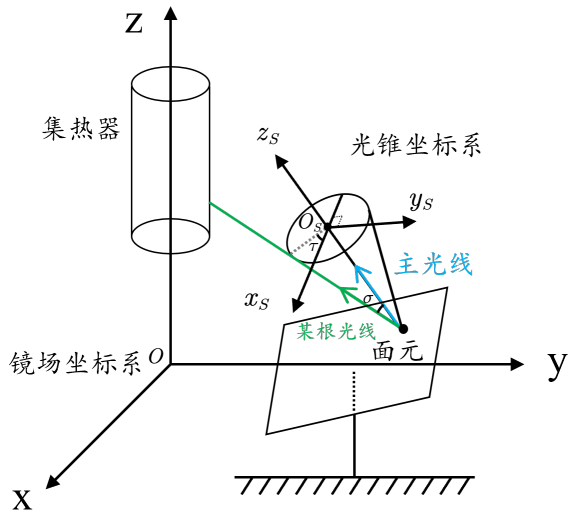

图 7: 光锥坐标系

如图，以主光线方向为光锥坐标系的 z_s 轴。光锥的底面为垂直于主光线的圆面，圆心为 O_s，过圆心做与地面平行（即平行于镜场坐标系 xOy 轴的平面）的平面与光锥底面的交线为光锥坐标系的 x_s 轴，显然 x_s 轴也与地面平行。光锥坐标系的 y_s 轴垂直于 x_s 轴与 z_s 轴，可由于 x_s 轴的方向向量与 z_s 轴的方向向量叉乘得出。

下面计算光锥中的一束光线在光锥坐标系下的方向向量：假设任一光线在光锥中与主光线的夹角为 σ，其在 $x_sO_sy_s$ 平面的投影与 x_s 轴的夹角为 τ，则任一光线在光锥坐标系下的表达式为：

$$\vec{S_s} = (\sin\sigma\cos\tau, \sin\sigma\sin\tau, \cos\sigma) \tag{23}$$

设光锥坐标系的 x_s 轴与镜场系的 x 轴夹角为 θ，则 x_s 的方向向量 $\vec{r_x}$ 在镜场坐标系下可表示为：

$$\vec{r_x} = (cos\theta, sin\theta, 0) \tag{24}$$

主光线单位向量为 $\vec{r_z}$，其数值上等于(5)，$\vec{r_x}$ 与 $\vec{r_z}$ 垂直所以满足：

$$\vec{r_x} \cdot \vec{r_z} = 0 \tag{25}$$

因为 $\vec{r_z} = S_{r,A}$ 联立方程(5)(25)，可以求出 θ y_s 轴的方向向量 $\vec{r_y}$ 满足

$$\vec{r_y} = \vec{r_x} \times \vec{r_z} \tag{26}$$

由上面计算出的列向量 $\vec{r_x}, \vec{r_y}, \vec{r_z}$ 得到锥形坐标系到镜场坐标系的旋转矩阵 R_{cone}：

$$R_{cone} = \begin{bmatrix} \vec{r_x} & \vec{r_y} & \vec{r_z} \end{bmatrix} \tag{27}$$

2. 在镜场系下联立方程，判断是否溢出

首先将光锥坐标系的光线向量 $\vec{S_s}$ 转化至镜场坐标系下的 $\vec{S_O}(m,n,l)$:

$$\vec{S_O} = R_{cone} \cdot \vec{S_s} \tag{28}$$

反射光线的方程为:

$$\frac{x-x_1}{m} = \frac{y-y_1}{n} = \frac{z}{l} \tag{29}$$

其中，x_1, y_1 面元在镜场坐标系下的坐标。镜场系的圆形集热器满足如下方程

$$\begin{cases} x^2+y^2 = R^2 \\ z \in \left[H_t - \frac{h}{2}, H_t + \frac{h}{2}\right] \end{cases} \tag{30}$$

其中，H_t 为吸收塔高度，h 为集热器。若方程有约束条件的解，则光线落入集热器内，否则为溢出。

5.3 定日镜场的输出热功率计算

5.3.1 法向直接辐射强度 DNI 计算

由题目法向直接辐射辐照度 DNI (单位: $\mathrm{kW/m^2}$) 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，计算公式如下:

$$\mathrm{DNI} = G_0 \left[a + b \exp\left(-\frac{c}{\sin \alpha_S} \right) \right] \tag{31}$$

$$a = 0.4237 - 0.00821(6-H)^2 \tag{32}$$

$$b = 0.5055 + 0.00595(6.5-H)^2 \tag{33}$$

$$c = 0.2711 + 0.01858(2.5-H)^2 \tag{34}$$

其中，G_0 为太阳常数，本题中取值为 $1.366kW/m^2$，H 为海拔高度，单位为 km，α_S 为太阳高度角

5.3.2 输出热功率计算

定日镜场的输出热功率 E_{field} 为

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

方程(35)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: $\mathrm{m^2}$) ; η_i 为第 i 面镜子的光学效率。

因此，只需要将计算出的每面镜子效率与对应的采光面积相乘加和，再乘以对应的 DNI，即可得到相应功率。若要计算平均面积的功率，则再除以总面积即可。

5.3.3 问题一结果展示

根据上述计算,得到年平均光学效率为 0.612853921;年平均余弦效率为 0.756465498;年平均阴影遮挡效率为 0.957042343；年平均截断效率为 0.944478309；年平均输出热功率（MW）为 37.47091817；单位面积镜面年平均输出热功率（kW/m^2）为 0.59648071。每个月的各类效率填入表格如表2:（完整表格参见 result1.xlsx）

表 2: 问题 1 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	0.570578831	0.719935764	0.94116205	0.944479673	0.497353605
2 月 21 日	0.598574125	0.740440353	0.956373766	0.944478935	0.564442464
3 月 21 日	0.620493838	0.761140015	0.962687042	0.944477696	0.617271503

本文将问题一以吸收塔为中心的镜场通过热力图和镜场效率图可视化如下:

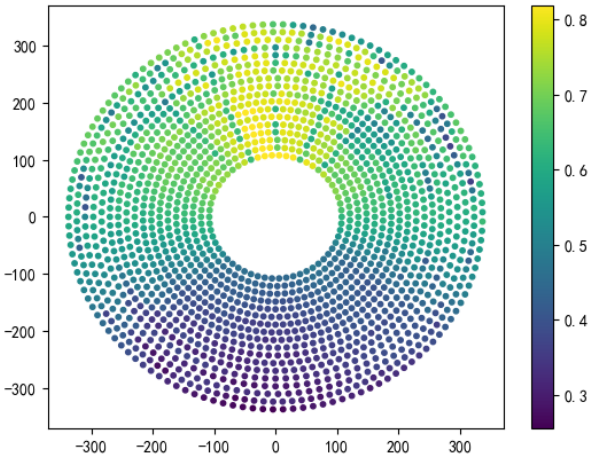

图 8: 定日镜场年平均功率热力图

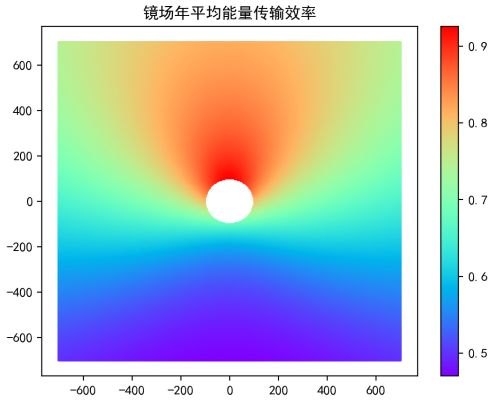

图 9: 定日镜场年平均光学效率

5.4 镜场的布局参数优化

问题二需要在有间距约束、功率约束、安装高度与镜面高度约束的条件下，重新设计镜场布局，实现单位面积平均输出热功率的最大化。鉴于布局变量之间存在相互制约的关系，考虑到第一问中余弦效率对总光学效率的影响最大，且距离吸收塔越远的定日镜大气透射率越低，因此先确定吸收塔在镜场中的位置以确保年均余弦效率与大气透射率的乘积最大，再设计每个定日镜在镜场中的位置布局使得定日镜间的阴影遮挡效率尽可能高，最后优化定日镜尺寸与安装高度变量使其在满足功率约束的条件下尽可能减小定日镜尺寸，从而提高单位镜面面积的年平均输出热功率。

5.4.1 Elimate Blocking 镜场布局

目前，镜场排布主要分为模式化布局和无模式化布局。在模式化布局中，镜场多以吸热塔为中心呈圆形、矩形、螺旋状、不规则等形状。本文首先调研几种常用镜场布局原理，接着以径向间距原理及约束条件建立本题适用的最优镜场布局。

辐射网格布局是一种应用较广泛的经典排布方式，它能有效降低阴影和遮挡效率损失，提高土地利用率和大气衰减效率。此布局方式是将定日镜沿等方位线交替放置在各恒定半径处，且随着径向距离的增加，镜场排布也呈现出由密到疏的趋势。

为定量表示定日镜与吸热塔的位置关系，本文引入了径向间距 ΔR 和方位间距 ΔA 这 2 个参数:

$$\begin{aligned}\Delta R = &(1.1442\cot\theta_{\mathrm{L}} - 1.0935+\\ &3.0648\theta_{\mathrm{L}} - 1.126\theta_{\mathrm{L}}^2)\cdot H_{\mathrm{S}}\end{aligned} \tag{36}$$

$$\begin{aligned}\Delta A &= (1.791 + 0.6396\theta_{\mathrm{L}})\cdot W_{\mathrm{S}} + \frac{0.02873}{\theta_{\mathrm{L}} - 0.04902}\\ \theta_{\mathrm{L}} &= \tan^{-1}(1/r)\end{aligned} \tag{37}$$

式(36)、(37)中，H_{S} 为定日镜高度；W_{S} 为定日镜宽度；θ_{L} 为集热器相对于定日镜的高度角；r 为以目标点高度 (即集热器孔口中心点到定日镜镜面中心点的垂直距离) 为单

位距离时定日镜到吸热塔的水平距离。

辐射性布局虽然简便，但考虑到圆形布局外环上定日镜阴影与遮挡损失较大，故基于辐射网格排布思想，结合以规避同一径向方向上相邻定日镜的遮挡损失为目标的几何作图设计方法，衍生出了无遮挡 (Eliminate Blocking) 布局、无阻塞 (No blockingdense) 布局及经验型 (DELSOL) 布局，在保证镜场中无遮挡损失的情况下尽可能密集排布定日镜。3 种布局的主要区别在于近塔区域的径向间距与方位间距布置规则不同。它们都通过方位间距重置极限因子 A_{rlim} 来限定镜场区域, 以提高土地利用率, 且同一区域中定日镜之间的方位角相等, 各镜环上定日镜数量相等, 同一镜环上定日镜之间的方位间距相等。查阅文献得知，Eliminate Blocking 布局镜场接受能量较多，因此本文采用此布局放置定日镜。

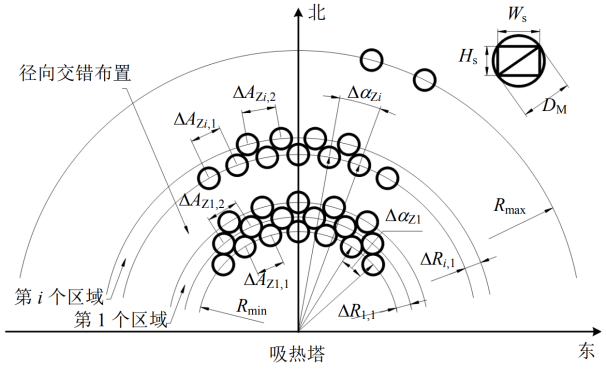

图 10: EB 布局示意图

EB 布局中镜场中各区域首环定日镜方位间距计算公式为:

$$\Delta A_{\mathrm{Z1,1}} = \Delta A_{\mathrm{Z2,1}} = \cdots = \Delta A_{\mathrm{Zi,1}} = A_{\mathrm{sf}} \cdot W_{\mathrm{S}} \tag{38}$$

式中，A_{sf} 为方位间距因子, 一般取值 1.5 或 2。各区域内除首环外，其余环定日镜方位间距通过该区域定日镜方位角得出, 计算公式为:

$$\Delta A_{Zi,j} = 2R_{i,j} \cdot \sin\left(\Delta\alpha_{Z,i}/2\right) \tag{39}$$

式中 $\Delta A_{Zi,j}$ 为镜场第 i 区域第 j 环定日镜的方位间距。第 i 个镜场区域中定日镜方位角计算公式为:

$$\Delta\alpha_{z,i} = 2\arcsin\left[\Delta A_{Zi,1}/\left(2R_{i,1}\right)\right] \tag{40}$$

各镜环定日镜数量计算公式:

$$N_{\mathrm{hel},i} = \frac{2\pi}{\Delta\alpha_{\mathrm{z},i}} \tag{41}$$

相邻镜场区域交界处定日镜径向间距恒定设置为:

$$\Delta R = D_M \tag{42}$$

$\Delta R_{1,1}$ 为镜场第 1 个区域首环与第 2 环之间的径向间距，计算公式为:

$$\Delta R_{1,1} = \sqrt{D_{\mathrm{M}}^2 - \left(\Delta A_{Z1,1}/2\right)^2} \tag{43}$$

同一区域相邻环的定日镜径向间距始终设置为 $\Delta R_{1,1}$ 。

无遮挡 (EB) 布局中无遮挡径向间距计算公式如下:

$$L_1 = \sqrt{\left(H_{\mathrm{t}} + 0.5L - Z_0\right)^2 + R_{i,j}^2} \tag{44}$$

$$\alpha_1 = \arcsin\left(R_{i,j}/L_1\right) \tag{45}$$

$$\alpha_2 = \arcsin\left(R_0/L_1\right) \tag{46}$$

$$L_3 = \tan\gamma \cdot \left(H_{\mathrm{t}} - Z_0\right) = \tan\left(\alpha_1 + \alpha_2\right)\left(H_{\mathrm{t}} - Z_0\right) \tag{47}$$

$$\Delta R_{i,j} = 2L_2 = 2\left(L_3 - R_{i,j}\right) \tag{48}$$

式(44) (48)中，H_{t} 为吸热塔光学高度; L 为吸热器高度; L_1 为吸热器中心点到前排定日镜镜面中心点连线的长度；L_2 为后排定日镜反射光线刚好不被前排定日镜遮挡时光线在前后 2 个定日镜边缘点处连线的中心点距前排定日镜镜面中心点的水平距离; L_3 为前排定日镜镜面中心点距吸热塔的水平距离; α_1 为吸热器中心点到前排定日镜镜面中心点连线与吸热塔竖直轴线的夹角; α_2 为吸热器中心点到前排定日镜镜面中心点连线与后排定日镜反射光线刚好不被前排定日镜遮挡时后排定日镜边缘点反射光线到吸热器中心点连线之间的夹角; Z_0 为定日镜中心距水平地面高度; R_0 为定日镜圆环半径; $R_{i,j}$ 为镜场第 i 个区域中第 j 个镜环的半径; γ 为后方定日镜反射光线刚好不被前排定日镜遮挡时反射光线与吸热塔轴线的夹角。

考虑到定日镜的尺寸最大为 8m，相邻定日镜底座中心之间的距离比镜面宽度多 5m，因此将图10中圆圈的半径设置为 6.5m，使用一款开源的塔式太阳能发电站布局和性能模拟软件 SolarPILOT 计算得到 Eliminate Blocking 布局下各定日镜的位置。

5.4.2 基于 EB 布局的镜场布局确立

本文在对定日镜进行排布的过程中，需考虑题目给定的几个约束条件:

1. 吸收塔与最内圈定日镜距离超过 $100m$
2. 定日镜的镜面边长在 2 m 至 8 m 之间
3. 安装高度在 2 m 至 6 m 之间

4. 安装高度必须保证镜面在绕水平转轴旋转时不会触及地面。即安装高度大于等于镜面高度的一半

5. 相邻定日镜底座中心之间的距离比镜面宽度多 5 m 以上。

根据以上约束，本文将布局确立分为以下几步：

1. 确定吸收塔位置及镜场整体区域

   定日镜场的设计与定日镜场效率紧密相关，吸热塔的位置和高度对定日镜余弦效率、遮挡和阴影效率以及大气透射率有一定的关系。以上三种辐射形布局均为标准圆形布局，吸收塔位于圆形定日镜场中心，这在镜场可以均匀接收到每个方向的太阳光的情况下符合对称性及效率最大化。然而，根据题目区域的位置信息，圆形定日镜场位于东经 98.5° , 北纬 39.4°, 海拔 3000 m, 半径 350 m 处，故太阳方位角 γs 并不在 $0-2\pi$ 间均匀分布，故针对北半球的定日镜场，吸热塔的位置处于定日镜场中部偏南的位置，有利于定日镜场效率的提升。

   从效率公式角度定量角度分析：效率 η 的表达式中，除去反射效率、大气衰减效率，遮挡、截断效率只与镜场内部布局相关，与镜场位置无关。而吸热塔的位置与定日镜余弦效率紧密相关。位于北半球，尤其是在北回归线以北的塔式光热电站，太阳大部分时间或全部时间在定日镜场南侧，吸热塔以北的定日镜场余弦效率远高于吸热塔南边的定日镜场。因此，吸热塔位置处于定日镜场南部，即北定日镜场定日镜数量高于南定日镜场时，有助于北半球塔式光热电站余弦效率的提升；吸热塔的位置与遮挡和阴影效率也有一定关系，但遮挡和阴影效率更多地受镜场布置影响。与余弦效率相反，吸热塔南部的定日镜遮挡和阴影效率高于吸热塔北镜场。综合考虑余弦效率以及遮挡和阴影效率的影响，北半球塔式光热电站北镜场的定日镜场效率仍高于南镜场。因此对于北半球的塔式光热电站，吸热塔的位置处于定日镜场中部偏南的位置，有利于定日镜场效率的提升。因此，为达到输出功率最大，我们不使用塔位于中心的布局方式，而是采取整个镜场区域向北偏移的布局，这样才能最佳地利用太阳光线。本部分以余弦效率最大化为目标，计算出最优的镜场位置。

   利用位置约束条件，在镜场坐标系下以区域总余弦效率为目标函数，最大化确定吸收塔的坐标。镜场坐标系以圆心区域为圆心，镜场半径固定为 $350m$，吸收塔定在正北方向（y 轴方向）上。结合塔内 $100m$ 为半径的圆形区域内不设置定日镜的约束，由几何关系即吸收塔位置在 y 轴的圆心 O 至 $-250m$ 范围内。本问利用等步长搜索算法，以 $0.1m$ 为精度，遍历吸收塔位置，求得余弦效率最大时，最终得到吸收塔位置在 $250m$ 处，即正好位于向北偏移的最大处。

2. 确定定日镜内部布局

   确定了定日镜场的排布区域为镜场坐标系后，接下来就要确立每个定日镜的具体排布方式。根据调研，在镜场半径为 $1000m$ 的范围内，No blocking-dense 布局下

镜场虽然土地利用率最高，EB 布局次之，但 EB 布局下定日镜数量和镜场接收能量最多，DELSOL 布局的各项性能特性均最低。考虑到本问中年均光学效率最大的优化目标，我们因此对内部镜场排布采取 EB 布局。

根据相邻定日镜底座中心之间的距离比镜面宽度多 5 m 以上的约束，设定 EB 布局的参数 A_{sf} 为 2，$W_S = 6m$，计算得共需要 1620 面定日镜，得到每个镜子的位置坐标（具体参见附录文件）。

3. 确定定日镜高度等尺寸等其他参数

问题二还需要确定定日镜的尺寸及安装高度，本文采用粒子群启发式优化算法，设定约束：定日镜的镜面边长在 2 m 至 8 m 之间；安装高度在 2 m 至 6 m 之间；高度大于等于镜面长边边长的一半。

粒子群优化算法 (Particle Swarm Optimization, PSO) 源于对鸟群觅食行为的研究，鸟群通过集体的信息共享使群体找到最优的目的地。在搜索的过程中每只鸟都会根据自己记忆中食物量最多的位置和当前鸟群记录的食物量最多的位置调整自己接下来搜索的方向。鸟群经过一段时间的搜索后就可以找到森林中哪个位置的食物量最多（全局最优解）。粒子群算法的 6 个重要参数如下：

假设在 D 维搜索空间中，有 N 个粒子，每个粒子代表一个解，则:

(a) 第 i 个粒子的位置为:

$X_{id} = (x_{i1}, x_{i2}, \ldots, x_{iD})$

(b) 第 i 个粒子的速度 (粒子移动的距离和方向) 为:

$V_{id} = (v_{i1}, v_{i2}, \ldots, v_{iD})$

(c) 第 i 个粒子搜索到的最优位置 (个体最优解) 为:

$P_{id,pbest} = (p_{i1}, p_{i2}, \ldots, p_{iD})$

(d) 群体搜索到的最优位置 (群体最优解) 为:

$P_{d,gbest} = (p_{1,gbest}, p_{2,gbest}, \ldots, p_{D,gbest})$

(e) 第 i 个粒子搜索到的最优位置的适应值 (优化目标函数的值) 为: f_p 一个体历史最优适应值

(f) 群体搜索到的最优位置的适应值为: f_g

粒子群的流程图如图11 ：

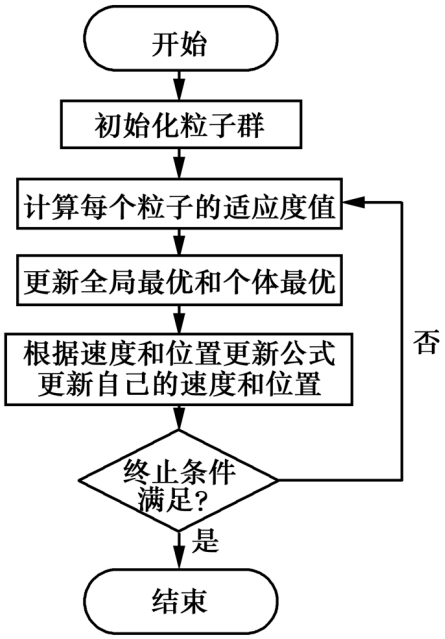

图 11: 粒子群算法流程图

将 1620 面定日镜的位置参数传入进行计算，得出最优的尺寸、高度参数为高度 $4m$, 尺寸为 $8m \times 8m$。

5.4.3 问题二结果展示

根据上述计算，得到年平均光学效率为 0.61511515；年平均余弦效率为 0.862726838；年平均阴影遮挡效率为 0.87966278；年平均截断效率为 0.94445543；年平均输出热功率（MW）为 61.84442211；单位面积镜面年平均输出热功率（kW/m^2）为 1.060432478。其余部分结果如下（完整参见 result2.xlsx）

表 3: 问题 2 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	0.611892206	0.889960991	0.850013717	0.944455591	0.94981692
2 月 21 日	0.621699256	0.882922942	0.870277778	0.944455593	0.043658931
3 月 21 日	0.624364341	0.869378866	0.88765775	0.944455653	1.104842554

表 4: 问题 2 设计参数表

吸收塔位置坐标	定日镜尺寸（宽 (m)× 高 (m))	定日镜安装高度 (m)	定日镜总面数	定日镜总面积 (m^2)
$(0, -250, 0)$	8×8	4	1620	103680

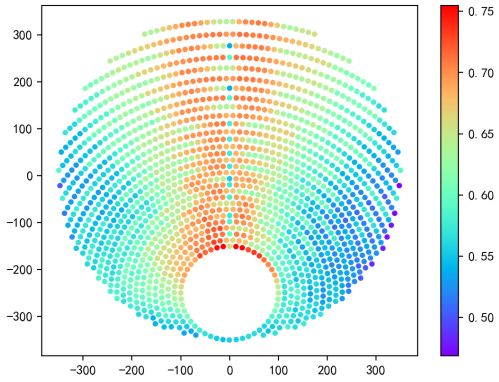

图 12: 定日镜场年平均光学效率热力图

5.5 基于高度尺寸的定日镜场布局调整

第三问解除了定日镜高度尺寸一致的限制，需要我们，需要设计单位面积下年平均效率最大化时的镜面尺寸及定日镜中心高度。因此我们采取在第二问的基础上，基于效率模型中的余弦效率及阴影效率，用几种调整方案进行尝试，选取最优的调整方式。

5.5.1 基于高度的调整

为使整体光学效率最高，定日镜的高度设计与遮挡效率、余弦效率密切相关。考虑到位于后排的定日镜会与前排定日镜产生遮挡，升高后排定日镜可以提升一定的遮挡效率；同时，定日镜高度越高，与吸收塔的中心余弦角度越大，余弦效率也越低。因此，寻找两者在镜场效率中的平衡是定日镜高度调整的关键。

我们以不同的参数做了以下四组试验模拟：

1. 均匀设定前后排高度差为 0.1m
2. 前后排高度差 0.2m
3. 前后排高度差成等比数列，公比为 1.1，初始值为 0.05
4. 前后排高度差成等比数列，公比为 1.2，初始值为 0.01

模拟结果显示，第三种高度调整策略最优。

5.5.2 基于尺寸的调整

为使整体光学效率最高，定日镜的尺寸设计与遮挡效率、余弦效率也密切相关。由于后排定日镜间隔较宽，本问可以采取“近小远大”的定日镜尺寸方案，设置后排定日镜较前排定日镜边长扩大特定倍数。根据计算，得到新的尺寸设计如表5:

表 5: 问题 3 设计参数表

吸收塔位置坐标	定日镜尺寸（宽 (m)× 高 (m))	定日镜安装高度 (m)	定日镜总面数	定日镜总面积 (m^2)
$(0, -250, 0)$			1850	132480

六 模型的评价、改进与推广

6.1 模型的优点

1. 充分利用光发反射定律进行定日镜反射太阳光建模，符合题目要求。
2. 本文模型从第一问固定吸热塔及定日镜参数，到后两问逐渐放宽位置、尺寸、高度限制，兼具个性化与一定的普适性。
3. 本文模型结合经纬度信息，依据北半球经纬度信息进行定日镜场针对设计。

6.2 模型的缺点

1. 计算定日镜间遮挡损失时采用蒙特卡洛光线追踪法计算量大，耗时较长
2. 问题二中优化算法耗时较久，没有很好地进行搜索剪枝
3. 当不同定日镜尺寸不同时计算遮挡效率较为复杂

6.3 模型的改进

1. 计算定日镜间遮挡损失时可以先获取阴影可能影响到的定日镜编号，再采用蒙特卡洛光线追踪法对指定坐标进行计算。

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] 孙浩, 高博, 刘建兴. 塔式太阳能电站定日镜场布局研究 [J]. 发电技术,2021,42(06):690-698.

[3] 房淼森, 逯静, 姜奕雯. 定日镜能量传输效率建模及镜场排布设计 [J]. 太阳能学报,2021,42(01):112-116.DOI:10.19912/j.0254-0096.tynxb.2018-0755.

[4] Toufik A,Adel B,Mawloud G, et al. Optimisation of heliostat field layout for solar power tower systems using iterative artificial bee colony algorithm: a review and case study[J]. International Journal of Ambient Energy,2021,42(1).

[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,2022.DOI:10.27205/d.cnki.gltec.2022.001089.

[6] 文龙, 肖斌, 周治等. 基于 HFLCAL 计算模型的定日镜成像仿真分析 [J]. 太阳能,2017(07):46-49.DOI:10.19911/j.1003-0417.2017.07.008.

[7] 张国勋, 饶孝枢. 塔式太阳能聚光系统太阳影象方程 [J]. 太阳能学报,1982(02):172-178.DOI:10.19912/j.0254-0096.1982.02.008.

附录

附录 1：问题一代码

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import sympy
from sympy import Symbol, solve
from datetime import datetime
from numpy import pi, sin, cos, tan, arcsin, arccos, arctan, sqrt
from sklearn.preprocessing import MinMaxScaler
from skspatial.objects import Circle, Line
from numba import njit
import random
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
PHI = 39.4 / 180 * pi
ST = [9.0, 10.5, 12.0, 13.5, 15.0]
D = [(datetime(2023,i,21)-datetime(2023,3,21)).days for i in range(1, 13)]
h_t = 80 # 集热器中心高度
h_max = 84 # 塔顶端高度
h_m = 4 # 定日镜中心高度
m_length, m_width = 6, 6
m_x_bound, m_y_bound = [-3,3], [-3,3] # 定日镜坐标边界
t_x_bound, t_z_bound = [-3.5, 3.5], [0, h_max]
def cal_mod_length(x):
    return sqrt(x[0]**2+x[1]**2)
## 计算太阳高度角，方位角
def cal_omega(st):
    # 计算太阳时角
    return pi / 12 * (st - 12)
def cal_delta(d):
    # 计算太阳赤纬角
    sin_delta = sin(2*pi*d/365) * sin(2*pi*23.45/360)
    return arcsin(sin_delta)
def scale_in_min_max(num, min=-1, max=1):
    # 将结果限制在范围内
    if (num < min): return min
```

```
    elif (num > max): return max
    else: return num
def cal_alpha_and_gamma_s(d, st):
    # 计算太阳高度角和方位角
    omega = cal_omega(st)
    delta = cal_delta(d)
    sin_alpha_s = cos(delta)*cos(PHI)*cos(omega) + sin(delta)*sin(PHI)
    sin_alpha_s = scale_in_min_max(sin_alpha_s)
    alpha_s = arcsin(sin_alpha_s)
    cos_gamma_s = (sin(delta) - sin(alpha_s) * sin(PHI)) / ((cos(alpha_s)
        * cos(PHI)))
    cos_gamma_s = scale_in_min_max(cos_gamma_s)
    gamma_s = arccos(cos_gamma_s)
    if (st > 12):
        gamma_s = 2*pi - gamma_s
    return alpha_s, gamma_s
alpha_mat = np.zeros((12, 5))
gamma_mat = np.zeros((12, 5))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        alpha_mat[i,j] = alpha
        gamma_mat[i,j] = gamma
## 计算定日镜俯仰角，方位角
df = pd.read_excel("附件.xlsx")
mirrors_xy = df.values
mirrors_xyz = np.insert(mirrors_xy, 2, values=h_m, axis=1)
def cal_S_r(mirror_xyz):
    # 计算反射光线单位向量
    O = np.array([0 ,0, h_t])
    O_A = np.array(mirror_xyz)
    v = O - O_A
    v = v / np.linalg.norm(v)
    return v
def cal_S_i(alpha, gamma):
    # 计算入射光线单位向量
    # alpha, gamma是弧度制
    x = -cos(alpha) * cos(gamma-pi/2)
    y = cos(alpha) * sin(gamma-pi/2)
    z = -sin(alpha)
```

```
    return np.array([x, y, z])
# 计算定日镜法向量
S_i_mat = np.zeros((12, 5, 3))
S_r_mat = np.zeros((12, 5, *mirrors_xyz.shape))
S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        S_i_mat[i,j] = S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            S_r_mat[i,j,k] = S_r
            S_n_mat[i,j,k] = S_n
def cal_theta(S_n):
    # 计算定日镜俯仰角，方位角
    # 俯仰角：[0, pi/2)
    # 方位角：[0, 2*pi)
    x, y, z = S_n
    theta_z = arctan(sqrt(x**2 + y**2) / z)
    if theta_z < 0:
        theta_z = theta_z + pi
    theta_s = arcsin(x / sqrt(x**2 + y**2))
    if x > 0 and y < 0:
        theta_s = pi - theta_s
    elif x <= 0 and y >= 0:
        theta_s = 2*pi + theta_s
    elif x <= 0 and y < 0:
        theta_s = pi - theta_s
    return theta_z, theta_s
# 计算定日镜俯仰角，方位角
theta_mat = np.zeros((12, 5, mirrors_xyz.shape[0], 2))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
```

```
            theta_z, theta_s = cal_theta(S_n)
            theta_mat[i,j,k] = [theta_z, theta_s]
## 计算简单效率
# 计算余弦效率
ita_cos_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        neg_S_i = -S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            ita_cos = S_n_mat[i,j,k] @ neg_S_i
            ita_cos_mat[i,j,k] = ita_cos

print("年平均余弦效率:", ita_cos_mat.mean())
for i, d in enumerate(D):
    print(f"{i+1:2}月平均余弦效率:", ita_cos_mat[i].mean())
# 计算大气透射率
def cal_ita_at(mirror_xyz):
    d = np.linalg.norm(mirror_xyz - np.array([0, 0, h_t]))
    return 0.99321 - 0.0001176*d + 1.97*10**(-8)*d**2
ita_at_mat = np.zeros(mirrors_xyz.shape[0])
for i, mirror_xyz in enumerate(mirrors_xyz):
    ita_at_mat[i] = cal_ita_at(mirror_xyz)
## 计算阴影遮挡
# 计算旋转矩阵
def cal_rot_mat(theta_z, theta_s):
    # 根据俯仰角和方位角，计算旋转矩阵
    beta, gamma = theta_z, theta_s
    sb, cb, sg, cg = sin(beta), cos(beta), sin(gamma), cos(gamma)
    return np.array([[cb*cg, -sg, sb*cg],
                     [sg*cb, cg, sb*sg],
                     [-sb,  0,   cb]])
rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            theta_z, theta_s = theta_mat[i,j,k]
            rot_mats[i,j,k] = cal_rot_mat(theta_z, theta_s)
def cal_distance(m1xy, m2xy):
```

```
    # 计算二维距离
    dis = np.linalg.norm(m1xy - m2xy)
    return dis
@njit
def in_bound(x, bound):
    # 判断点是否出了边界
    if (x < bound[0] or x > bound[1]):
        return False
    return True
@njit
def a_blocked_by_b(xy, V_0, day_id, st_id, a_id, b_id, shoot_in):
    m_x_bound, m_y_bound = [-3,3], [-3,3] # 定日镜坐标边界
    # shoot_in参数是指光为入射光还是为反射光，可取bool值
    # 后缀为0，a，b指地面坐标系，a镜坐标系，b镜坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    O_b = mirrors_xyz[b_id]
    if (O_a[:2] - O_b[:2]) @ V_0[:2] <= 0 and shoot_in:
        return False
    elif (O_a[:2] - O_b[:2]) @ V_0[:2] > 0 and not shoot_in:
        return False
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    b_rot_mat = rot_mats[day_id, st_id, b_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    h1_b = b_rot_mat.T @ (h1_0 - O_b)
    V_b = b_rot_mat.T @ (V_0)
    x2_b = h1_b[0] - V_b[0]*h1_b[2]/ V_b[2]
    y2_b = h1_b[1] - V_b[1]*h1_b[2]/ V_b[2]
    h2_b = np.array([x2_b, y2_b, 0])
    if in_bound(x2_b, m_x_bound) and in_bound(y2_b, m_y_bound):
        return True
    return False
@njit
def a_blocked_by_tower(xy, day_id, st_id, a_id):
    t_x_bound, t_z_bound = [-3.5, 3.5], [0, 84]
    # 后缀为0，a，t指地面坐标系，a镜坐标系，塔坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    if (O_a[1] <= 0):
        return False
```

```
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    alpha, gamma = alpha_mat[day_id,st_id], gamma_mat[day_id,st_id]
    t_xy_rot_mat = np.array([[-cos(gamma), -sin(gamma)],
                             [sin(gamma), -cos(gamma)]])
    xy_t = t_xy_rot_mat @ h1_0[0:2]
    h1_t = np.array([*xy_t, h1_0[2]])
    if h1_t[1] >= 0:
        z_delta = tan(alpha)*sqrt(xy[0]**2 + xy[1]**2)
        z2_t = z_delta + h1_t[2]
        h2_t = np.array([h1_t[0], 0, z2_t]) #
            h2_t为射向h1点的光与塔平面的交点
        if t_x_bound[0] <=h2_t[0] <= t_x_bound[1] and t_z_bound[0] <=
            h2_t[2] <=t_z_bound[1]:
            return True
    return False
# 蒙特卡洛法，将定日镜等分成36个1m^2区域，每一个区域取中心点
x_mont = [-2.5, -1.5, -0.5, 0.5, 1.5, 2.5]
y_mont = [-2.5, -1.5, -0.5, 0.5, 1.5, 2.5]
xy_mont = np.array([[x, y] for x in x_mont for y in y_mont])
# 取每面定日镜的最近邻8面镜子，判断是否发生阴影或遮挡
MAX_NEAR_NUM = 8
near_mat = np.zeros((mirrors_xyz.shape[0], MAX_NEAR_NUM), dtype=np.int64)
for a_id, m1xy in enumerate(mirrors_xy):
    # 计算和其他镜子的距离
    distances = {}
    for b_id, m2xy in enumerate(mirrors_xy):
        if a_id == b_id:
            continue
        distances[b_id] = cal_distance(m1xy, m2xy)
    # 排序取出最近邻的8面镜子
    ls = []
    for new_b_id, _ in sorted(distances.items(), key = lambda kv:(kv[1],
        kv[0]))[:MAX_NEAR_NUM]:
        ls.append(new_b_id)
    near_mat[a_id] = ls
# 计算塔平面产生的阴影
blocked_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
    len(y_mont)), dtype=bool)
for i, d in enumerate(D):
```

```
    for j, st in enumerate(ST):
        for a_id, _ in enumerate(mirrors_xyz):
            for x_id, x in enumerate(x_mont):
                for y_id, y in enumerate(y_mont):
                    if a_blocked_by_tower([x,y], i, j, a_id):
                        # 被塔挡住，产生阴影
                        blocked_mat[i, j, a_id, x_id, y_id] = True
# 计算定光镜相互阴影
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        S_i = S_i_mat[i, j]
        for a_id, near_pairs in enumerate(near_mat):
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if a_blocked_by_b([x,y], S_i, i, j, a_id, b_id,
                            shoot_in=True):
                            # 把入射光挡住，产生阴影
                            blocked_mat[i, j, a_id, x_id, y_id] = True
# 计算遮挡
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for a_id, near_pairs in enumerate(near_mat):
            S_r = S_r_mat[i,j,a_id]
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if (not blocked_mat[i, j, a_id, x_id, y_id]) and \
                        a_blocked_by_b([x,y], S_r, i, j, a_id, b_id,
                            shoot_in=False):
                            # 没有阴影，并且反射光被挡住
                            blocked_mat[i, j, a_id, x_id, y_id] = True
ita_sb_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_sb_mat[i,j,k] = 1 - blocked_mat[i,j,k].sum() /
                (len(x_mont)*len(y_mont))
print("年平均阴影遮挡效率", ita_sb_mat.mean())
for i, _ in enumerate(D):
```

```
    print(f"{i+1:2}月平均阴影遮挡效率:", ita_sb_mat[i].mean())
## 计算截断效率
MAX_ANGLE = 4.65 * 10**(-3)
angle_divide_num = 3
circle_divide_num = 6
tao_arr = np.linspace(0, 2*pi, circle_divide_num, endpoint=False)
sigma_arr = np.linspace(MAX_ANGLE, 0, angle_divide_num, endpoint=False)
light_mat = np.zeros((len(tao_arr) * len(sigma_arr), 3))
for i, tao in enumerate(tao_arr):
    for j, sigma in enumerate(sigma_arr):
        light_mat[len(sigma_arr)*i+j] = [sin(sigma)*cos(tao),
            sin(sigma)*sin(tao), cos(sigma)]
light_rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            S_r = S_r_mat[i,j,k]
            if (S_r[1] == 0):
                theta_x = pi / 2
            else:
                theta_x = arctan(-S_r[0] / S_r[1])
            x_s = np.array((cos(theta_x), sin(theta_x), 0))
            y_s = np.cross(S_r, x_s)
            # y_s = y_s / np.linalg.norm(y_s)
            light_rot_mats[i,j,k] = np.array((x_s, y_s, S_r)).T
R = 3.5
r_bound = [-3.5, 3.5]
machine_bound = [76, 84]
circle = Circle([0, 0], R)
def light_meet_machine(xy, light_0, day_id, st_id, m_id):
    m, n, l = light_0
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = rot_mat @ h1_a + O_a
    x1, y1, z1 = h1_0
    if (abs(m*y1 - n*x1) / sqrt(n**2+m**2) < R):
        line = Line([x1, y1], [x1+m, y1+n])
        point_a, point_b = circle.intersect_line(line)
        if abs(x1) < abs(y1):
```

```
            z = min(l*(point_a[1]-y1)/n+z1, l*(point_b[1]-y1)/n+z1)
        else:
            z = min(l*(point_a[0]-x1)/m+z1, l*(point_b[0]-x1)/m+z1)
        if in_bound(z, machine_bound):
            return True
    return False
trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
    len(y_mont)))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            for l_id, light in enumerate(light_mat):
                light_0 = light_rot_mats[i,j,k] @ light
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        trunc_mat[i,j,k,x_id,y_id] = angle_divide_num *
                            circle_divide_num
                        if blocked_mat[i,j,k,x_id,y_id]:
                            trunc_mat[i,j,k,x_id,y_id] = 0
                        if not blocked_mat[i,j,k,x_id,y_id] and \
                        not light_meet_machine([x,y], light_0, i, j, k):
                            trunc_mat[i,j,k,x_id,y_id] -= 1
ita_trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            cnt = 0
            for x_id, x in enumerate(x_mont):
                for y_id, y in enumerate(y_mont):
                    if not blocked_mat[i,j,k,x_id,y_id]:
                        cnt += 1
            if cnt == 0:
                ita_trunc_mat[i,j,k] = 0.94444444 # 对全阴影镜赋均值
            else:
                ita_trunc_mat[i,j,k] = trunc_mat[i,j,k].sum() / cnt /
                    (angle_divide_num * circle_divide_num)
print("年平均截断效率:", ita_trunc_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均截断效率:", ita_trunc_mat[i].mean())
## 计算DNI
```

```
G0 = 1.366
H = 3
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_mat = np.zeros((12, 5))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(D[i], ST[j])
        DNI_mat[i,j] = G0*(a + b*pow(np.e,-c/sin(alpha)))
## 计算光学效率
ita_ref = 0.92
ita_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_cos = ita_cos_mat[i,j,k]
            ita_at = ita_at_mat[i]
            ita_sb = ita_sb_mat[i,j,k]
            ita_trunc = ita_trunc_mat[i,j,k]
            ita_mat[i,j,k] = ita_cos * ita_sb * ita_at * ita_ref * ita_trunc
print("年平均光学效率:", ita_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均光学效率:", ita_mat[i].mean())
## 计算热功率
E_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            E_mat[i,j,k] = DNI_mat[i,j] * m_length * m_width * ita_mat[i,j,k]
cm = matplotlib.colormaps['viridis']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1],
    c=ita_mat[i].mean(axis=0), s=10, cmap=cm)
plt.colorbar(sc)
plt.show()
print("年单位面积镜面平均输出热功率:", E_mat.mean() / 36)
for i, _ in enumerate(D):
    print(f"{i+1:2}月单位面积镜面平均输出热功率:", E_mat[i].mean() / 36)
print("年平均输出热功率:", (E_mat.sum() / 5 / 12))
for i, _ in enumerate(D):
```

```
    print(f"{i+1:2}月平均输出热功率(kW):", E_mat[i].sum() / 5)
```

附录 2: 问题 2 代码 A2.ipynb

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from datetime import datetime
from numpy import pi, sin, cos, tan, arcsin, arccos, arctan, sqrt
from sko.PSO import PSO

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
PHI = 39.4 / 180 * pi
ST = [9.0, 10.5, 12.0, 13.5, 15.0]
D = [(datetime(2023,i,21)-datetime(2023,3,21)).days for i in range(1, 13)]
h_t = 80 # 集热器中心高度
h_max = 84 # 塔顶端高度
h_m = 4 # 定日镜中心高度
m_length, m_width = 6, 6
m_x_bound, m_y_bound = [-3,3], [-3,3] # 定日镜坐标边界
t_x_bound, t_z_bound = [-3.5, 3.5], [0, h_max]
## 常用函数
def cal_distance(m1xy, m2xy):
    # 计算二维距离
    dis = np.linalg.norm(m1xy - m2xy)
    return dis
def in_bound(x, bound):
    # 判断点是否出了边界
    if (x < bound[0] or x > bound[1]):
        return False
    return True
## 计算太阳高度角，方位角
def cal_omega(st):
    # 计算太阳时角
    return pi / 12 * (st - 12)

def cal_delta(d):
    # 计算太阳赤纬角
```

```
    sin_delta = sin(2*pi*d/365) * sin(2*pi*23.45/360)
    return arcsin(sin_delta)

def scale_in_min_max(num, min=-1, max=1):
    # 将结果限制在范围内
    if (num < min): return min
    elif (num > max): return max
    else: return num

def cal_alpha_and_gamma_s(d, st):
    # 计算太阳高度角和方位角
    omega = cal_omega(st)
    delta = cal_delta(d)
    sin_alpha_s = cos(delta)*cos(PHI)*cos(omega) + sin(delta)*sin(PHI)
    sin_alpha_s = scale_in_min_max(sin_alpha_s)
    alpha_s = arcsin(sin_alpha_s)
    cos_gamma_s = (sin(delta) - sin(alpha_s) * sin(PHI)) / ((cos(alpha_s)
        * cos(PHI)))
    cos_gamma_s = scale_in_min_max(cos_gamma_s)
    gamma_s = arccos(cos_gamma_s)
    if (st > 12):
        gamma_s = 2*pi - gamma_s
    return alpha_s, gamma_s
alpha_mat = np.zeros((12, 5))
gamma_mat = np.zeros((12, 5))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        alpha_mat[i,j] = alpha
        gamma_mat[i,j] = gamma
## 计算定日镜俯仰角，方位角
def cal_S_r(mirror_xyz):
    # 计算反射光线单位向量
    O = np.array([0 ,0, h_t])
    O_A = np.array(mirror_xyz)
    v = O - O_A
    v = v / np.linalg.norm(v)
    return v

def cal_S_i(alpha, gamma):
```

```
    # 计算入射光线单位向量
    # alpha, gamma是弧度制
    x = -cos(alpha) * cos(gamma-pi/2)
    y = cos(alpha) * sin(gamma-pi/2)
    z = -sin(alpha)
    return np.array([x, y, z])
# 计算定日镜法向量
S_i_mat = np.zeros((12, 5, 3))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        S_i_mat[i,j] = S_i
## 计算简单效率
# 计算大气透射率
def cal_ita_at(mirror_xyz):
    d = np.linalg.norm(mirror_xyz - np.array([0, 0, h_t]))
    return 0.99321 - 0.0001176*d + 1.97*10**(-8)*d**2
## 计算DNI
G0 = 1.366
H = 3
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_mat = np.zeros((12, 5))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(D[i], ST[j])
        DNI_mat[i,j] = G0*(a + b*pow(np.e,-c/sin(alpha)))
def cal_mod_length(x):
    return sqrt(x[0]**2+x[1]**2)
slice_num = 350
max_bound = 700
x = np.linspace(-max_bound, max_bound, slice_num)
y = np.linspace(-max_bound, max_bound, slice_num)
z = np.ones((slice_num)) * 5
xyz = []
for i in range(slice_num ):
    for j in range(slice_num):
        if cal_mod_length((x[i],y[j])) > 100:
```

```
            xyz.append([x[i], y[j], z[i]])
mirrors_xyz = np.array(xyz)
mirrors_xy = mirrors_xyz[:,:2]
m_num = mirrors_xyz.shape[0]
S_r_mat = np.zeros((12, 5, *mirrors_xyz.shape))
S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        S_i = S_i_mat[i,j]
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            S_r_mat[i,j,k] = S_r
            S_n_mat[i,j,k] = S_n
# 计算余弦效率
ita_cos_mat = np.zeros((12, 5, m_num))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = alpha_mat[i,j], gamma_mat[i,j]
        S_i = S_i_mat[i,j]
        neg_S_i = -S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            ita_cos = S_n_mat[i,j,k] @ neg_S_i
            ita_cos_mat[i,j,k] = ita_cos
# 计算大气透射率
ita_at_mat = np.zeros(m_num)
for k, mirror_xyz in enumerate(mirrors_xyz):
    ita_at_mat[k] = cal_ita_at(mirror_xyz)
ita_mat = np.zeros((12, 5, m_num))
E_mat = np.zeros((m_num))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_mat[i,j,k] = ita_at_mat[k] * ita_cos_mat[i,j,k]
            E_mat[k] += DNI_mat[i,j] * ita_mat[i,j,k]
E_mat /= 60
for k, mirror_xyz in enumerate(mirrors_xyz):
    if cal_mod_length(mirror_xyz[:2]) < 100:
        E_mat[k] = 0
title = "镜场年平均能量传输效率"
```

```
cm = matplotlib.colormaps['rainbow']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1],
    c=ita_mat.mean(axis=0).mean(axis=0), s=1, cmap=cm)
plt.title(title)
plt.colorbar(sc)
plt.savefig(title + ".png", dpi=200)
plt.show()
ita_mat = ita_mat.mean(axis=0).mean(axis=0)
R_field = 350
mont_bound = 250
mont_num = 50
x_mont = np.linspace(-mont_bound, mont_bound, mont_num)
y_mont = np.linspace(-mont_bound, mont_bound, mont_num)
z_mont = np.ones((mont_num)) * 5
xyz_mont = np.zeros((mont_num**2 , 3))
for i in range(mont_num):
    for j in range(mont_num):
        xyz_mont[i*mont_num +j] = [x_mont[i], y_mont[j], z_mont[i]]

sm_mat = np.zeros((xyz_mont.shape[0]))
for i, (x, y, z) in enumerate(xyz_mont):
    sm = 0
    if cal_mod_length((x,y)) <= 250:
        for k, mirror_xyz in enumerate(mirrors_xyz):
            if cal_distance(mirror_xyz[:2], [x,y]) < R_field:
                sm += ita_mat[k]
    sm_mat[i] = sm
def find_max(xy_mont):
    R_field = 350
    x, y = xy_mont
    sm = 0
    for k, mirror_xyz in enumerate(mirrors_xyz):
        if cal_distance(mirror_xyz[:2], [x,y]) < R_field:
            sm += ita_mat[k]
    return -sm
constraint_ueq = (
    lambda x: cal_mod_length((x[0],x[1])) - 250,
)
pso = PSO(func=find_max, n_dim=2, pop=40, max_iter=20, lb=[-mont_bound,
    -mont_bound], ub=[mont_bound, mont_bound],
```

```
    constraint_ueq=constraint_ueq)
pso.run()
print('best_x is ', pso.gbest_x, 'best_y is', pso.gbest_y)
plt.plot(pso.gbest_y_hist)
plt.show()
```

附录 2: 问题 2 代码——A2.ipynb

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from datetime import datetime
from numpy import pi, sin, cos, tan, arcsin, arccos, arctan, sqrt
from skspatial.objects import Circle, Line
from numba import njit

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
## 常用函数
def cal_distance(m1xy, m2xy):
    # 计算二维距离
    dis = np.linalg.norm(m1xy - m2xy)
    return dis
def in_bound(x, bound):
    # 判断点是否出了边界
    if (x < bound[0] or x > bound[1]):
        return False
    return True
def cal_mod_length(x):
    return sqrt(x[0]**2+x[1]**2)
PHI = 39.4 / 180 * pi
ST = [9.0, 10.5, 12.0, 13.5, 15.0]
D = [(datetime(2023,i,21)-datetime(2023,3,21)).days for i in range(1, 13)]
h_t = 80 # 集热器中心高度
h_max = 84 # 塔顶端高度
h_m = 4 # 定日镜中心高度
m_length, m_width = 8, 8
m_x_bound, m_y_bound = [-m_width, m_width], [-m_length, m_length] #
    定日镜坐标边界
```

```
t_x_bound, t_z_bound = [-3.5, 3.5], [0, h_max]
mirrors_xy = pd.read_excel("EB布局13.xlsx").values
center = (0, 250)
m_xy = mirrors_xy
m_in_xy = []
for xy in m_xy:
    if cal_distance(xy, center) < 350:
        m_in_xy.append(xy)
m_in_xy = np.array(m_in_xy)
mirrors_xy = m_in_xy
mirrors_xyz = np.insert(mirrors_xy, 2, values=h_m, axis=1)
pd.DataFrame(mirrors_xyz-(0,250,0)).to_csv("result2xy.csv")
def func(x):
    ml = x[0]
    mw = x[1]
    mh = x[2]
    mirrors_xy = (gen_mirrors_xy(mw))
    m_num = mirrors_xy.shape[0]
    mirrors_xyz = np.zeros((m_num,3))
    for i in range(m_num):
        mirrors_xyz[i] = [*mirrors_xy[i], mh]
    # 计算定日镜法向量
    S_i_mat = np.zeros((12, 5, 3))
    S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
    for i, d in enumerate(D):
        for j, st in enumerate(ST):
            alpha, gamma = cal_alpha_and_gamma_s(d, st)
            S_i = cal_S_i(alpha, gamma)
            S_i_mat[i,j] = S_i
            for k, mirror_xyz in enumerate(mirrors_xyz):
                S_r = cal_S_r(mirror_xyz)
                S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
                S_n_mat[i,j,k] = S_n
    # 计算余弦效率
    ita_cos_mat = np.zeros((12, 5, m_num))
    for i, d in enumerate(D):
        for j, st in enumerate(ST):
            for k, mirror_xyz in enumerate(mirrors_xyz):
                ita_cos = -S_n_mat[i,j,k] @ S_i_mat[i,j]
                ita_cos_mat[i,j,k] = ita_cos
```

```
    # 计算大气透射率
    ita_at_mat = np.zeros(m_num)
    for k, mirror_xyz in enumerate(mirrors_xyz):
        ita_at_mat[k] = cal_ita_at(mirror_xyz)
    ita_mat = np.zeros((12, 5, m_num))
    E_mat = np.zeros((m_num))
    for i, _ in enumerate(D):
        for j, _ in enumerate(ST):
            for k, _ in enumerate(mirrors_xyz):
                ita_mat[i,j,k] = ita_at_mat[k] * ita_cos_mat[i,j,k]
                E_mat[k] += DNI_mat[i,j] * ita_mat[i,j,k] * mw * ml
    E_mat /= 60
    return -E_mat.sum()
## 计算太阳高度角，方位角
def cal_omega(st):
    # 计算太阳时角
    return pi / 12 * (st - 12)

def cal_delta(d):
    # 计算太阳赤纬角
    sin_delta = sin(2*pi*d/365) * sin(2*pi*23.45/360)
    return arcsin(sin_delta)

def scale_in_min_max(num, min=-1, max=1):
    # 将结果限制在范围内
    if (num < min): return min
    elif (num > max): return max
    else: return num
def cal_alpha_and_gamma_s(d, st):
    # 计算太阳高度角和方位角
    omega = cal_omega(st)
    delta = cal_delta(d)
    sin_alpha_s = cos(delta)*cos(PHI)*cos(omega) + sin(delta)*sin(PHI)
    sin_alpha_s = scale_in_min_max(sin_alpha_s)
    alpha_s = arcsin(sin_alpha_s)
    cos_gamma_s = (sin(delta) - sin(alpha_s) * sin(PHI)) / ((cos(alpha_s)
        * cos(PHI)))
    cos_gamma_s = scale_in_min_max(cos_gamma_s)
    gamma_s = arccos(cos_gamma_s)
    if (st > 12):
```

```
        gamma_s = 2*pi - gamma_s
    return alpha_s, gamma_s
alpha_mat = np.zeros((12, 5))
gamma_mat = np.zeros((12, 5))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        alpha_mat[i,j] = alpha
        gamma_mat[i,j] = gamma
## 计算DNI
G0 = 1.366
H = 3
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_mat = np.zeros((12, 5))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(D[i], ST[j])
        DNI_mat[i,j] = G0*(a + b*pow(np.e,-c/sin(alpha)))
## 计算定日镜俯仰角，方位角
def cal_S_r(mirror_xyz):
    # 计算反射光线单位向量
    O = np.array([0 ,0, h_t])
    O_A = np.array(mirror_xyz)
    v = O - O_A
    v = v / np.linalg.norm(v)
    return v

def cal_S_i(alpha, gamma):
    # 计算入射光线单位向量
    # alpha, gamma是弧度制
    x = -cos(alpha) * cos(gamma-pi/2)
    y = cos(alpha) * sin(gamma-pi/2)
    z = -sin(alpha)
    return np.array([x, y, z])

# 计算定日镜法向量
S_i_mat = np.zeros((12, 5, 3))
S_r_mat = np.zeros((12, 5, *mirrors_xyz.shape))
```

```
S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        S_i_mat[i,j] = S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            S_r_mat[i,j,k] = S_r
            S_n_mat[i,j,k] = S_n
def cal_theta(S_n):
    # 计算定日镜俯仰角，方位角
    # 俯仰角：[0, pi/2)
    # 方位角：[0, 2*pi)
    x, y, z = S_n
    theta_z = arctan(sqrt(x**2 + y**2) / z)
    if theta_z < 0:
        theta_z = theta_z + pi
    theta_s = arcsin(x / sqrt(x**2 + y**2))
    if x > 0 and y < 0:
        theta_s = pi - theta_s
    elif x <= 0 and y >= 0:
        theta_s = 2*pi + theta_s
    elif x <= 0 and y < 0:
        theta_s = pi - theta_s
    return theta_z, theta_s
# 计算定日镜俯仰角，方位角
theta_mat = np.zeros((12, 5, mirrors_xyz.shape[0], 2))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            theta_z, theta_s = cal_theta(S_n)
            theta_mat[i,j,k] = [theta_z, theta_s]
# 计算大气透射率
def cal_ita_at(mirror_xyz):
```

```
    d = np.linalg.norm(mirror_xyz - np.array([0, 0, h_t]))
    return 0.99321 - 0.0001176*d + 1.97*10**(-8)*d**2
x = (8,8,4)
ml = x[0]
mw = x[1]
mh = x[2]
mirrors_xy = m_in_xy
m_num = mirrors_xy.shape[0]
mirrors_xyz = np.zeros((m_num,3))
for i in range(m_num):
    mirrors_xyz[i] = [*mirrors_xy[i], mh]
# 计算定日镜法向量
S_i_mat = np.zeros((12, 5, 3))
S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        S_i_mat[i,j] = S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            S_n_mat[i,j,k] = S_n
# 计算余弦效率
ita_cos_mat = np.zeros((12, 5, m_num))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            ita_cos = -S_n_mat[i,j,k] @ S_i_mat[i,j]
            ita_cos_mat[i,j,k] = ita_cos
# 计算大气透射率
ita_at_mat = np.zeros(m_num)
for k, mirror_xyz in enumerate(mirrors_xyz):
    ita_at_mat[k] = cal_ita_at(mirror_xyz)
ita_mat = np.zeros((12, 5, m_num))
E_mat = np.zeros((m_num))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_mat[i,j,k] = ita_at_mat[k] * ita_cos_mat[i,j,k]
```

```
            E_mat[k] += DNI_mat[i,j] * ita_mat[i,j,k] * mw * ml
E_mat /= 60
title = "镜场年平均能量传输效率"
cm = matplotlib.colormaps['rainbow']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1],
    c=ita_cos_mat[0].mean(axis=0), s=8, cmap=cm)
plt.title(title)
plt.colorbar(sc)
plt.savefig(title, dpi=200)
plt.show()
## 计算阴影遮挡
# 计算旋转矩阵
def cal_rot_mat(theta_z, theta_s):
    # 根据俯仰角和方位角，计算旋转矩阵
    beta, gamma = theta_z, theta_s
    sb, cb, sg, cg = sin(beta), cos(beta), sin(gamma), cos(gamma)
    return np.array([[cb*cg, -sg, sb*cg],
                     [sg*cb, cg, sb*sg],
                     [-sb,  0,   cb]])

rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            theta_z, theta_s = theta_mat[i,j,k]
            rot_mats[i,j,k] = cal_rot_mat(theta_z, theta_s)
def cal_distance(m1xy, m2xy):
    # 计算二维距离
    dis = np.linalg.norm(m1xy - m2xy)
    return dis
@njit
def in_bound(x, bound):
    # 判断点是否出了边界
    if (x < bound[0] or x > bound[1]):
        return False
    return True
@njit
def a_blocked_by_b(xy, V_0, day_id, st_id, a_id, b_id, shoot_in):
    m_x_bound, m_y_bound = [-4,4], [-4,4] # 定日镜坐标边界
    # shoot_in参数是指光为入射光还是为反射光，可取bool值
```

```
    # 后缀为0，a，b指地面坐标系，a镜坐标系，b镜坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    O_b = mirrors_xyz[b_id]
    if (O_a[:2] - O_b[:2]) @ V_0[:2] <= 0 and shoot_in:
        return False
    elif (O_a[:2] - O_b[:2]) @ V_0[:2] > 0 and not shoot_in:
        return False
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    b_rot_mat = rot_mats[day_id, st_id, b_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    h1_b = b_rot_mat.T @ (h1_0 - O_b)
    V_b = b_rot_mat.T @ (V_0)
    x2_b = h1_b[0] - V_b[0]*h1_b[2]/ V_b[2]
    y2_b = h1_b[1] - V_b[1]*h1_b[2]/ V_b[2]
    h2_b = np.array([x2_b, y2_b, 0])
    if in_bound(x2_b, m_x_bound) and in_bound(y2_b, m_y_bound):
        return True
    return False
@njit
def a_blocked_by_tower(xy, day_id, st_id, a_id):
    t_x_bound, t_z_bound = [-3.5, 3.5], [0, 84]
    # 后缀为0，a，t指地面坐标系，a镜坐标系，塔坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    if (O_a[1] <= 0):
        return False
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    alpha, gamma = alpha_mat[day_id,st_id], gamma_mat[day_id,st_id]
    t_xy_rot_mat = np.array([[-cos(gamma), -sin(gamma)],
                             [sin(gamma), -cos(gamma)]])
    xy_t = t_xy_rot_mat @ h1_0[0:2]
    h1_t = np.array([*xy_t, h1_0[2]])
    if h1_t[1] >= 0:
        z_delta = tan(alpha)*sqrt(xy[0]**2 + xy[1]**2)
        z2_t = z_delta + h1_t[2]
        h2_t = np.array([h1_t[0], 0, z2_t]) #
            h2_t为射向h1点的光与塔平面的交点
        if t_x_bound[0] <=h2_t[0] <= t_x_bound[1] and t_z_bound[0] <=
```

```
            h2_t[2] <=t_z_bound[1]:
309             return True
310     return False
311 # 蒙特卡洛法，将定日镜等分成36个1m^2区域，每一个区域取中心点
312 x_mont = np.linspace(-11/12*m_width, 11/12*m_width, 6)
313 y_mont = np.linspace(-11/12*m_length, 11/12*m_length, 6)
314 xy_mont = np.array([[x, y] for x in x_mont for y in y_mont])
315 # 取每面定日镜的最近邻8面镜子，判断是否发生阴影或遮挡
316 MAX_NEAR_NUM = 8
317 near_mat = np.zeros((mirrors_xyz.shape[0], MAX_NEAR_NUM), dtype=np.int64)
318 for a_id, m1xy in enumerate(mirrors_xy):
319     # 计算和其他镜子的距离
320     distances = {}
321     for b_id, m2xy in enumerate(mirrors_xy):
322         if a_id == b_id:
323             continue
324         distances[b_id] = cal_distance(m1xy, m2xy)
325     # 排序取出最近邻的8面镜子
326     ls = []
327     for new_b_id, _ in sorted(distances.items(), key = lambda kv:(kv[1],
            kv[0]))[:MAX_NEAR_NUM]:
328         ls.append(new_b_id)
329     near_mat[a_id] = ls
330 # 计算塔平面产生的阴影
331 blocked_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
        len(y_mont)), dtype=bool)
332 for i, d in enumerate(D):
333     for j, st in enumerate(ST):
334         for a_id, _ in enumerate(mirrors_xyz):
335             for x_id, x in enumerate(x_mont):
336                 for y_id, y in enumerate(y_mont):
337                     if a_blocked_by_tower([x,y], i, j, a_id):
338                         # 被塔挡住，产生阴影
339                         blocked_mat[i, j, a_id, x_id, y_id] = True
340
341 # 计算定光镜相互阴影
342 for i, d in enumerate(D):
343     for j, st in enumerate(ST):
344         S_i = S_i_mat[i, j]
345         for a_id, near_pairs in enumerate(near_mat):
```

```
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if a_blocked_by_b([x,y], S_i, i, j, a_id, b_id,
                            shoot_in=True):
                            # 把入射光挡住，产生阴影
                            blocked_mat[i, j, a_id, x_id, y_id] = True

# 计算遮挡
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for a_id, near_pairs in enumerate(near_mat):
            S_r = S_r_mat[i,j,a_id]
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if (not blocked_mat[i, j, a_id, x_id, y_id]) and \
                        a_blocked_by_b([x,y], S_r, i, j, a_id, b_id,
                            shoot_in=False):
                            # 没有阴影，并且反射光被挡住
                            blocked_mat[i, j, a_id, x_id, y_id] = True
ita_sb_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_sb_mat[i,j,k] = 1 - blocked_mat[i,j,k].sum() /
                (len(x_mont)*len(y_mont))

print("年平均阴影遮挡效率", ita_sb_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均阴影遮挡效率:", ita_sb_mat[i].mean())
cm = matplotlib.colormaps['rainbow']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1], vmin=0.6, vmax=1,
    c=ita_sb_mat.mean(axis=0).mean(axis=0), s=10, cmap=cm)
plt.colorbar(sc)
plt.show()
## 计算截断效率
MAX_ANGLE = 4.65 * 10**(-3)
angle_divide_num = 3
circle_divide_num = 6
```

```
tao_arr = np.linspace(0, 2*pi, circle_divide_num, endpoint=False)
sigma_arr = np.linspace(MAX_ANGLE, 0, angle_divide_num, endpoint=False)
light_mat = np.zeros((len(tao_arr) * len(sigma_arr), 3))
for i, tao in enumerate(tao_arr):
    for j, sigma in enumerate(sigma_arr):
        light_mat[len(sigma_arr)*i+j] = [sin(sigma)*cos(tao),
            sin(sigma)*sin(tao), cos(sigma)]
light_rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            S_r = S_r_mat[i,j,k]
            if (S_r[1] == 0):
                theta_x = pi / 2
            else:
                theta_x = arctan(-S_r[0] / S_r[1])
            x_s = np.array((cos(theta_x), sin(theta_x), 0))
            y_s = np.cross(S_r, x_s)
            # y_s = y_s / np.linalg.norm(y_s)
            light_rot_mats[i,j,k] = np.array((x_s, y_s, S_r)).T
R = 3.5
r_bound = [-3.5, 3.5]
machine_bound = [76, 84]
circle = Circle([0, 0], R)

def light_meet_machine(xy, light_0, day_id, st_id, m_id):
    m, n, l = light_0
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = rot_mat @ h1_a + O_a
    x1, y1, z1 = h1_0
    if (abs(m*y1 - n*x1) / sqrt(n**2+m**2) < R):
        line = Line([x1, y1], [x1+m, y1+n])
        point_a, point_b = circle.intersect_line(line)
        if abs(x1) < abs(y1):
            z = min(l*(point_a[1]-y1)/n+z1, l*(point_b[1]-y1)/n+z1)
        else:
            z = min(l*(point_a[0]-x1)/m+z1, l*(point_b[0]-x1)/m+z1)
        if in_bound(z, machine_bound):
```

```
            return True
    return False
trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
    len(y_mont)))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            for l_id, light in enumerate(light_mat):
                light_0 = light_rot_mats[i,j,k] @ light
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        trunc_mat[i,j,k,x_id,y_id] = angle_divide_num *
                            circle_divide_num
                        if blocked_mat[i,j,k,x_id,y_id]:
                            trunc_mat[i,j,k,x_id,y_id] = 0
                        if not blocked_mat[i,j,k,x_id,y_id] and \
                        not light_meet_machine([x,y], light_0, i, j, k):
                            trunc_mat[i,j,k,x_id,y_id] -= 1
        print(i,j)
ita_trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            cnt = 0
            for x_id, x in enumerate(x_mont):
                for y_id, y in enumerate(y_mont):
                    if not blocked_mat[i,j,k,x_id,y_id]:
                        cnt += 1
            if cnt == 0:
                ita_trunc_mat[i,j,k] = 0.94444444 # 对全阴影镜赋均值
            else:
                ita_trunc_mat[i,j,k] = trunc_mat[i,j,k].sum() / cnt /
                    (angle_divide_num * circle_divide_num)
print("年平均截断效率:", ita_trunc_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均截断效率:", ita_trunc_mat[i].mean())
## 计算DNI
G0 = 1.366
H = 3
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_mat = np.zeros((12, 5))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(D[i], ST[j])
        DNI_mat[i,j] = G0*(a + b*pow(np.e,-c/sin(alpha)))
## 计算光学效率
ita_ref = 0.92
ita_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_cos = ita_cos_mat[i,j,k]
            ita_at = ita_at_mat[i]
            ita_sb = ita_sb_mat[i,j,k]
            ita_trunc = ita_trunc_mat[i,j,k]
            ita_mat[i,j,k] = ita_cos * ita_sb * ita_at * ita_ref * ita_trunc
print("年平均光学效率:", ita_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均光学效率:", ita_mat[i].mean())
cm = matplotlib.colormaps['viridis']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1],
    c=ita_mat[i].mean(axis=0), s=10, cmap=cm)
plt.colorbar(sc)
plt.show()
## 计算热功率
E_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            E_mat[i,j,k] = DNI_mat[i,j] * m_length * m_width * ita_mat[i,j,k]
print("年单位面积镜面平均输出热功率:", E_mat.mean() / 36)
for i, _ in enumerate(D):
    print(f"{i+1:2}月单位面积镜面平均输出热功率:", E_mat[i].mean() / 36)
print("年平均输出热功率:", (E_mat.sum() / 5 / 12))
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均输出热功率(kW):", E_mat[i].sum() / 5)
```

附录 3: 问题三代码:

```python
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import sympy
from sympy import Symbol, solve
from datetime import datetime
from numpy import pi, sin, cos, tan, arcsin, arccos, arctan, sqrt
from sklearn.preprocessing import MinMaxScaler
from skspatial.objects import Circle, Line
from numba import njit
import random

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
def cal_distance(m1xy, m2xy):
    # 计算二维距离
    dis = np.linalg.norm(m1xy - m2xy)
    return dis
def cal_mod_length(x):
    return sqrt(x[0]**2+x[1]**2)
PHI = 39.4 / 180 * pi
ST = [9.0, 10.5, 12.0, 13.5, 15.0]
D = [(datetime(2023,i,21)-datetime(2023,3,21)).days for i in range(1, 13)]
h_t = 80 # 集热器中心高度
h_max = 84 # 塔顶端高度
h_m = 4 # 定日镜中心高度
m_length, m_width = 6, 6
m_x_bound, m_y_bound = [-3,3], [-3,3] # 定日镜坐标边界
t_x_bound, t_z_bound = [-3.5, 3.5], [0, h_max]
mirrors_xy = pd.read_excel("EB布局13.xlsx").values
center = (0, 250)
m_xy = mirrors_xy
m_in_xy = []
for xy in m_xy:
    if cal_distance(xy, center) < 350:
        m_in_xy.append(xy)
m_in_xy = np.array(m_in_xy)
```

```
mirrors_xy = m_in_xy
mirrors_xyz = np.insert(mirrors_xy, 2, values=h_m, axis=1)
for k, mirror_xy in enumerate(mirrors_xy):
    mirrors_xyz[k,2] = 4+(6-4)*cal_mod_length(mirror_xy)/600
pd.DataFrame(mirrors_xyz[:,2]).to_excel("z.xlsx")
mirrors_xyz[:,2].mean()
## 计算太阳高度角，方位角
def cal_omega(st):
    # 计算太阳时角
    return pi / 12 * (st - 12)

def cal_delta(d):
    # 计算太阳赤纬角
    sin_delta = sin(2*pi*d/365) * sin(2*pi*23.45/360)
    return arcsin(sin_delta)

def scale_in_min_max(num, min=-1, max=1):
    # 将结果限制在范围内
    if (num < min): return min
    elif (num > max): return max
    else: return num

def cal_alpha_and_gamma_s(d, st):
    # 计算太阳高度角和方位角
    omega = cal_omega(st)
    delta = cal_delta(d)
    sin_alpha_s = cos(delta)*cos(PHI)*cos(omega) + sin(delta)*sin(PHI)
    sin_alpha_s = scale_in_min_max(sin_alpha_s)
    alpha_s = arcsin(sin_alpha_s)
    cos_gamma_s = (sin(delta) - sin(alpha_s) * sin(PHI)) / ((cos(alpha_s)
        * cos(PHI)))
    cos_gamma_s = scale_in_min_max(cos_gamma_s)
    gamma_s = arccos(cos_gamma_s)
    if (st > 12):
        gamma_s = 2*pi - gamma_s
    return alpha_s, gamma_s
alpha_mat = np.zeros((12, 5))
gamma_mat = np.zeros((12, 5))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
```

```
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        alpha_mat[i,j] = alpha
        gamma_mat[i,j] = gamma
## 计算定日镜俯仰角，方位角
def cal_S_r(mirror_xyz):
    # 计算反射光线单位向量
    O = np.array([0 ,0, h_t])
    O_A = np.array(mirror_xyz)
    v = O - O_A
    v = v / np.linalg.norm(v)
    return v
def cal_S_i(alpha, gamma):
    # 计算入射光线单位向量
    # alpha，gamma是弧度制
    x = -cos(alpha) * cos(gamma-pi/2)
    y = cos(alpha) * sin(gamma-pi/2)
    z = -sin(alpha)
    return np.array([x, y, z])
# 计算定日镜法向量
S_i_mat = np.zeros((12, 5, 3))
S_r_mat = np.zeros((12, 5, *mirrors_xyz.shape))
S_n_mat = np.zeros((12, 5, *mirrors_xyz.shape))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        S_i_mat[i,j] = S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            S_r_mat[i,j,k] = S_r
            S_n_mat[i,j,k] = S_n
def cal_theta(S_n):
    # 计算定日镜俯仰角，方位角
    # 俯仰角：[0, pi/2)
    # 方位角：[0, 2*pi)
    x, y, z = S_n
    theta_z = arctan(sqrt(x**2 + y**2) / z)
    if theta_z < 0:
        theta_z = theta_z + pi
```

```
    theta_s = arcsin(x / sqrt(x**2 + y**2))
    if x > 0 and y < 0:
        theta_s = pi - theta_s
    elif x <= 0 and y >= 0:
        theta_s = 2*pi + theta_s
    elif x <= 0 and y < 0:
        theta_s = pi - theta_s
    return theta_z, theta_s
# 计算定日镜俯仰角，方位角
theta_mat = np.zeros((12, 5, mirrors_xyz.shape[0], 2))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        for k, mirror_xyz in enumerate(mirrors_xyz):
            S_r = cal_S_r(mirror_xyz)
            S_n = (S_r - S_i) / np.linalg.norm(S_r - S_i)
            theta_z, theta_s = cal_theta(S_n)
            theta_mat[i,j,k] = [theta_z, theta_s]
## 计算简单效率
# 计算余弦效率
ita_cos_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
        neg_S_i = -S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            ita_cos = S_n_mat[i,j,k] @ neg_S_i
            ita_cos_mat[i,j,k] = ita_cos

print("年平均余弦效率:", ita_cos_mat.mean())
for i, d in enumerate(D):
    print(f"{i+1:2}月平均余弦效率:", ita_cos_mat[i].mean())
# 计算余弦效率
ita_cos_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        alpha, gamma = cal_alpha_and_gamma_s(d, st)
        S_i = cal_S_i(alpha, gamma)
```

```
        neg_S_i = -S_i
        for k, mirror_xyz in enumerate(mirrors_xyz):
            ita_cos = S_n_mat[i,j,k] @ neg_S_i
            ita_cos_mat[i,j,k] = ita_cos

print("年平均余弦效率:", ita_cos_mat.mean())
for i, d in enumerate(D):
    print(ita_cos_mat[i].mean())
# 计算大气透射率
def cal_ita_at(mirror_xyz):
    d = np.linalg.norm(mirror_xyz - np.array([0, 0, h_t]))
    return 0.99321 - 0.0001176*d + 1.97*10**(-8)*d**2
ita_at_mat = np.zeros(mirrors_xyz.shape[0])
for i, mirror_xyz in enumerate(mirrors_xyz):
    ita_at_mat[i] = cal_ita_at(mirror_xyz)
## 计算阴影遮挡
# 计算旋转矩阵
def cal_rot_mat(theta_z, theta_s):
    # 根据俯仰角和方位角，计算旋转矩阵
    beta, gamma = theta_z, theta_s
    sb, cb, sg, cg = sin(beta), cos(beta), sin(gamma), cos(gamma)
    return np.array([[cb*cg, -sg, sb*cg],
                     [sg*cb, cg, sb*sg],
                     [-sb,  0,   cb]])

rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            theta_z, theta_s = theta_mat[i,j,k]
            rot_mats[i,j,k] = cal_rot_mat(theta_z, theta_s)
@njit
def in_bound(x, bound):
    # 判断点是否出了边界
    if (x < bound[0] or x > bound[1]):
        return False
    return True
@njit
def a_blocked_by_b(xy, V_0, day_id, st_id, a_id, b_id, shoot_in):
    m_x_bound, m_y_bound = [-3,3], [-3,3] # 定日镜坐标边界
```

```
    # shoot_in参数是指光为入射光还是为反射光，可取bool值
    # 后缀为0，a，b指地面坐标系，a镜坐标系，b镜坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    O_b = mirrors_xyz[b_id]
    if (O_a[:2] - O_b[:2]) @ V_0[:2] <= 0 and shoot_in:
        return False
    elif (O_a[:2] - O_b[:2]) @ V_0[:2] > 0 and not shoot_in:
        return False
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    b_rot_mat = rot_mats[day_id, st_id, b_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    h1_b = b_rot_mat.T @ (h1_0 - O_b)
    V_b = b_rot_mat.T @ (V_0)
    x2_b = h1_b[0] - V_b[0]*h1_b[2]/ V_b[2]
    y2_b = h1_b[1] - V_b[1]*h1_b[2]/ V_b[2]
    h2_b = np.array([x2_b, y2_b, 0])
    if in_bound(x2_b, m_x_bound) and in_bound(y2_b, m_y_bound):
        return True
    return False
@njit
def a_blocked_by_tower(xy, day_id, st_id, a_id):
    t_x_bound, t_z_bound = [-3.5, 3.5], [0, 84]
    # 后缀为0，a，t指地面坐标系，a镜坐标系，塔坐标系
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    if (O_a[1] <= 0):
        return False
    a_rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = a_rot_mat @ h1_a + O_a
    alpha, gamma = alpha_mat[day_id,st_id], gamma_mat[day_id,st_id]
    t_xy_rot_mat = np.array([[-cos(gamma), -sin(gamma)],
                             [sin(gamma), -cos(gamma)]])
    xy_t = t_xy_rot_mat @ h1_0[0:2]
    h1_t = np.array([*xy_t, h1_0[2]])
    if h1_t[1] >= 0:
        z_delta = tan(alpha)*sqrt(xy[0]**2 + xy[1]**2)
        z2_t = z_delta + h1_t[2]
        h2_t = np.array([h1_t[0], 0, z2_t]) #
            h2_t为射向h1点的光与塔平面的交点
```

```
237         if t_x_bound[0] <=h2_t[0] <= t_x_bound[1] and t_z_bound[0] <=
                h2_t[2] <=t_z_bound[1]:
238             return True
239     return False
240 # 蒙特卡洛法，将定日镜等分成36个1m^2区域，每一个区域取中心点
241 x_mont = [-2.5, -1.5, -0.5, 0.5, 1.5, 2.5]
242 y_mont = [-2.5, -1.5, -0.5, 0.5, 1.5, 2.5]
243 xy_mont = np.array([[x, y] for x in x_mont for y in y_mont])
244 # 取每面定日镜的最近邻8面镜子，判断是否发生阴影或遮挡
245 MAX_NEAR_NUM = 8
246 near_mat = np.zeros((mirrors_xyz.shape[0], MAX_NEAR_NUM), dtype=np.int64)
247 for a_id, m1xy in enumerate(mirrors_xy):
248     # 计算和其他镜子的距离
249     distances = {}
250     for b_id, m2xy in enumerate(mirrors_xy):
251         if a_id == b_id:
252             continue
253         distances[b_id] = cal_distance(m1xy, m2xy)
254     # 排序取出最近邻的8面镜子
255     ls = []
256     for new_b_id, _ in sorted(distances.items(), key = lambda kv:(kv[1],
            kv[0]))[:MAX_NEAR_NUM]:
257         ls.append(new_b_id)
258     near_mat[a_id] = ls
259 # 计算塔平面产生的阴影
260 blocked_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
        len(y_mont)), dtype=bool)
261 for i, d in enumerate(D):
262     for j, st in enumerate(ST):
263         for a_id, _ in enumerate(mirrors_xyz):
264             for x_id, x in enumerate(x_mont):
265                 for y_id, y in enumerate(y_mont):
266                     if a_blocked_by_tower([x,y], i, j, a_id):
267                         # 被塔挡住，产生阴影
268                         blocked_mat[i, j, a_id, x_id, y_id] = True
269
270 # 计算定光镜相互阴影
271 for i, d in enumerate(D):
272     for j, st in enumerate(ST):
273         S_i = S_i_mat[i, j]
```

```
        for a_id, near_pairs in enumerate(near_mat):
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if a_blocked_by_b([x,y], S_i, i, j, a_id, b_id,
                            shoot_in=True):
                            # 把入射光挡住，产生阴影
                            blocked_mat[i, j, a_id, x_id, y_id] = True

# 计算遮挡
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for a_id, near_pairs in enumerate(near_mat):
            S_r = S_r_mat[i,j,a_id]
            for b_id in near_pairs:
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        if (not blocked_mat[i, j, a_id, x_id, y_id]) and \
                        a_blocked_by_b([x,y], S_r, i, j, a_id, b_id,
                            shoot_in=False):
                            # 没有阴影，并且反射光被挡住
                            blocked_mat[i, j, a_id, x_id, y_id] = True
ita_sb_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_sb_mat[i,j,k] = 1 - blocked_mat[i,j,k].sum() /
                (len(x_mont)*len(y_mont))

print("年平均阴影遮挡效率", ita_sb_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均阴影遮挡效率:", ita_sb_mat[i].mean())
## 计算截断效率
MAX_ANGLE = 4.65 * 10**(-3)
angle_divide_num = 3
circle_divide_num = 6
tao_arr = np.linspace(0, 2*pi, circle_divide_num, endpoint=False)
sigma_arr = np.linspace(MAX_ANGLE, 0, angle_divide_num, endpoint=False)
light_mat = np.zeros((len(tao_arr) * len(sigma_arr), 3))
for i, tao in enumerate(tao_arr):
```

```
    for j, sigma in enumerate(sigma_arr):
        light_mat[len(sigma_arr)*i+j] = [sin(sigma)*cos(tao),
            sin(sigma)*sin(tao), cos(sigma)]
light_rot_mats = np.zeros((12, 5, mirrors_xyz.shape[0], 3, 3))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            S_r = S_r_mat[i,j,k]
            if (S_r[1] == 0):
                theta_x = pi / 2
            else:
                theta_x = arctan(-S_r[0] / S_r[1])
            x_s = np.array((cos(theta_x), sin(theta_x), 0))
            y_s = np.cross(S_r, x_s)
            # y_s = y_s / np.linalg.norm(y_s)
            light_rot_mats[i,j,k] = np.array((x_s, y_s, S_r)).T
R = 3.5
r_bound = [-3.5, 3.5]
machine_bound = [76, 84]
circle = Circle([0, 0], R)

def light_meet_machine(xy, light_0, day_id, st_id, m_id):
    m, n, l = light_0
    h1_a = np.array([*xy, 0])
    O_a = mirrors_xyz[a_id]
    rot_mat = rot_mats[day_id, st_id, a_id]
    h1_0 = rot_mat @ h1_a + O_a
    x1, y1, z1 = h1_0
    if (abs(m*y1 - n*x1) / sqrt(n**2+m**2) < R):
        line = Line([x1, y1], [x1+m, y1+n])
        point_a, point_b = circle.intersect_line(line)
        if abs(x1) < abs(y1):
            z = min(l*(point_a[1]-y1)/n+z1, l*(point_b[1]-y1)/n+z1)
        else:
            z = min(l*(point_a[0]-x1)/m+z1, l*(point_b[0]-x1)/m+z1)
        if in_bound(z, machine_bound):
            return True
    return False
trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0], len(x_mont),
    len(y_mont)))
```

```
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            for l_id, light in enumerate(light_mat):
                light_0 = light_rot_mats[i,j,k] @ light
                for x_id, x in enumerate(x_mont):
                    for y_id, y in enumerate(y_mont):
                        trunc_mat[i,j,k,x_id,y_id] = angle_divide_num *
                            circle_divide_num
                        if blocked_mat[i,j,k,x_id,y_id]:
                            trunc_mat[i,j,k,x_id,y_id] = 0
                        if not blocked_mat[i,j,k,x_id,y_id] and \
                        not light_meet_machine([x,y], light_0, i, j, k):
                            trunc_mat[i,j,k,x_id,y_id] -= 1
ita_trunc_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, d in enumerate(D):
    for j, st in enumerate(ST):
        for k, mirror_xyz in enumerate(mirrors_xyz):
            cnt = 0
            for x_id, x in enumerate(x_mont):
                for y_id, y in enumerate(y_mont):
                    if not blocked_mat[i,j,k,x_id,y_id]:
                        cnt += 1
            if cnt == 0:
                ita_trunc_mat[i,j,k] = 0.94444444 # 对全阴影镜赋均值
            else:
                ita_trunc_mat[i,j,k] = trunc_mat[i,j,k].sum() / cnt /
                    (angle_divide_num * circle_divide_num)
print("年平均截断效率:", ita_trunc_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均截断效率:", ita_trunc_mat[i].mean())
## 计算DNI
G0 = 1.366
H = 3
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_mat = np.zeros((12, 5))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
```

```
        alpha, gamma = cal_alpha_and_gamma_s(D[i], ST[j])
        DNI_mat[i,j] = G0*(a + b*pow(np.e,-c/sin(alpha)))
## 计算光学效率
ita_ref = 0.92
ita_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            ita_cos = ita_cos_mat[i,j,k]
            ita_at = ita_at_mat[i]
            ita_sb = ita_sb_mat[i,j,k]
            ita_trunc = ita_trunc_mat[i,j,k]
            ita_mat[i,j,k] = ita_cos * ita_sb * ita_at * ita_ref * ita_trunc
print("年平均光学效率:", ita_mat.mean())
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均光学效率:", ita_mat[i].mean())
## 计算热功率
E_mat = np.zeros((12, 5, mirrors_xyz.shape[0]))
for i, _ in enumerate(D):
    for j, _ in enumerate(ST):
        for k, _ in enumerate(mirrors_xyz):
            E_mat[i,j,k] = DNI_mat[i,j] * 8*8 * ita_mat[i,j,k]
cm = matplotlib.colormaps['viridis']
sc = plt.scatter(mirrors_xy[:,0], mirrors_xy[:,1],
    c=ita_mat[i].mean(axis=0), s=10, cmap=cm)
plt.colorbar(sc)
plt.show()
print("年单位面积镜面平均输出热功率:", E_mat.mean() / 36)
for i, _ in enumerate(D):
    print(f"{i+1:2}月单位面积镜面平均输出热功率:", E_mat[i].mean() / 36)
print("年平均输出热功率:", (E_mat.sum() / 5 / 12))
for i, _ in enumerate(D):
    print(f"{i+1:2}月平均输出热功率(kW):", E_mat[i].sum() / 5)
```