太阳能路灯光伏板的朝向设计问题

摘要

本文分析探究了太阳能光伏板的最优朝向，达到提高其发电效率及储能效率的目的。根据相关地理知识进行数据预处理，计算准确的当地时间，并进行了剔除和插值，获得更符合客观事实的相关数据，之后根据**微分方程**建立了**太阳辐射强度衰减模型**，采用**穷举法**和**模拟退火算法**计算出了太阳能光伏板的最优朝向，并针对光伏发电设施蓄电池的储电效率高和储电量大这两个目标，建立**多目标优化模型**，基于**多目标遗传算法**、**熵权法-优劣解距离法** 设计出了最优的光伏板安装朝向。

针对问题一，本文分析了太阳辐射穿过大气层时的衰减以及阳光倾斜照射在光伏板上造成的衰减，建立**微分方程模型**，确定了**大气衰减系数K**，同时引入**日-地距离修正系数**。在确定光伏板方位角等于正午时太阳方位角后计算出了θ =20°、40°、60°时2025年各月15日的最大太阳辐射强度，计算结果如表3。再通过对一天内所有时刻的太阳辐射积分得出了2025年各月15日的太阳直射辐射总能量，计算结果如表4。

针对问题二，本文建立了关于太阳日均辐射总能量的**优化模型**，采用了 **穷举法**和**模拟退火算法**分别计算出了最优水平倾角，得出最后的光伏板最优的安装水平倾角为25.5°，太阳直射辐射日均总能量最大值为13665.221443KJ，并利用相关实测数据进行了准确性分析。

针对问题三，本文针对储电效率高和储电量大两个目标建立了**多目标优化模型**。基于**遗传算法**来优化太阳辐射模型中光伏板方位角和水平倾角的取值，得到一组最终种群后，利用**熵权法-优劣解距离法**，对每一个个体进行评估，得出实现储电效率高和储电量大的两个目标的光伏板最优安装朝向，为水平倾角$\theta = 24.3782°$、光伏板方位角$A_0 = 2.1409°$。最后计算得出太阳直射辐射日均总能量为13660.88809KJ，高效发电时长为8.39h。

关键词：光伏板最优朝向；微分方程；穷举法；模拟退火算法；优化模型；遗传算法；熵权法-优劣解距离法

目录

1　问题重述

1.1　问题背景

随着全球对可再生能源的需求不断增加以及对环境保护的日益关注，新能源的发展已成为当今社会的重要议题。在这一背景下，太阳能作为一种清洁、可再生的能源备受关注，越来越多太阳能发电设施在生活中的应用逐渐广泛，如太阳能路灯。太阳能路灯主要依托于太阳能电池板即光伏板的光伏效应接受太阳辐射的能量，转化成电能并实现照明和储能。

光伏板发电量的决定因素为接收的太阳辐射量的大小，而由于地球自转与公转等因素，光伏板的安装朝向设计直接关系到能量的收集和转化效率。通过合理的朝向设计，可以最大化光伏板表面对太阳辐射的吸收、转换电能的效率以及储能效率。

1.2　问题提出

现需根据提供的数据以及相关定义，通过建立数学模型，解决以下问题：

（1）计算给定条件下该城区各每月15日的太阳能光伏板接收的最大太阳直射强度和太阳直射辐射总能量；

（2）设计该城区固定安装太阳能光伏板的最优朝向，以最大化太阳直射辐射日均总能量，从而使路灯蓄电池的储电量最大；

（3）光伏板受到太阳直射辐射强度过高时，储电效率会受到限制，理想情况为：光伏板受到太阳直射强度上午大于 $150W/m^2$、下午大于 $100W/m^2$ 的时间尽可能长。设计该城区光伏板固定安装的最优朝向，最终达到路灯蓄电池的储电效率高和储电量大这两个目标；并计算太阳直射辐射日均总能量和理想情况的时长。

2　问题分析

2.1　问题一的分析

问题一要求计算给定条件下该城区的最大太阳直射强度和太阳直射辐射总能量。

对于当地时间的理解：对于某城区地处北纬 30°35′，东经 114°19′ 而言，由于地球自转和地理经纬度差异的存在，该城区的当地时间是北京时间（120°E）减去22分钟45秒，为了简化计算，取当地时间与北京时间的差值的绝对值为23分钟。由于城区的在东八区的中央经线以西，该城区的时间应比北京时间更晚，即在北京时间的基础上与23分钟作差，即可得到城区时间。

对于最大太阳直射强度的理解：由地理学相关知识可知，当太阳时为当地正午12:00时，太阳高度角最大，太阳辐射最强，此时，太阳能电池的采光面若垂直于太阳光线，光伏发电板将会接受到太阳直射，瞬时太阳辐射强度最大。

2.2 问题二的分析

问题二要求设计该城区固定安装太阳能光伏板的最优朝向，以最大化太阳直射辐射日均总能量。

对于太阳直射辐射日均总能量的理解：光伏板受到的太阳直射辐射日均总能量最大，本文认为是以年作为时间尺度进行每日的总能量累加后，再除以总天数，使得该均值最大，即为太阳直射辐射日均总能量最大。

2.3 问题三的分析

问题三要求对于单个光伏板而言，要求其受到太阳直射强度上午大于 $150W/m^2$、下午大于$100W/m^2$的时间尽可能长。且要在尽可能保证发电效率高的情况下，光伏板受到的日均总辐射量更长，是一个多目标优化问题。对此，本文建立了一个多目标规划模型。

目标一：使得光伏干受到的太阳直射辐射总能量最大

目标二：使得光伏板储电效率尽可能高

3 模型假设

本文进行如下假定：

1. 在以天为单位计算时，假设太阳光到达大气层外层上的平均太阳能辐射强度不变。
2. 假设附件中sheet2中每个月的太阳辐射强度变化周期稳定，无年际变化。
3. 假设光伏板不会受到阴影遮挡。
4. 假设地球表面大气厚度保持1000Km不变。
5. 假设确定地区的大气衰减系数不变。
6. 假设不考虑地球曲率带来的影响。

4　符号说明

表 1: 符号说明

符号	说明	单位
K	衰减系数	$W/(m^2 \cdot km)$
I_s	太阳常数	W/m^2
I_0	大气层外层的太阳能辐射强度	W/m^2
ΔI	穿过大气层减弱的太阳辐射强度	W/m^2
I_1	穿过大气层后的太阳辐射强度	W/m^2
I_2	经过余弦损失后的太阳辐射强度	W/m^2
D	地球表面大气层厚度	km
γ	日-地距离修正系数	/
θ_r	太阳射线与斜面法线之间的夹角	°
φ	太阳方位角与光伏板方位角的差值	°
t_s	太阳时	°
ω	时角	°
δ	赤纬角	°
α	太阳高度角	°
ϕ	当地纬度	°
A	太阳方位角	°
A_0	光伏板方位角	°
θ	水平倾角	°
Q	太阳直射辐射总能量	kJ
T	昼长	h
T_{rise}	日出时刻	/
T_{set}	日落时刻	/

5　数据预处理

根据地理学中关于地球自转公转、经纬度等知识点，时间的计量以地球自转为依据，地球自转一周，计24太阳时，当太阳达到正南处为12:00。钟表所指的时间也称为平太阳时（简称为平时）。达到平太阳时之后，当地太阳辐射达到最大值，正午太阳高度角最大。

我国采用东经120度经圈上的平太阳时作为全国的标准时间，即“北京时间”。相对而言，该城区地处北纬 30°35′，东经 114°19′，其城区时间为在北京时间的基础上与23分钟作差。因此，

表 2: 修正前后本地时间

北京时间	当地精准时间	北京时间	当地精准时间
6:00	5:37	**13:00**	**12:37**
6:30	6:07	13:30	13:07
7:00	6:37	14:00	13:37
7:30	7:07	14:30	14:07
8:00	7:37	15:00	14:37
8:30	8:07	15:30	15:07
9:00	8:37	16:00	15:37
9:30	9:07	16:30	16:07
10:00	9:37	17:00	16:37
10:30	10:07	17:30	17:07
11:00	10:37	18:00	17:37
11:30	11:07	18:30	18:07
12:00	11:37	19:00	18:37
12:30	12:07		

经过修正之后的本地时间如表2所示。

经过修正后，发现地面直射辐射量的最大值并不在当地平太阳时（当地时间12:00左右）之上，与太阳辐射的地理原理存在偏差。为处理数据，本文以时间作为横坐标，以附件中 sheet1提供的测定地面直射辐射强度为纵坐标，作出时间—太阳辐射强度图，如图1。观察到时间-太阳辐射强度曲线存在较为明显的凸起和不平整，不适合作为后续的分析，因此，本文选择剔除了北京时间为“13:00”的数据点，并利用三次样条插值，分段构造三次多项式，保证数据平滑的同时，能够较好地避免因高次多项式拟合所产生的lounge现象，剔除、插值后得到的时间-太阳辐射强度曲线如图2。

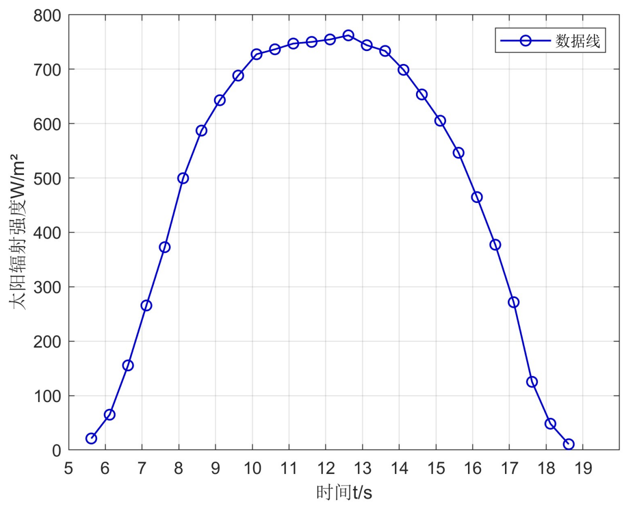

图 1: 时间-太阳辐射强度曲线

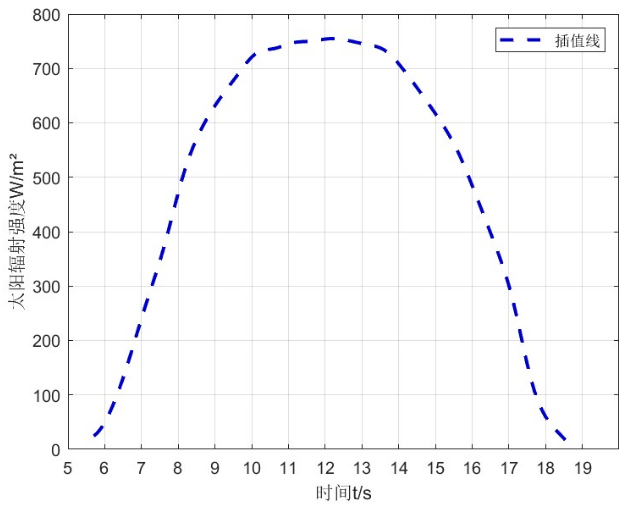

图 2: 插值后时间-太阳辐射强度曲线

可以看到，经过插值后的时间-太阳辐射强度曲线，在当地平太阳时（当地时间12:00）取得最大值，符合地理学认知规律。除此之外，曲线变得相对平滑，有利于后续数学分析。

6 模型的建立与求解

6.1 问题一模型的建立与求解

6.1.1 计算最大太阳直射强度

（1）构建太阳辐射强度衰减模型。

大气层外太阳辐射强度I_0在照射到光伏板的过程中要经过2次衰减。

首先是穿过大气层时大气层对太阳能直射辐射强度的衰减，此衰减量为ΔI，经衰减后太阳辐射强度为I_1,

$$I_1 = I_0 - \Delta I \tag{1}$$

ΔI 与其辐射强度I_0、所穿过大气层时光线传播时所经路程dl成正比,有：

$$\begin{cases} dI = -IKdl \\ I(0) = I_0 \end{cases} \tag{2}$$

进一步可得Bouguer公式[1]：

$$I = I_0 e^{-kl} \tag{3}$$

对于穿过大气层厚度D的理解：随着太阳入射方向的改变，其穿过的大气厚度将发生改变，如图3，大气厚度为D，太阳高度角为α，光线传播时所经路程为l，则

$$l = \frac{D}{\sin\alpha} \tag{4}$$

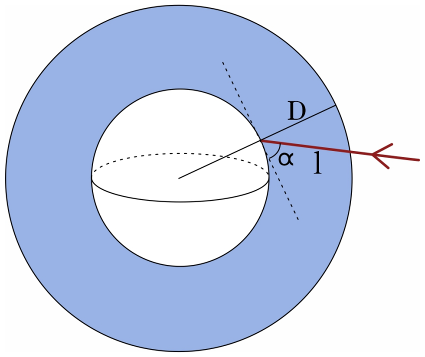

图 3: 大气层衰减示意图

由式(3)、式(4)可得最终

$$I_1 = I_0 e^{-\frac{KD}{\sin\alpha}} \tag{5}$$

其次是由于余弦损失造成的衰减，倾斜面上太阳光线的入射角θ_r 即太阳射线与斜面法线之间的夹角，可由下式确定[2]：

$$\cos\theta_r = \cos\theta\sin\alpha + \sin\theta\cos\alpha\cos(A - A_0) \tag{6}$$

其中，α为太阳高度角，θ为光伏板的水平倾角，A_0为光伏板方位角，A为太阳方位角；

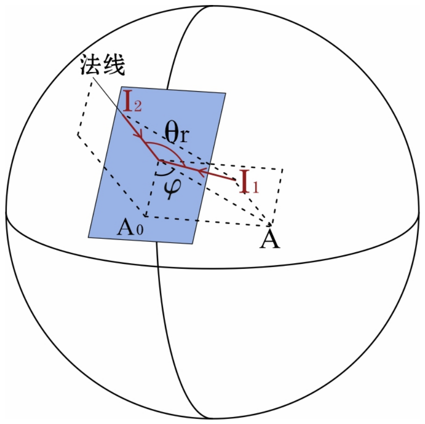

图 4: 余弦损失示意图

如下图，可得经此次衰减后太阳辐射强度I_2为：

$$I_2 = I_1 cos\theta_r \tag{7}$$

由式(1)—式(7)可得，最终光伏板接受到的有效太阳辐射强度I_2,

$$I_2 = I_0 e^{-\frac{KD}{\sin\alpha}} (\cos\theta \sin\alpha + \sin\theta \cos\alpha \cos(A - A_0)) \tag{8}$$

（2）计算不同条件下最大太阳辐射强度。

问题一条件为在晴天条件下，光伏板朝向正南方，即光伏板的方位角$A_0 = 0$，要求最大太阳辐射强度，当太阳时t_s 为当地正午12:00时，太阳高度角α最大，又

$$\varphi = A - A_0 = A = arcsin(\frac{-\sin\omega\cos\delta}{cos\alpha}) \tag{9}$$

$$\omega = \frac{\pi}{12}(t_s - 12) \tag{10}$$

$$sin\alpha = sin\phi sin\delta + cos\phi cos\delta cos\omega \tag{11}$$

其中，太阳方位角与光伏板方位角的差值$\varphi = A - A_0 = 0$；时角$\omega = 0$，ϕ为当地纬度，即30°35′；δ为赤纬角，只与日期有关，对于确定的日期，δ为常数，故α为常数。

（3）大气衰减系数K值的确定。

阳光穿过大气层时，大气层对太阳能直射辐射强度有一定的衰减，其大气衰减系数 K反映了一个地区大气层的透光性能。

附录sheet1已提供该城区（北纬 30°35′，东经 114°19′）2023年5月23日晴天状况下受到的太阳直射强度值，sheet2中提供了五月份的大气层上界太阳辐射均值，将之视为视为5月17日的大气层上界的太阳辐射通量。关于两日之间大气层上界太阳辐射的修正，本文采用了日-地距离修正

系数γ [2]:

$$\gamma = 1 + 0.034\cos\left(\frac{2\pi \times n}{365}\right) \tag{12}$$

$$I_0 = \gamma I_S \tag{13}$$

其中n为天数，I_S为太阳常数，其值为$1353W/m^2$。

对于日-地距离修正系数γ的引入，虽已有文章证明，但对于实际情况而言，处理仍需谨慎。本文利用线性回归的方法，利用公式（12）计算出每一天的太阳辐射强度之后，求取平均值，得到大气层上界太阳辐射强度月平均值I_0，利用附录sheet2中提供的每个月份的大气层上界太阳能辐射强度具体数值进行线性回归，以验证公式（13）的正确性。

回归结果：相关系数$r = 0.944$　均方根误差$MSE = 100.473$

回归结果良好，证明日-地距离修正系数γ的引入可靠。

日-地距离修正系数γ较好地描述了地球在近日点与远日点受到的太阳辐射之间的插值，同时，余弦分布函数很好地描述了地球公转的周期性。可得两日内的大气层上界太阳辐射通量满足如下关系：

$$\frac{I_{23}}{I_{17}} = \frac{1 + 0.034\cos\left(\frac{2\pi \times 143}{365}\right)}{1 + 0.034\cos\left(\frac{2\pi \times 137}{365}\right)} \tag{14}$$

利用上述关系，可以推导出5月23日该城区具体的大气上界太阳辐射值，并将其修正后，作为5月23日一整天的太阳辐射入射值，联立K值计算公式式(8)，得到K值为：5.5485×10^{-4}。

（4）计算结果。

由式(8)——式(14)可得，$\theta = 20°$、$40°$、$60°$时的最大太阳辐射强度。

表 3: 2025年各月15日最大太阳直射强度(W/m^2)

月份	一月	二月	三月	四月	五月	六月
$\theta = 20°$	483.4397	586.7309	683.8479	740.159	742.0672	729.7171
$\theta = 40°$	557.0147	640.1578	698.3367	700.6851	660.7029	629.3227
$\theta = 60°$	563.4056	616.3723	628.5959	576.6983	499.648	453.0227
月份	七月	八月	九月	十月	十一月	十二月
$\theta = 20°$	732.3373	738.1312	706.0357	619.3541	509.8573	451.9682
$\theta = 40°$	639.7841	679.5009	698.9705	659.8689	578.7003	528.6588
$\theta = 60°$	470.0634	538.9129	607.5991	620.7937	577.7435	541.5854

6.1.2 计算太阳直射辐射总能量

从日出到日落，太阳方位角A、太阳高度角α不断变化。当太阳视图面中心出现在地平线的瞬间，太阳高度$\alpha_{rise}=0$,若不考虑地表曲线及大气折射、散射的影响，由式（11）得：

$$\cos\omega_{rise}=-\tan\phi\tan\delta \tag{15}$$

其中ω_{rise}为日出时角；

由于$\cos\omega_{rise}=\cos(-\omega_{set})$，其中$\omega_{rise}$为日落时角，且按照时角规定，上午为正，下午为负，正值表示日出时角ω_{rise}，负值表示日落时角ω_{set}，则昼长T可以用

$$T=\frac{\omega_{rise}-\omega_{set}}{15^\circ/h} \tag{16}$$

来计算，则：

$$T=\frac{2}{15}\arccos\left(-\tan\delta\tan\theta\right) \tag{17}$$

$$T_{rise}=12:00-\frac{T}{2} \tag{18}$$

$$T_{set}=12:00+\frac{T}{2} \tag{19}$$

其中，T_{rise}、T_{set}分别为日出时刻、日落时刻；

故一天的太阳直射辐射总能量Q：

$$Q=\int_{T_{rise}}^{T_{set}} I_0e^{-\frac{KD}{\sin\alpha}}\cos\theta_r s dt_s \tag{20}$$

其中s为光伏板的面积，题中为$1m^2$，根据式(9)-式(20)计算结果：

表 4: 2025年各月15日太阳直射辐射总能量(KJ)

月份	一月	二月	三月	四月	五月	六月
$\theta=20^\circ$	8679.42	11347.03	14215.48	16387.12	17088.25	17096.99
$\theta=40^\circ$	10214.49	12575.8	14575.39	15255.29	14619.46	13965.92
$\theta=60^\circ$	10517.55	12287.75	13177.29	12283.45	10387.35	9150.34
月份	七月	八月	九月	十月	十一月	十二月
$\theta=20^\circ$	17042.51	16656.1	15094.2	12318.69	9356.98	7927.81
$\theta=40^\circ$	14187.25	14930.76	14891.36	13284.94	10834.98	9480.6
$\theta=60^\circ$	9620.8	11404.55	12892.41	12648.82	11006.12	9889.88

6.2 问题二模型的建立与求解

6.2.1 太阳直射辐射日均总能量最大值求解模型建立

由问题一的分析求解可知，每日的太阳直射辐射总能量Q不同，由式(20)得每日的Q值与光伏板的水平倾角θ 和方位角A_0相关，要求设计该城区固定安装太阳能光伏板的最优朝向，以最大化太阳直射辐射日均总能量，目标函数及约束条件如下：

$$\begin{cases} \max Q = \frac{1}{365}\sum_{n=1}^{365}\int_{T_{rise}}^{T_{set}} I_0 e^{-\frac{KD}{\sin\alpha}}\cos\theta_r s dt_s \\ I_0 = \gamma I_s \\ I_{0i} \geq 0 \\ \forall n, 0 \leq cos\theta_r \leq \frac{\pi}{2} \end{cases}; \tag{21}$$

要使Q最大，可以认为$I_0 e^{-\frac{KD}{\sin\alpha}}$和$\cos\theta_r$的乘积最大，此时有：

$$\begin{cases} 当\omega = 0时，I_0 e^{-\frac{KD}{\sin\alpha}}最大 \\ 当\omega = 0，A - A_0 = 0时，cos\theta_r最大 \end{cases}$$

即

$$\begin{aligned} \max Q &= \frac{1}{365}\sum_{n=1}^{365}\int_{T_{rise}}^{T_{set}} I_0 e^{-\frac{KD}{\sin\alpha}}\cos\theta_r s dt_s \\ &\leq \frac{1}{365}\sum_{n=1}^{365}\int_{T_{rise}}^{T_{set}} I_0 e^{-\frac{KD}{\sin\alpha}} sin(\theta+\alpha) s dt_s \end{aligned} \tag{22}$$

即当正午时分，太阳能光伏板的方位角一致时（$A = A_0$），Q最大

6.2.2 穷举法计算最优朝向

本文规定了以特定的步长x对0°< θ <90°有限个水平仰角进行穷举后，找出解的大致范围，接着，缩短步长，缩短解的可能范围后再次穷举，锁定该问题的最优解。

- 步长0.5°，θ范围为0-90°

  得到太阳直射辐射日均总能量的最大值为13665.22143 KJ，在水平倾角θ = 25.5°时取得最大值。

- 步长0.1°，θ范围为15-40°

  得到太阳直射辐射日均总能量的最大值为13665.22143 KJ，在水平倾角θ = 25.499999999999964°时取得最大值。

本算法流程图如下图：

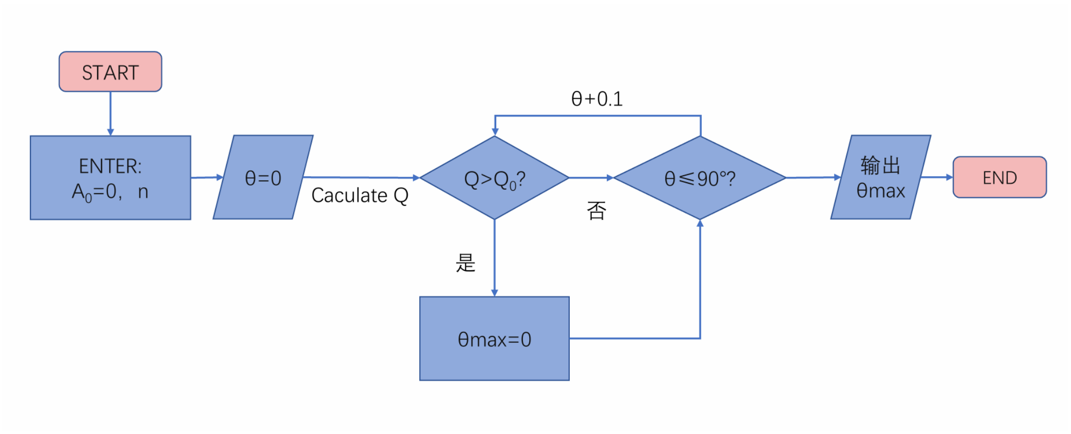

图 5: 穷举算法流程图

6.2.3 模拟退火算法获得最优朝向

本文构造了初始温度为100，最终温度为0.1，降温速率为0.95，最大迭代次数10000，初始状态30°进行迭代，找出最优解。

在当前温度下，以当前状态为基础，在解空间内随机选取一个邻域解（角度的微小变化）。计算目标函数（在此问题中即为平均日能量）的差值$\varDelta E$。

如果$\varDelta E < 0$，接受新解。如果$\varDelta E > 0$，以一定概率接受新解，这个概率随着温度的下降逐渐减小。重复以上步骤，直到系统温度降低到接近于0，或达到最大迭代次数为止，此时得到的角度即为最优角度。接受新解的概率由 Metropolis 准则给出，它的表达式是：

$$P(\Delta E) = \begin{cases} 1, & \text{if} \Delta E < 0 \\ e^{-\Delta E/T}, & \text{if} \Delta E > 0 \end{cases} \tag{23}$$

其中，$\varDelta E$ 是能量差，T 是当前温度。模拟退火某次迭代过程如图6：

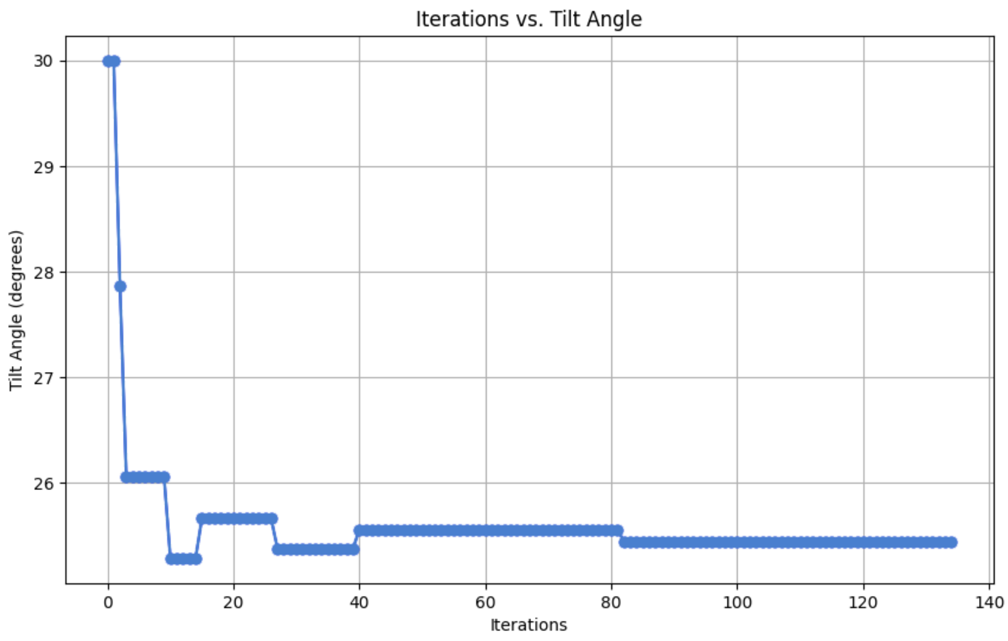

图 6: 模拟退火某次迭代过程图

由图，在迭代开始时，解的取值是随机的，解的分布比较广泛，处于种群空间中的不同位置。在迭代次数不断增加的过程中，解逐渐收敛到局部最优解或全局最优解附近，解的变化趋于稳定，不再发生剧烈变化。当迭代次数足够大时，解的取值趋于稳定，不断收敛到25.5附近。

最后得到太阳直射辐射日均总能量的最大值为13665.21039 KJ，在水平倾角$\theta = 25.5716146°$时取得最大值。

6.2.4 结果分析

结合穷举法和模拟退火算法，可以较为全面地搜索倾斜角度的最优解，比较两者的结果，十分相近，如表5，验证了结果的正确性。

表 5: 穷举法-模拟退火法对比

算法名称	日均能量（KJ/m^2）	最佳倾角（°）
穷举法	13665.22143	25.49999
模拟退火法	13665.21039	25.571615
穷举法-模拟退火法之差	0.01104	-0.071625

由于模拟退火算法的不确定性，取穷举法（步长为0.1°）的结果为最终结果，即光伏板最优的安装水平倾角为25.5°，太阳直射辐射日均总能量最大值为13665.221443KJ。

6.3 问题三模型的建立与求解

6.3.1 多目标优化模型建立

问题三实质是一个多目标组合优化问题，为此本文建立了一个多目标规划模型。对于两个目标的解释如下：

$$\begin{cases} \text{目标一：使得光伏干受到的太阳直射辐射总能量最大} \\ \text{目标二：使得光伏板储电效率尽可能高} \end{cases};$$

约束条件为：

$$\begin{cases} \text{上午时间段内光伏板所得到辐射强度大于}150W/m^2 \\ \text{下午时间段内光伏板所得到的辐射强度大于}100W/m^2 \\ \text{上午的时角为正，下午的视角为负} \\ \text{上午时角的取值必须在太阳升起之后，正午视角之前} \\ \text{下午的时角取值必须在正午视角以后，太阳落下之前} \end{cases};$$

要使储电效率最高，规定理想状态为：上午：光伏板受到太阳直射强度$\geq 150W/m^2$；下午：光伏板受到太阳直射强度$\geq 100W/m^2$。

定义太阳直射强度关于ω的函数：

$$F(\omega) = \gamma I_0 e^{-\frac{KD}{\sin\alpha}} \cos\theta_r \tag{24}$$

要使光伏板路灯蓄电池储电效率高，则理想状态时长长；同时要使路灯蓄电池储电量大，则有以下两目标函数：

$$\begin{cases} \max Q = \frac{T}{2\pi} \int_{-\omega_{max}}^{\omega_{max}} F(\omega)\, d\omega \\ \max T = \omega_{am} - \omega_{pm} \end{cases}; \tag{25}$$

注意到当光伏板方位角发生变化时，光伏板所能接受到的最大时角也会随之变化，其能够接受到太阳光的最大时角和最小视角根据下式确定：

$$\begin{cases} \omega_{\max} = \min\{A_0 + 90°, \omega_0\} \\ \omega_{\min} = \max\{-\omega_0, A_0 - 90°\} \end{cases}; \tag{26}$$

其中，$\omega_{\max}$、$\omega_{\min}$分别为日出和日落时刻的时角；

约束条件为:

$$\text{std.}\begin{cases} F(\omega_{am}) \geq 150 \\ F(\omega_{pm}) \geq 100 \\ 0 \leq \omega_{am} \leq \omega_{max} \\ \omega_{min} \leq \omega_{pm} \leq 0 \\ 0 \leq \omega \leq \pi \\ F(\omega) \geq 0 \end{cases}; \tag{27}$$

其中ω_{am}为上午达到理想状况之后的时角，ω_{pm}为下午达到理想状况之后的时角

6.3.2 基于遗传算法得出最优待选种群

（一）算法原理及描述

为解决上述多目标优化问题，本文利用了一个启发式算法：多目标遗传算法（Multi-objective Genetic Algorithm, MOGA）来优化太阳辐射模型中角度（θ）和A_0的取值。该算法使用 Python 中的 DEAP 库实现。遗传算法是一种基于生物进化过程的优化方法，通过模拟自然选择、交叉和变异等操作来搜索最优解。

在遗传算法中，定义了一个双目标适应度函数，以最大化平均太阳辐射(avg_q)和日照时长(avg_t)。适应度函数定义为：$fitness = avg_q + avg_t$。使用 DEAP 的 creator创建两个类：FitnessMax 和 Individual。FitnessMax 用于最大化适应度，而 Individual 是一个包含两个元素的列表，表示θ 和A_0。使用toolbox 生成初始种群、应用遗传操作(如交叉和变异)迭代。θ 和A_0的取值范围分别设定为 0 到 60 和 -90 到 90。 evaluate 函数通过计算平均太阳辐射和日照时长来计算种群中每个个体的适应度，使用$average_q$和 $average_T$函数进行计算。遗传算法的主循环运行固定数量的代（$NGEN$ = 40）。在每一代中，选择种群并通过交叉和变异操作生成后代。使用 evaluate 函数评估后代，选择最佳个体组成下一代种群。最终种群代表了θ 和A_0的最优解，以及相应的适应度值。算法描述图:

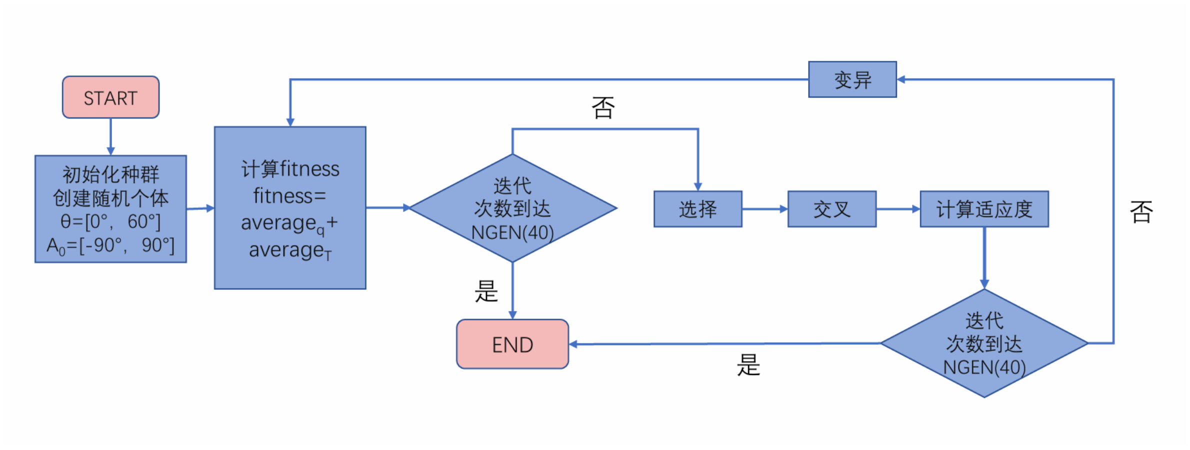

图 7: 遗传算法流程图

(二)参数设置

- 种群大小：40
- 交叉概率：0.7
- 变异概率：0.2

(三)优化策略

- 为了加速计算过程，采用了并行计算来提高遗传算法的效率。
- 参考相关文献[3]，将θ范围进一步缩小到 0° 到 60°

(四)迭代趋势可视化

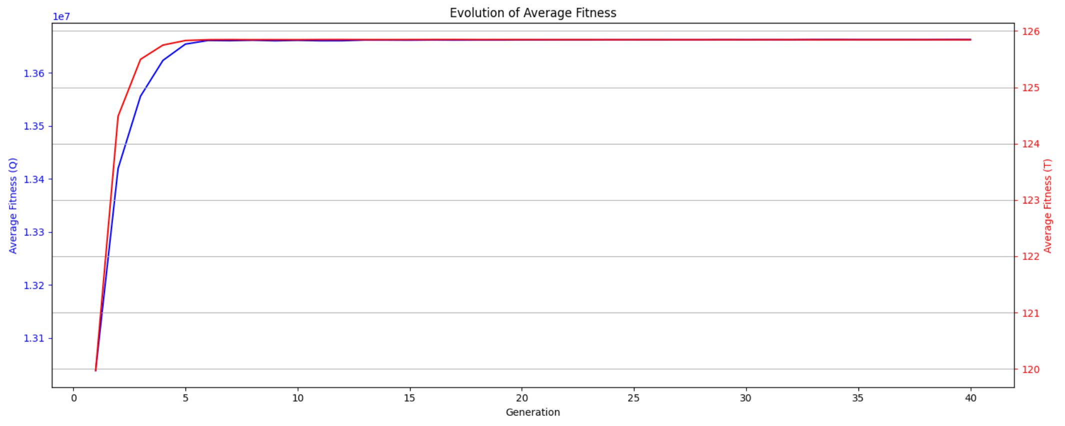

图 8: 某次迭代-Q-T变化

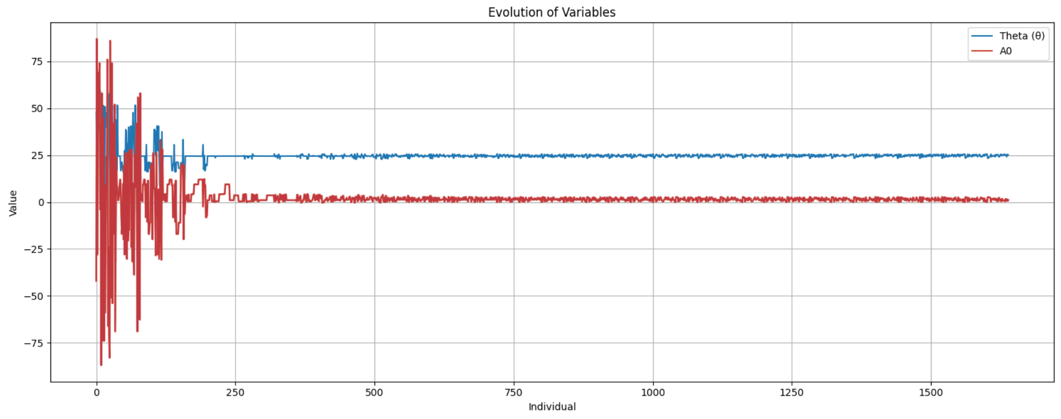

图 9: 某次迭代-θ-A_0变化

6.3.3 熵权法-优劣解距离法确定最优朝向

经过遗传算法得到一组最终种群之后，不难发现个体与个体之间的差异性并不明显，人为主观因素影响较为巨大。为了避免这个问题，本文利用熵权法-优劣解距离法，首先计算出光伏板受到的太阳直射辐射日均总能量（Q）和太阳直射辐射时长（T）各自的权重，得出结果为：$[\rho_Q,\rho_T]$=[0.3049 , 0.6951]。

然后利用优劣解距离法，对于每一项个体的合适度进行评估，得分最高的即为算法结果。最终得分最高的为水平倾角$\theta = 24.3782°$、光伏板方位角$A_0 = 2.1409°$，分数为0.013427。

进一步计算得出太阳直射辐射日均总能量为13660.88809KJ，达到理想状况（上午：光伏板受到太阳直射强度$\geq 150W/m^2$；下午：光伏板受到太阳直射强度$\geq 100W/m^2$）的时长为8.39h。

7 模型总结

7.1 模型一的准确性评价

关于模型一，本文建立了微分方程模型以描述大气衰减系数、大气透射后I_1，太阳光线穿过大气厚度之间的关系，接着，利用求解最大值的思想求解最大太阳直射强度、利用数值积分的方式计算了水平倾角分别为 20°、40°、60°时受到的太阳直射辐射总能量。

本文在建立模型一的建立过程中做了一些合适的化简，因此在模型的描述能力上会有所下降。不过，本文建立的模型相对清晰易懂，有着快速有效的解决实际问题的能力。

7.2 模型二的评价

（一）模型二的准确性评价

查询我国地理位置位于（$30°N$，$114°E$）的太阳能资源分布状况，发现该地区位于我国第三，第四类太阳能分布区之间，以此查询数据。《2022年中国风能太阳能资源年景公报》（以下简称“公报”）[4]直接给出了各省（区、市）2022年固定式光伏发电最佳斜面总辐照量平均值。由于本文与公报共同采用最佳倾角下获取太阳辐射的最大值，因此二者之间的比较为同一尺度，无需换算为水平地表。

取第四类分布区的最大值：全年太阳能辐射总量：$5000MJ/m^2$，约合每一天的太阳辐射总量为:$13.698 \times 10^6 J/m^2$ ，与本文计算结果$1.3665 \times 10^7 J/m^2$非常相近，这直接反映了该模型的正确性。

下表表6为关于模型二结果与更多实测数据的对比：

表 6: 模型二结果与实测数据对比

数据来源	实测数据	标准化后	模型二数据	相对误差
2022年中国风能太阳能资源年景公报	$1507.1KW/m^2$	1.4864×10^7	1.3665×10^7	**0.087742408**
贵州省能源局	$502 \times 10^4 KJ/m^2$	1.3753×10^7	1.3665×10^7	**0.00643981**

（二）模型二的误差分析

本文与所有的实测数据相比都相对偏小，与“公报”所提供的数据存在一定的偏差（8%），而与贵州省能源局给出的数据而言偏差不特别明显（千分之六），对于误差来源，本文给出如下可能的原因：

- 以点代面所产生的系统误差

  对于附件sheet1 中提供的该城区测定的地面水平太阳辐射总值，是以点数据呈现的；而“公报”等平台是以省市为单位所提供的平均数据，二者并不完全等同因此产生误差。

- 单独年份的气候波动

  “公报”指出，2022年全国太阳能源偏大，较近十年平均值偏大54KWh/m^2，较2021年偏大70KWh/m^2，因此2022年的数据本身是偏大的。而对于贵州省能源局则提供的是一个长期的相对稳定的指标，因此本模型与其的差值微乎其微。

- 模型简化带来计算误差

  如本文模型假设6，本模型并不考虑地球曲率带来的影响，因此太阳辐射穿过大气之后并不完全如同模型所预测一般，会存在路径偏差，造成误差。

7.3 模型三的准确性评价

对于模型三，我们利用启发式算法中的遗传算法进行最优种族的寻找，总共进行了两次模拟，总共生成了90个最优个体，在每个最优个体之间，本文利用熵权法-TOPSIS方法来确定在最优个体之间的最优分数。在结合智能算法的基础上，尽可能避免主观因素的影响，使得本文的结论真实可信。

7.4 模型整体评价

（1）运用的数学工具和编程语言较为简单，清晰易懂，对于任意倾斜角度都存在对应的计算公式，可推广性强；

（2）针对题目所提出的问题进行了较好的解答，结合地理学和数学的分析方法对模型进行判定，模型准确度和契合度极高，模型非常符合客观事实。

参考文献

[1] 吴继臣.全国主要城市冬季太阳辐射强度的研究[J].哈尔滨工业大学学报,2003(10):1236-1239.

[2] 方荣生. 太阳能应用技术[J]. 北京: 中国农业机械出版杜, 1985.

[3] 杨建华.太阳能热水器安装的最佳朝向和倾角[J].河南科技,1999(11):25.

[4] 2022年中国风能太阳能资源年景公报

A 附录

A.1 预处理

```
xi =5.10:0.1:19.10;
di=interp1(x,y,xi,'cubic');

plot(x1,y1 ,'-o',Color='r')
 title ("时间太阳辐射强度曲线-",'FontSize',19)
legend("数据线")
xlabel("时间t/s")
ylabel("太阳辐射强度W/²m")
set(gca,'XTick',[5:1:19]);
grid on
figure
plot(xi,di,'--',Color='r')
 title ("插值后","FontSize",19)
legend("插值线")
xlabel("时间t/s")
ylabel("太阳辐射强度W/²m")
set(gca,'XTick',[5:1:19]);
grid on
max(di)
```

A.2 问题一

Listing 1: calculate Imax monthly

```
from my_functions import calculate_hour_angle , calculate_max_radiation_at_angle , solar_radiation_model

# Constants
K = 0.0005548530101863522 # K calculated
d = 1000 # Thickness of the atmosphere : 1000 km

# day of year every month 15
```

```

day_of_year = [15, 46, 74, 105, 135, 166, 196, 227, 258, 288, 319, 349]

# hour angle 0
omega = calculate_hour_angle (12)
print(f"hour angle : {omega}")

# 30°35' the sun's altitude Angle
varphi = 30+(35/60)
print(f"latitude : {varphi}")



# Calculate and print the radiation at each angle monthly
import pandas as pd

angles = [20, 40, 60]

data = {"Tilt Angle (degrees)": [], "Month": [], "Max Radiation (W/m2)": []}

for angle in angles:
    print(f"When the tilt angle of the receiving panel is {angle} degrees:")
    for i, day_num in enumerate(day_of_year):
        month = i + 1
        max_radiation = calculate_max_radiation_at_angle (1353, d, K, varphi, day_num, omega, angle, 0)
        data["Tilt Angle (degrees)"].append(angle)
        data["Month"].append(month)
        data["Max Radiation (W/m2)"].append(max_radiation)
        print(f"\tMonth {month}: {max_radiation:.2f} W/m2")
df = pd.DataFrame(data)
df.to_excel("radiation_data_monthly.xlsx", index=False)

```

Listing 2: calculate K monthly

```
import math
from my_functions import calculate_declination_angle , calculate_hour_angle , calculate_altitude_angle
# Constants
I0 = 1330.296 # I0 in May
I1 = 754.717 # I1 after attenuation
d = 1000 # Thickness of the atmosphere : 1000 km

# 5/15 declination angle
delta = calculate_declination_angle (31+28+31+30+15)
print(f"declination angle : {delta}")

# hour angle 0
omega = calculate_hour_angle (12)
print(f"hour angle : {omega}")
```

```

# 30°35' the sun's altitude Angle
alpha = calculate_altitude_angle (30+(35/60), delta ,omega)
print(f"altitude angle : {alpha}")

# calculate K
K = math.sin(math.radians(alpha)) * math.log(I0 / I1) / d
print(f"K : {K}")

```

Listing 3: calculate total energy monthly

```
import math
from scipy.integrate import quad
from my_functions import calculate_max_radiation_at_angle , calculate_day_of_year

# Constants
I0 = 1353  # I0 average
d = 1000  # Thickness of the atmosphere: 1000 km
varphi = 30 + (35 / 60)  # Latitude
A0 = 0  # Azimuth of the board
K = 0.0005548530101863522 # K calculated
T = 1440  # Hours a day

# angles
angles = [20, 40, 60]

# 12 months of omega values
omega_values = [76.69934, 81.97589, 88.33245, 95.62391, 101.6014, 104.756,
                103.4743, 98.33642, 91.31096, 84.26344, 78.1584, 75.2289]

# Perform numerical integration for each tilt angle and month
for angle in angles:
    print(f"\nTilt Angle: {angle} degrees")
    for month, omega in enumerate(omega_values, start=1):
        day_of_year = calculate_day_of_year (2025, month, 15)

        def integrand_function (omega_value):
            return calculate_max_radiation_at_angle (I0, d, K, varphi, day_of_year, omega_value, angle, A0)

        lower_limit = -omega
        upper_limit = omega

        integral_value , _ = quad(integrand_function, lower_limit, upper_limit)
        integral_value *= T / (2 * math.pi)
        print(f"Total energy for Month {month}, Day 15: {round(integral_value/1000,2)} KJ")

```

Listing 4: solar radiation model accuracy evaluation

```python
import numpy as np
from scipy . stats  import  linregress

# Constants
Iaverage  = 1353

# Solar radiation intensity model function
def  solar_radiation_model (day_number, Iaverage ):
    return Iaverage  * (1  + 0.034  * np.cos(2  * np.pi  * day_number / 365))

# Calculate average radiation
def  calculate_average_radiation  (day_num_start,  day_num_end):
     total_radiation   = 0
    num_days = day_num_end - day_num_start + 1

    for  day_number in range(day_num_start,  day_num_end + 1):
        total_radiation   += solar_radiation_model (day_number, Iaverage )

     average_radiation   =   total_radiation   /  num_days
    return  average_radiation
def  calculate_average_radiation  (day_num_start,  day_num_end, Iaverage):
     total_radiation   = 0
    num_days = day_num_end - day_num_start + 1

    for  day_number in range(day_num_start,  day_num_end + 1):
        total_radiation   += solar_radiation_model (day_number, Iaverage )

     average_radiation   =   total_radiation   /  num_days
    return  average_radiation

def get_day_num(year,  month,  day):
    days_in_month = [31,  28,  31,  30,  31,  30,  31,  31,  30,  31,  30,  31]

     if  year  % 4 == 0 and (year days_in_month[1] = 29

    day_num = sum(days_in_month[:month - 1]) + day
    return day_num

def get_month_day_range(year,  month):
    days_in_month = [31,  28,  31,  30,  31,  30,  31,  31,  30,  31,  30,  31]
    start_day_num  = get_day_num(year,  month,  1)
    end_day_num = get_day_num(year, month, days_in_month[month-1])

    return start_day_num,  end_day_num

def  calculate_monthly_average ( year ,  month,  Iaverage ) :
        start_day , end_day = get_month_day_range(year, month)
        average_radiation  =  calculate_average_radiation ( start_day , end_day, Iaverage )
        return  average_radiation

# sheet 2 鏁版嵁
 solar_radiation_data   = {
        1:  1405,
        2:  1394,
        3:  1378,
        4:  1353,
        5:  1334,
        6:  1316,
        7:  1308,
        8:  1315,
        9:  1330,
        10:  1350,
        11:  1372,
        12:  1392,
}


 results  = [ calculate_monthly_average (2023, month, Iaverage ) for month in  solar_radiation_data  .keys () ]

 residuals  = [ calculate_monthly_average (2023, month, Iaverage ) — average_value for month, average_value in
         solar_radiation_data  .items () ]

 residuals_squared  = [ residual  ** 2 for  residual  in  residuals ]

（Mean Squared ,）ErrorMSE
mse = np.mean(residuals_squared)
print(f"Mean Squared Error (MSE): {mse}")

# 璁＄畻鐩稿叧绯绘暟
  correlation_coefficient  , _ =  linregress ( list ( solar_radiation_data  . values () ),  results ) [:2]
print(f"Correlation Coefficient: {correlation_coefficient}")
```

A.3 问题二

Listing 5: brute force optimization

```
import math
from scipy. integrate  import quad
import numpy as np
import random
from my_functions import  calculate_max_radiation_at_angle

# Constants
```

```
I0 = 1353  # I0 average
d = 1000  # Thickness of the atmosphere: 1000 km
varphi = 30 + (35 / 60)  # Latitude
A0 = 0  # Azimuth of the board
K = 0.0005548530101863522  # K calculated
T = 1440  # Hours a day

# Omega values for each day of the year
omega_values = [
    75.46278732, 75.5211872, 75.58438419, 75.65233709, 75.72500191, 75.80233199, 75.88427807, 75.97078841,
    76.06180891, 76.1572832, 76.25715276, 76.36135705, 76.46983361, 76.5825182, 76.6993449, 76.82024624,
    76.94515333, 77.07399597, 77.2067028, 77.34320137, 77.48341829, 77.62727935, 77.77470964, 77.92563364,
    78.07997533, 78.23765832, 78.39860596, 78.5627414, 78.72998771, 78.90026799, 79.07350543, 79.24962342,
    79.42854561, 79.61019601, 79.79449903, 79.98137959, 80.17076314, 80.36257575, 80.55674417, 80.75319584,
    80.95185898, 81.15266261, 81.35553658, 81.56041163, 81.7672194, 81.97589245, 82.18636429, 82.39856939,
    82.61244323, 82.82792223, 83.04494383, 83.26344647, 83.48336956, 83.70465354, 83.92723978, 84.15107066,
    84.37608951, 84.6022406, 84.82946912, 85.05772117, 85.28694373, 85.51708462, 85.74809251, 85.97991684,
    86.21250783, 86.44581644, 86.6797943, 86.91439372, 87.14956763, 87.38526954, 87.62145351, 87.85807408,
    88.09508627, 88.33244551, 88.57010759, 88.80802865, 89.04616509, 89.28447355, 89.52291085, 89.76143398,
    90, 90.23856602, 90.47708915, 90.71552645, 90.95383491, 91.19197135, 91.42989241, 91.66755449,
     91.90491373,
    92.14192592, 92.37854649, 92.61473046, 92.85043237, 93.08560628, 93.3202057, 93.55418356, 93.78749217,
    94.02008316, 94.25190749, 94.48291538, 94.71305627, 94.94227883, 95.17053088, 95.3977594, 95.62391049,
    95.84892934, 96.07276022, 96.29534646, 96.51663044, 96.73655353, 96.95505617, 97.17207777, 97.38755677,
    97.60143061, 97.81363571, 98.02410755, 98.2327806, 98.43958837, 98.64446342, 98.84733739, 99.04814102,
    99.24680416, 99.44325583, 99.63742425, 99.82923686, 100.0186204, 100.205501, 100.389804, 100.5714544,
    100.7503766, 100.9264946, 101.099732, 101.2700123, 101.4372586, 101.601394, 101.7623417, 101.9200247,
    102.0743664, 102.2252904, 102.3727206, 102.5165817, 102.6567986, 102.7932972, 102.926004, 103.0548467,
    103.1797538, 103.3006551, 103.4174818, 103.5301664, 103.6386429, 103.7428472, 103.8427168, 103.9381911,
    104.0292116, 104.1157219, 104.197668, 104.2749981, 104.3476629, 104.4156158, 104.4788128, 104.5372127,
    104.5907771, 104.6394707, 104.6832611, 104.7221192, 104.7560187, 104.784937, 104.8088545, 104.8277549,
    104.8416254, 104.8504566, 104.8542425, 104.8529805, 104.8466714, 104.8353195, 104.8189327, 104.7975219,
    104.7711018, 104.7396903, 104.7033084, 104.6619807, 104.6157349, 104.5646016, 104.5086148, 104.4478114,
    104.3822312, 104.3119166, 104.2369132, 104.1572688, 104.073034, 103.9842616, 103.8910071, 103.7933277,
    103.6912831, 103.5849347, 103.474346, 103.359582, 103.2407095, 103.1177966, 102.9909129, 102.8601291,
    102.7255173, 102.5871503, 102.445102, 102.2994469, 102.1502603, 101.997618, 101.8415961, 101.6822711,
    101.51972, 101.3540195, 101.1852466, 101.0134781, 100.8387909, 100.6612613, 100.4809655, 100.2979794,
    100.1123782, 99.92423687, 99.73362959, 99.54062996, 99.3453109, 99.14774457, 98.94800236, 98.74615485,
    98.54227174, 98.33642184, 98.12867306, 97.91909237, 97.70774576, 97.49469828, 97.28001398, 97.06375592,
    96.84598617, 96.62676579, 96.40615487, 96.1842125, 95.9609968, 95.73656491, 95.51097303, 95.28427643,
    95.05652946, 94.82778558, 94.59809739, 94.36751663, 94.13609425, 93.90388042, 93.67092453, 93.4372753,
    93.20298073, 92.96808822, 92.73264453, 92.49669588, 92.26028796, 92.02346599, 91.78627472, 91.54875856,
    91.31096152, 91.07292734, 90.83469948, 90.59632121, 90.35783562, 90.11928569, 89.88071431, 89.64216438,
    89.40367879, 89.16530052, 88.92707266, 88.68903848, 88.45124144, 88.21372528, 87.97653401, 87.73971204,
    87.50330412, 87.26735547, 87.03191178, 86.79701927, 86.5627247, 86.32907547, 86.09611958, 85.86390575,
    85.63248337, 85.40190261, 85.17221442, 84.94347054, 84.71572357, 84.48902697, 84.26343509, 84.0390032,
    83.8157875, 83.59384513, 83.37323421, 83.15401383, 82.93624408, 82.71998602, 82.50530172, 82.29225424,
    82.08090763, 81.87132694, 81.66357816, 81.45772826, 81.25384515, 81.05199764, 80.85225543, 80.6546891,
    80.45937004, 80.26637041, 80.07576313, 79.88762179, 79.70202065, 79.51903452, 79.33873875, 79.16120913,
    78.98652186, 78.8147534, 78.6459805, 78.48028001, 78.31772886, 78.15840395, 78.00238204, 77.84973967,
    77.70055305, 77.55489797, 77.41284965, 77.27448268, 77.13987087, 77.00908714, 76.88220342, 76.75929051,
    76.64041797, 76.52565399, 76.41506527, 76.30871693, 76.20667231, 76.10899294, 76.01573836, 75.92696604,
    75.84273121, 75.76308682, 75.68808337, 75.61776884, 75.55218857, 75.49138516, 75.4353984, 75.38426515,
    75.33801928, 75.29669159, 75.26030973, 75.22889817, 75.20247807, 75.18106733, 75.16468047, 75.15332863,
    75.14701955, 75.14575752, 75.1495434, 75.15837461, 75.17224514, 75.19114554, 75.21506299, 75.24398126,
    75.27788083, 75.31673886, 75.36052929, 75.4092229
]

# Objective Function
def objective_function(angle, omega_values):
    daily_energy = []
    for day, omega in enumerate(omega_values, start=1):
        day_of_year = day
        def integrand_function(omega_value):
            return calculate_max_radiation_at_angle(I0, d, K, varphi, day_of_year, omega_value, angle, A0)

        lower_limit = -omega
        upper_limit = omega

        integral_value, _ = quad(integrand_function, lower_limit, upper_limit)
        integral_value *= T / (2 * math.pi)

        daily_energy.append(integral_value)

    # Calculate the average daily energy for each tilt angle
    return np.mean(daily_energy)

# Simulated Annealing Function
def simulated_annealing(objective_function, omega_values, initial_angle, initial_temperature,
    final_temperature, alpha, max_iter):
    current_state = initial_angle
    T = initial_temperature

    for i in range(max_iter):
        new_state = current_state + random.uniform(-5, 5)
        delta_E = objective_function(new_state, omega_values) - objective_function(current_state,
    omega_values)

        if delta_E > 0:
            current_state = new_state
        else:
            if random.random() < np.exp(delta_E / T):
                current_state = new_state

        T = T * alpha

```

```
            if T <= final_temperature :
                break

    return current_state , objective_function ( current_state , omega_values)

# Define the range of tilt angles to be searched
initial_temperature = 100
final_temperature = 0.1
alpha = 0.95
max_iter = 10000
initial_angle = 30
optimal_angle, max_avg_energy = simulated_annealing( objective_function , omega_values, initial_angle ,
      initial_temperature , final_temperature , alpha, max_iter)

print(f"Optimal tilt angle: {optimal_angle} degrees")
print(f"Maximum average daily energy: {round(max_avg_energy, 2)} J")

```

Listing 6: calculate best angle

```
import math
from scipy . integrate import quad
import numpy as np
from my_functions import calculate_max_radiation_at_angle

# Constants
I0 = 1353 # I0 average
d = 1000 # Thickness of the atmosphere: 1000 km
varphi = 30 + (35 / 60) # Latitude
A0 = 0 # Azimuth of the board
K = 0.0005548530101863522 # K calculated
T = 1440 # Hours a day

# angles
angles = [20, 40, 60]

# Omega values for each day of the year
omega_values = [
    75.46278732, 75.5211872, 75.58438419, 75.65233709, 75.72500191, 75.80233199, 75.88427807, 75.97078841,
    76.06180891, 76.1572832, 76.25715276, 76.36135705, 76.46983361, 76.5825182, 76.6993449, 76.82024624,
    76.94515333, 77.07399597, 77.2067028, 77.34320137, 77.48341829, 77.62727935, 77.77470964, 77.92563364,
    78.07997533, 78.23765832, 78.39860596, 78.5627414, 78.72998771, 78.90026799, 79.07350543, 79.24962342,
    79.42854561, 79.61019601, 79.79449903, 79.98137959, 80.17076314, 80.36257575, 80.55674417, 80.75319584,
    80.95185898, 81.15266261, 81.35553658, 81.56041163, 81.7672194, 81.97589245, 82.18636429, 82.39856939,
    82.61244323, 82.82792223, 83.04494383, 83.26344647, 83.48336956, 83.70465354, 83.92723978, 84.15107066,
    84.37608951, 84.6022406, 84.82946912, 85.05772117, 85.28694373, 85.51708462, 85.74809251, 85.97991684,
    86.21250783, 86.44581644, 86.6797943, 86.91439372, 87.14956763, 87.38526954, 87.62145351, 87.85807408,
    88.09508627, 88.33244551, 88.57010759, 88.80802865, 89.04616509, 89.28447355, 89.52291085, 89.76143398,
        90, 90.23856602, 90.47708915, 90.71552645, 90.95383491, 91.19197135, 91.42989241, 91.66755449, 91.90491373,
        92.14192592, 92.37854649, 92.61473046, 92.85043237, 93.08560628, 93.3202057, 93.55418356, 93.78749217,
        94.02008316, 94.25190749, 94.48291538, 94.71305627, 94.94227883, 95.17053088, 95.3977594, 95.62391049,
        95.84892934, 96.07276022, 96.29534646, 96.51663044, 96.73655353, 96.95505617, 97.17207777, 97.38755677,
        97.60143061, 97.81363571, 98.02410755, 98.2327806, 98.43958837, 98.64446342, 98.84733739, 99.04814102,
        99.24680416, 99.44325583, 99.63742425, 99.82923686, 100.0186204, 100.205501, 100.389804, 100.5714544,
        100.7503766, 100.9264946, 101.099732, 101.2700123, 101.4372586, 101.601394, 101.7623417, 101.9200247,
        102.0743664, 102.2252904, 102.3727206, 102.5165817, 102.6567986, 102.7932972, 102.926004, 103.0548467,
        103.1797538, 103.3006551, 103.4174818, 103.5301664, 103.6386429, 103.7428472, 103.8427168, 103.9381911,
        104.0292116, 104.1157219, 104.197668, 104.2749981, 104.3476629, 104.4156158, 104.4788128, 104.5372127,
        104.5907771, 104.6394707, 104.6832611, 104.7221192, 104.7560187, 104.784937, 104.8088545, 104.8277549,
        104.8416254, 104.8504566, 104.8542425, 104.8529805, 104.8466714, 104.8353195, 104.8189327, 104.7975219,
        104.7711018, 104.7396903, 104.7033084, 104.6619807, 104.6157349, 104.5646016, 104.5086148, 104.4478114,
        104.3822312, 104.3119166, 104.2369132, 104.1572688, 104.073034, 103.9842616, 103.8910071, 103.7933277,
        103.6912831, 103.5849347, 103.474346, 103.359582, 103.2407095, 103.1177966, 102.9909129, 102.8601291,
        102.7255173, 102.5871503, 102.445102, 102.2994469, 102.1502603, 101.997618, 101.8415961, 101.6822711,
        101.51972, 101.3540195, 101.1852466, 101.0134781, 100.8387909, 100.6612613, 100.4809655, 100.2979794,
        100.1123782, 99.92423687, 99.73362959, 99.54062996, 99.3453109, 99.14774457, 98.94800236, 98.74615485,
        98.54227174, 98.33642184, 98.12867306, 97.91909237, 97.70774576, 97.49469828, 97.28001398, 97.06375592,
        96.84598617, 96.62676579, 96.40615487, 96.1842125, 95.9609968, 95.73656491, 95.51097303, 95.28427643,
        95.05652946, 94.82778558, 94.59809739, 94.36751663, 94.13609425, 93.90388042, 93.67092453, 93.4372753,
        93.20298073, 92.96808822, 92.73264453, 92.49669588, 92.26028796, 92.02346599, 91.78627472, 91.54875856,
        91.31096152, 91.07292734, 90.83469948, 90.59632121, 90.35783562, 90.11928569, 89.88071431, 89.64216438,
        89.40367879, 89.16530052, 88.92707266, 88.68903848, 88.45124144, 88.21372528, 87.97653401, 87.73971204,
        87.50330412, 87.26735547, 87.03191178, 86.79701927, 86.5627247, 86.32907547, 86.09611958, 85.86390575,
        85.63248337, 85.40190261, 85.17221442, 84.94347054, 84.71572357, 84.48902697, 84.26343509, 84.0390032,
        83.8157875, 83.59384513, 83.37323421, 83.15401383, 82.93624408, 82.71998602, 82.50530172, 82.29225424,
        82.08090763, 81.87132694, 81.66357816, 81.45772826, 81.25384515, 81.05199764, 80.85225543, 80.6546891,
        80.45937004, 80.26637041, 80.07576313, 79.88762179, 79.70202065, 79.51903452, 79.33873875, 79.16120913,
        78.98652186, 78.8147534, 78.6459805, 78.48028001, 78.31772886, 78.15840395, 78.00238204, 77.84973967,
        77.70055305, 77.55489797, 77.41284965, 77.27448268, 77.13987087, 77.00908714, 76.88220342, 76.75929051,
        76.64041797, 76.52565399, 76.41506527, 76.30871693, 76.20667231, 76.10899294, 76.01573836, 75.92696604,
        75.84273121, 75.76308682, 75.68808337, 75.61776884, 75.55218857, 75.49138516, 75.4353984, 75.38426515,
        75.33801928, 75.29669159, 75.26030973, 75.22889817, 75.20247807, 75.18106733, 75.16468047, 75.15332863,
        75.14701955, 75.14575752, 75.1495434, 75.15837461, 75.17224514, 75.19114554, 75.21506299, 75.24398126,
        75.27788083, 75.31673886, 75.36052929, 75.4092229
]

# Define the range of tilt angles to be searched

#first time in step 0.5 from 0 to 90



#second time in step 0.1 from 0 to 90
theta_range = np.arange(15, 40, 0.1)
daily_energies = []
for angle in theta_range:
```

```python
        daily_energy  = []
        for  day,  omega in enumerate(omega_values, start=1):
            day_of_year  = day

            def  integrand_function (omega_value):
                return  calculate_max_radiation_at_angle  (I0,  d,  K,  varphi,  day_of_year,  omega_value, angle,  A0)

            lower_limit  = −omega
            upper_limit  = omega

            integral_value ,  _ = quad(integrand_function ,  lower_limit ,  upper_limit )
            integral_value  *= T /  (2 * math.pi)

            daily_energy .append( integral_value )

        # Calculate the average daily energy for each tilt angle
        daily_energies .append(np.mean(daily_energy))
        print(f"angle now : {angle}" )
    # Find the tilt angle that maximizes the average daily energy
    optimal_angle_index  = np.argmax( daily_energies )
    optimal_angle  =  theta_range [ optimal_angle_index ]
    max_avg_energy = daily_energies [ optimal_angle_index ]

    print(f"Optimal tilt angle: {optimal_angle} degrees")
    print(f"Maximum average daily energy: {round(max_avg_energy, 2)} J")
```

Listing 7: 模拟退火迭代过程画图

```python
import math
from scipy. integrate  import quad
import numpy as np
import random
from my_functions import  calculate_max_radiation_at_angle
import  matplotlib . pyplot  as  plt

# Constants
I0  = 1353   # I0 average
d  = 1000   # Thickness of the atmosphere:  1000 km
varphi  = 30 + (35  /  60)   # Latitude
A0 = 0   # Azimuth of the board
K = 0.0005548530101863522  # K calculated
T  = 1440   # Hours a day

# Omega values for each day of the year
omega_values = [
    75.46278732, 75.5211872, 75.58438419, 75.65233709, 75.72500191, 75.80233199, 75.88427807, 75.97078841,
    76.06180891, 76.1572832, 76.25715276, 76.36135705, 76.46983361, 76.5825182, 76.6993449, 76.82024624,
    76.94515333, 77.07399597, 77.2067028, 77.34320137, 77.48341829, 77.62727935, 77.77470964, 77.92563364,
    78.07997533, 78.23765832, 78.39860596, 78.5627414, 78.72998771, 78.90026799, 79.07350543, 79.24962342,
    79.42854561, 79.61019601, 79.79449903, 79.98137959, 80.17076314, 80.36257575, 80.55674417, 80.75319584,
    80.95185898, 81.15266261, 81.35553658, 81.56041163, 81.7672194, 81.97589245, 82.18636429, 82.39856939,
    82.61244323, 82.82792223, 83.04494383, 83.26344647, 83.48336956, 83.70465354, 83.92723978, 84.15107066,
    84.37608951, 84.6022406, 84.82946912, 85.05772117, 85.28694373, 85.51708462, 85.74809251, 85.97991684,
    86.21250783, 86.44581644, 86.6797943, 86.91439372, 87.14956763, 87.38526954, 87.62145351, 87.85807408,
    88.09508627, 88.33244551, 88.57010759, 88.80802865, 89.04616509, 89.28447355, 89.52291085, 89.76143398,
    90, 90.23856602, 90.47708915, 90.71552645, 90.95383491, 91.19197135, 91.42989241, 91.66755449,
     91.90491373,
    92.14192592, 92.37854649, 92.61473046, 92.85043237, 93.08560628, 93.3202057, 93.55418356, 93.78749217,
    94.02008316, 94.25190749, 94.48291538, 94.71305627, 94.94227883, 95.17053088, 95.3977594, 95.62391049,
    95.84892934, 96.07276022, 96.29534646, 96.51663044, 96.73655353, 96.95505617, 97.17207777, 97.38755677,
    97.60143061, 97.81363571, 98.02410755, 98.2327806, 98.43958837, 98.64446342, 98.84733739, 99.04814102,
    99.24680416, 99.44325583, 99.63742425, 99.82923686, 100.0186204, 100.205501, 100.389804, 100.5714544,
    100.7503766, 100.9264946, 101.099732, 101.2700123, 101.4372586, 101.601394, 101.7623417, 101.9200247,
    102.0743664, 102.2252904, 102.3727206, 102.5165817, 102.6567986, 102.7932972, 102.926004, 103.0548467,
    103.1797538, 103.3006551, 103.4174818, 103.5301664, 103.6386429, 103.7428472, 103.8427168, 103.9381911,
    104.0292116, 104.1157219, 104.197668, 104.2749981, 104.3476629, 104.4156158, 104.4788128, 104.5372127,
    104.5907771, 104.6394707, 104.6832611, 104.7221192, 104.7560187, 104.784937, 104.8088545, 104.8277549,
    104.8416254, 104.8504566, 104.8542425, 104.8529805, 104.8466714, 104.8353195, 104.8189327, 104.7975219,
    104.7711018, 104.7396903, 104.7033084, 104.6619807, 104.6157349, 104.5646016, 104.5086148, 104.4478114,
    104.3822312, 104.3119166, 104.2369132, 104.1572688, 104.073034, 103.9842616, 103.8910071, 103.7933277,
    103.6912831, 103.5849347, 103.474346, 103.359582, 103.2407095, 103.1177966, 102.9909129, 102.8601291,
    102.7255173, 102.5871503, 102.445102, 102.2994469, 102.1502603, 101.997618, 101.8415961, 101.6822711,
    101.51972, 101.3540195, 101.1852466, 101.0134781, 100.8387909, 100.6612613, 100.4809655, 100.2979794,
    100.1123782, 99.92423687, 99.73362959, 99.54062996, 99.3453109, 99.14774457, 98.94800236, 98.74615485,
    98.54227174, 98.33642184, 98.12867306, 97.91909237, 97.70774576, 97.49469828, 97.28001398, 97.06375592,
    96.84598617, 96.62676579, 96.40615487, 96.1842125, 95.9609968, 95.73656491, 95.51097303, 95.28427643,
    95.05652946, 94.82778558, 94.59809739, 94.36751663, 94.13609425, 93.90388042, 93.67092453, 93.4372753,
    93.20298073, 92.96808822, 92.73264453, 92.49669588, 92.26028796, 92.02346599, 91.78627472, 91.54875856,
    91.31096152, 91.07292734, 90.83469948, 90.59632121, 90.35783562, 90.11928569, 89.88071431, 89.64216438,
    89.40367879, 89.16530052, 88.92707266, 88.68903848, 88.45124144, 88.21372528, 87.97653401, 87.73971204,
    87.50330412, 87.26735547, 87.03191178, 86.79701927, 86.5627247, 86.32907547, 86.09611958, 85.86390575,
    85.63248337, 85.40190261, 85.17221442, 84.94347054, 84.71572357, 84.48902697, 84.26343509, 84.0390032,
    83.8157875, 83.59384513, 83.37323421, 83.15401383, 82.93624408, 82.71998602, 82.50530172, 82.29225424,
    82.08090763, 81.87132694, 81.66357816, 81.45772826, 81.25384515, 81.05199764, 80.85225543, 80.6546891,
    80.45937004, 80.26637041, 80.07576313, 79.88762179, 79.70202065, 79.51903452, 79.33873875, 79.16120913,
    78.98652186, 78.8147534, 78.6459805, 78.48028001, 78.31772886, 78.15840395, 78.00238204, 77.84973967,
    77.70055305, 77.55489797, 77.41284965, 77.27448268, 77.13987087, 77.00908714, 76.88220342, 76.75929051,
    76.64041797, 76.52565399, 76.41506527, 76.30871693, 76.20667231, 76.10899294, 76.01573836, 75.92696604,
    75.84273121, 75.76308682, 75.68808337, 75.61776884, 75.55218857, 75.49138516, 75.4353984, 75.38426515,
    75.33801928, 75.29669159, 75.26030973, 75.22889817, 75.20247807, 75.18106733, 75.16468047, 75.15332863,
    75.14701955, 75.14575752, 75.1495434, 75.15837461, 75.17224514, 75.19114554, 75.21506299, 75.24398126,
    75.27788083, 75.31673886, 75.36052929, 75.4092229
]

# Objective Function
def objective_function (angle, omega_values):
```

```python
    daily_energy = []
    for day, omega in enumerate(omega_values, start=1):
        day_of_year = day
        def integrand_function(omega_value):
            return calculate_max_radiation_at_angle(I0, d, K, varphi, day_of_year, omega_value, angle, A0)

        lower_limit = -omega
        upper_limit = omega

        integral_value, _ = quad(integrand_function, lower_limit, upper_limit)
        integral_value *= T / (2 * math.pi)

        daily_energy.append(integral_value)

    # Calculate the average daily energy for each tilt angle
    return np.mean(daily_energy)



def plot_iterations_and_angles(iterations, angles):
    plt.figure(figsize=(10, 6))
    plt.plot(iterations, angles, marker='o', linestyle='-', color='b')
    plt.xlabel('Iterations')
    plt.ylabel('Tilt Angle (degrees)')
    plt.title('Iterations vs. Tilt Angle')
    plt.grid(True)
    plt.show()

# Simulated Annealing Function with modifications for tracking iterations and angles
def simulated_annealing_with_tracking(objective_function, omega_values, initial_angle, initial_temperature, final_temperature, alpha, max_iter):
    current_state = initial_angle
    T = initial_temperature
    iterations = []
    angles = []

    for i in range(max_iter):
        new_state = current_state + random.uniform(-5, 5)
        delta_E = objective_function(new_state, omega_values) - objective_function(current_state, omega_values)

        if delta_E > 0:
            current_state = new_state
        else:
            if random.random() < np.exp(delta_E / T):
                current_state = new_state

        T = T * alpha
```

```
            iterations .append(i)
            angles.append( current_state )

            if T <= final_temperature :
                break

    return current_state , objective_function ( current_state , omega_values), iterations , angles

# Define the range of tilt angles to be searched
initial_temperature = 100
final_temperature = 0.1
alpha = 0.95
max_iter = 10000
initial_angle = 30

optimal_angle, max_avg_energy, iterations , angles = simulated_annealing_with_tracking ( objective_function ,
    omega_values, initial_angle , initial_temperature , final_temperature , alpha, max_iter)

print(f"Optimal tilt angle: {optimal_angle} degrees")
print(f"Maximum average daily energy: {round(max_avg_energy, 2)} J")

# Plot iterations vs. angles
plot_iterations_and_angles ( iterations , angles)

```

A.4 问题三

Listing 8: Multi objective Genetic(parallel)

```
import math
import numpy as np
from scipy. integrate import quad
from joblib import Parallel , delayed
from deap import base, creator , tools
import random
import pandas as pd
from my_functions import calculate_declination_angle , calculate_hour_angle , calculate_altitude_angle ,
    calculate_cos_theta_r , calculate_max_radiation_at_angle , calculate_day_of_year , solar_radiation_model ,
    calculate_omega_0

# Constants
I0 = 1353 # I0 average
d = 1000 # Thickness of the atmosphere: 1000 km
varphi = 30 + (35 / 60) # Latitude
K = 0.0005548530101863522 # K calculated
T = 1440 # Hours a day

```

```
# calculate energy Q
def calculate_Q(I0, day_of_year, angle, A0):
    omega = calculate_omega_0(varphi, calculate_declination_angle (day_of_year))

    def integrand_function (omega_value):
        return calculate_max_radiation_at_angle (I0, d, K, varphi, day_of_year, omega_value, angle, A0)

    lower_limit = −omega
    upper_limit = omega

    integral_value , _ = quad(integrand_function , lower_limit , upper_limit )
    integral_value *= T / (2 * math.pi)
    return integral_value

# calculate day time T
def calculate_T(I0, day_of_year, angle, A0):
    def f(omega):
        return calculate_max_radiation_at_angle (I0, d, K, varphi, day_of_year, omega, angle, A0)

    omega_0 = calculate_omega_0(varphi, calculate_declination_angle (day_of_year))

    omega_min = min([omega_0,A0+90])
    omega_values_am = np.linspace(omega_min − 0.0001, 0, 1000)
    f_values_am = np.array ([f(omega) for omega in omega_values_am])
    omega_am_index = np.argmax(f_values_am > 150)
    omega_am = omega_values_am[omega_am_index]

    omega_max = max([−omega_0,A0−90])
    omega_values_pm = np.linspace(0, omega_max + 0.0001, 1000)
    f_values_pm = np.array ([f(omega) for omega in omega_values_pm])
    omega_pm_index = len(f_values_pm) − np.argmax(f_values_pm[::−1] > 100) − 1
    omega_pm = omega_values_pm[omega_pm_index]

    T = omega_am − omega_pm
    return T

# calculate average energy in a year
def average_q(I0, angle, A0):
    Q_values = Parallel (n_jobs=−1)(delayed(calculate_Q)(I0, day_of_year, angle, A0) for day_of_year in range(1,
        366))
    Q_average = np.mean(Q_values)
    return Q_average

# calculate average day time in a year
def average_T(I0, angle, A0):
    T_values = Parallel (n_jobs=−1)(delayed(calculate_T)(I0, day_of_year, angle, A0) for day_of_year in range(1,
        366))
    T_average = np.mean(T_values)
    return T_average

# Genetic Algorithm
creator.create("FitnessMax", base.Fitness, weights=(1.0, 1.0))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, 0, 60)
toolbox.register("attr_int", random.randint, -90, 90)
toolbox.register("individual", tools.initCycle, creator.Individual, (toolbox.attr_float, toolbox.attr_int), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def evaluate(individual):
    angle, A0 = individual
    avg_q = average_q(I0, angle, A0)
    avg_t = average_T(I0, angle, A0)
    return avg_q, avg_t

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selNSGA2)

def main():
    pop = toolbox.population(n=40)
    CXPB, MUTPB, NGEN = 0.7, 0.2, 40


    fitnesses = toolbox.map(toolbox.evaluate, pop)
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit


    for gen in range(NGEN):
        offspring = toolbox.select(pop, len(pop))
        offspring = list(map(toolbox.clone, offspring))


        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values

```

```

          invalid_ind  = [ind for ind in offspring if not ind. fitness . valid ]
          fitnesses  = toolbox .map(toolbox.evaluate ,  invalid_ind )
          for ind,  fit in zip( invalid_ind ,  fitnesses ):
              ind. fitness . values = fit


          pop = toolbox . select (pop + offspring , len(pop))

    return pop

if __name__ == "__main__":
    pop = main()
    # 灏嗙□缇ゅ拰閫傚簲搴︽Ă艰浆鎹□负 DataFrame
    pop_data = [( ind, ind. fitness . values [0], ind. fitness . values [1]) for ind in pop]
    df = pd.DataFrame(pop_data, columns=["Individual", "Avg Q", "Avg T"])

    # 灏咲ataFrame 杈撳嚭鍒癳xcel 鏂囦欢
    df. to_excel ("3_3rd_population_fitness.xlsx", index=False)
```

Listing 9: 熵权 TOPSIS

```
    P = stand_X ./ repmat(sum(stand_X),n,1);
    for i = 1 : n
        for j = 1 : m
            if (P(i,j) == 0)
                P(i,j) = 0.00001;
            end
        end
    end
    H_x = sum(−P .∗ log(P));
    e_j = H_x ./ log(n);
    d_j = 1 − e_j;
    disp('熵权完成，权值为：');
    w = d_j ./ sum(d_j)
    [n,m] = size(stand_X);
    tmp = ones(m);
    w_j = tmp(:,1);
    w_j = [0.3049,0.6951]
    T_plus = repmat(max_x,n,1);
    T_sub = repmat(min_x,n,1);
    F_plus = sum(((stand_X − T_plus).^2) ∗ w_j, 2).^0.5
    F_sub = sum(((stand_X − T_sub)).^2 ∗ w_j, 2).^0.5
    S = F_sub ./ (F_sub + F_plus)
     res_topsis = S ./ sum(S)
```

Listing 10: 遗传算法迭代过程画图

```
    import matplotlib . pyplot as plt
```

```
import math
import numpy as np
from scipy . integrate  import quad
from joblib  import  Parallel ,  delayed
from deap import base,  creator ,  tools
import random
import pandas  as  pd
from my_functions import  calculate_declination_angle  ,  calculate_hour_angle ,  calculate_altitude_angle  ,
      calculate_cos_theta_r ,  calculate_max_radiation_at_angle ,  calculate_day_of_year ,  solar_radiation_model ,
     calculate_omega_0

# Constants
I0  =  1353  # I0 average
d  =  1000  # Thickness of the atmosphere:  1000 km
varphi  = 30 + (35  /  60)  # Latitude
K = 0.0005548530101863522  # K calculated
T =  1440  # Hours a day

# calculate energy Q
def  calculate_Q(I0,  day_of_year,  angle,  A0):
    omega = calculate_omega_0(varphi ,   calculate_declination_angle  (day_of_year))

    def  integrand_function (omega_value):
        return  calculate_max_radiation_at_angle  (I0,  d,  K,  varphi ,  day_of_year,  omega_value, angle,  A0)

    lower_limit  = −omega
    upper_limit  = omega

    integral_value ,  _ = quad( integrand_function ,  lower_limit ,  upper_limit )
    integral_value  ∗= T /  (2  ∗ math.pi)
    return  integral_value

# calculate day time T
def  calculate_T (I0,  day_of_year,  angle,  A0):
    def  f(omega):
        return  calculate_max_radiation_at_angle  (I0,  d,  K,  varphi ,  day_of_year,  omega, angle,  A0)

    omega_0 = calculate_omega_0(varphi ,   calculate_declination_angle  (day_of_year))

    omega_min = min([omega_0,A0+90])
    omega_values_am = np.linspace(omega_min − 0.0001, 0, 1000)
    f_values_am = np. array ([ f(omega) for  omega in  omega_values_am])
    omega_am_index = np.argmax(f_values_am > 150)
    omega_am = omega_values_am[omega_am_index]

    omega_max = max([−omega_0,A0−90])
    omega_values_pm = np.linspace(0,  omega_max + 0.0001, 1000)
    f_values_pm = np. array ([ f(omega) for  omega in  omega_values_pm])
```

```
        omega_pm_index = len(f_values_pm) − np.argmax(f_values_pm[::−1] > 100) − 1
        omega_pm = omega_values_pm[omega_pm_index]

        T = omega_am − omega_pm
        return T

    # calculate average energy in a year
    def average_q(I0, angle, A0):
        Q_values = Parallel (n_jobs=−1)(delayed(calculate_Q)(I0, day_of_year, angle, A0) for day_of_year in range
          (1, 366))
        Q_average = np.mean(Q_values)
        return Q_average

    # calculate average day time in a year
    def average_T(I0, angle, A0):
        T_values = Parallel (n_jobs=−1)(delayed(calculate_T)(I0, day_of_year, angle, A0) for day_of_year in range
          (1, 366))
        T_average = np.mean(T_values)
        return T_average

    # Genetic Algorithm
    creator . create ("FitnessMax", base.Fitness, weights =(1.0, 1.0))
    creator . create ("Individual", list, fitness =creator .FitnessMax)
    toolbox = base.Toolbox()
    toolbox. register ("attr_float", random.uniform, 0, 60)
    toolbox. register ("attr_int", random.randint, −90, 90)
    toolbox. register ("individual", tools. initCycle , creator . Individual , (toolbox. attr_float , toolbox. attr_int ), n
        =1)
    toolbox. register ("population", tools. initRepeat , list , toolbox. individual )

    def evaluate ( individual ):
        angle, A0 = individual
        avg_q = average_q(I0, angle, A0)
        avg_t = average_T(I0, angle, A0)
        return avg_q, avg_t

    toolbox. register ("evaluate", evaluate)
    toolbox. register ("mate", tools .cxBlend, alpha=0.5)
    toolbox. register ("mutate", tools .mutGaussian, mu=0, sigma=1, indpb=0.2)
    toolbox. register ("select", tools .selNSGA2)

    def main():
        pop = toolbox. population (n=40)
        CXPB, MUTPB, NGEN = 0.7, 0.2, 40



         theta_values = []
        A0_values = []

    # 璇勮及鏁翠釜绉嶇兢
    fitnesses = toolbox.map(toolbox.evaluate, pop)
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit
        theta_values.append(ind[0])
        A0_values.append(ind[1])


    avg_q_list = []
    avg_t_list = []


    for gen in range(NGEN):
        offspring = toolbox.select(pop, len(pop))
        offspring = list(map(toolbox.clone, offspring))

        # 瀵瑰悗浠ｈ繘琛屼氦鍙夊拰鍙樺紓鎿嶄綔
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values


        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit


        pop = toolbox.select(pop + offspring, len(pop))

        avg_q_values = [ind.fitness.values[0] for ind in pop]
        avg_t_values = [ind.fitness.values[1] for ind in pop]
        avg_q_list.append(np.mean(avg_q_values))
        avg_t_list.append(np.mean(avg_t_values))


        theta_values.extend([ind[0] for ind in pop])
        A0_values.extend([ind[1] for ind in pop])

    # 缁樺埗杩涘寲鏇茬嚎
    fig, ax1 = plt.subplots(figsize=(15, 6))

    ax1.plot(range(1, NGEN + 1), avg_q_list, label='Average Q', color='b')
    ax1.set_xlabel('Generation')
    ax1.set_ylabel('Average Fitness (Q)', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(range(1, NGEN + 1), avg_t_list, label='Average T', color='r')
    ax2.set_ylabel('Average Fitness (T)', color='r')
    ax2.tick_params('y', colors='r')

    fig.tight_layout()
    plt.title('Evolution of Average Fitness')
    plt.grid(True)

    plt.figure(figsize=(15, 6))
    plt.plot(theta_values, label='Theta θ()')
    plt.plot(A0_values, label='A0')
    plt.xlabel('Individual')
    plt.ylabel('Value')
    plt.title('Evolution of Variables')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    return pop

if __name__ == "__main__":
    pop = main()
    # 灏嗘□缇ゅ拰閫傚簲搴﹀Ă艰浆鎹□负 DataFrame
    pop_data = [(ind, ind.fitness.values[0], ind.fitness.values[1]) for ind in pop]
    df = pd.DataFrame(pop_data, columns=["Individual", "Avg Q", "Avg T"])

    # 灏咲ataFrame 杈撳嚭鍒癳xcel 鏂囦欢
    df.to_excel("3_3rd_population_fitness.xlsx", index=False)
```

Listing 11: my functions

```
import math
import numpy as np
import datetime


# Calculate the declination Angle of the sun 未
def calculate_declination_angle(day_of_year):
    delta = 23.45 * math.sin(2 * math.pi * (284 + day_of_year) / 365)
    return delta

# Calculate the solar hour Angle 蝇
def calculate_hour_angle(hour):
    omega = 15 * (hour - 12)
    return omega

# Calculate the sun's altitude Angle 伪
def calculate_altitude_angle(latitude, delta, omega):
    phi = math.radians(latitude)
    alpha = math.degrees(math.asin(math.sin(phi) * math.sin(math.radians(delta)) + math.cos(phi) * math.cos(
     math.radians(delta)) * math.cos(math.radians(omega))))
    return alpha

# Calculation the angle between the incident sunlight and the surface normal
def calculate_cos_theta_r(theta, varphi, delta, omega, A0):
    theta_rad = math.radians(theta)
    varphi_rad = math.radians(varphi)
    delta_rad = math.radians(delta)
    omega_rad = math.radians(omega)
    A0_rad = math.radians(A0)
    cos_theta_r = math.cos(theta_rad) * math.sin(varphi_rad) * math.sin(delta_rad) \
        + math.cos(theta_rad) * math.cos(varphi_rad) * math.cos(delta_rad) * math.cos(omega_rad) \
        + math.sin(theta_rad) * math.sin(A0_rad) * math.cos(delta_rad) * math.sin(omega_rad) \
        + math.sin(theta_rad) * math.sin(varphi_rad) * math.cos(delta_rad) * math.cos(omega_rad) * math.cos(
     A0_rad) \
        - math.sin(theta_rad) * math.cos(A0_rad) * math.sin(delta_rad) * math.cos(varphi_rad)
    return cos_theta_r

# Caculate solar radiation intensity
def calculate_max_radiation_at_angle(I0, d, K, varphi, day_of_year, omega, angle, A0):
    delta = calculate_declination_angle(day_of_year)
    alpha = calculate_altitude_angle(varphi, delta, omega)
    alpha_rad = math.radians(alpha)
    cos_theta_r = calculate_cos_theta_r(angle, varphi, delta, omega, A0)
    Is = solar_radiation_model(day_of_year, I0)
    max_radiation = Is * cos_theta_r * math.exp(-K * d / math.sin(alpha_rad))
    return max_radiation

# Calculate day in year N
def calculate_day_of_year(year, month, day):
    # Create a datetime object for the given date
    date_obj = datetime.date(year, month, day)
    # Get the day of the year from the datetime object
    day_of_year = date_obj.timetuple().tm_yday
    return day_of_year

# Solar radiation intensity model function
```

```python
def  solar_radiation_model (day_number, Imax):
    return Imax * (1  + 0.034 * np.cos(2 * np.pi * day_number / 365))

# Calculate the azimuth angle of sunrise
def calculate_omega_0(phi,  delta):
    phi_rad  = math.radians(phi)
    delta_rad  = math.radians(delta)
    omega_0_rad = math.acos(-math.tan(delta_rad) * math.tan(phi_rad))
    omega_0_deg = math.degrees(omega_0_rad)
    return omega_0_deg
```