基于机理分析的光伏板朝向决策模型

摘要

本文研究太阳能光伏板的朝向问题，运用相关的**光学定理**进行**机理分析**，建立了单位面积上光伏板的朝向和接收太阳直射总能量的模型。并且分别对于单目标和多目标构建优化模型，找到在不同目标的最优解。

对于问题一，在三维直角坐标系上使用**球体距离公式**得到大气厚度，结合题目给定的太阳直射强度 $1353W/(m^2)$，拟合出大气对太阳直射能量的**衰减系数**，根据先验知识，加上随时间变化的**三次偏差项**，得到拟合结果在训练集上的 R 方为 **95%**，表明拟合效果良好，得出大气对太阳直射的**衰减系数**为 $8.6833\times10^{-6}W/(m^2\cdot km)$。接着在三维直角坐标系上，运用几何知识构建出太阳入射光线的直线向量和光伏板的法向量，计算**两向量余弦**，得到每个小时在光伏板上吸收的能量。最后在光伏板水平倾角分别 20°、40°、60° 时计算得到**最大太阳直射强度和太阳直射辐射总能量**，具体结果呈现在文章表格内。

对于问题二，本文将 25 年每月 15 日太阳**直射辐射总能量**设为目标，将光伏板水平倾角和光伏板的方位角作为变量，构建基于**禁忌搜索**的 **MPSO 算法**对模型进行求解，模型**迅速收敛**，收敛效果在附件中给出，同时得到最优的水平倾角和光伏板的方位角分别为 $24.1474°$ 和 $-2.8694°$。使用该解计算出光伏板在晴天条件下受到的太阳直射辐射日均总能量最大值为 **7306**W。

针对问题三，为了实现**储电效率高**和**储电量大**两个目标，具体量化为光伏板太阳直射辐射日均总能量和太阳直射辐射时长（上午大于 $150W/m^2$ 、下午大于 $100W/m^2$ ）。本文建立了双目标规划模型，分别使用 **NSGA-II 算法**在 **Pareto 前沿解集**和**网格搜索算法**确定步长为一情况下的最优值，发现两个方法给出的最优值均落在光伏板方位角为 0 度，水平倾角为 30 度附近，基于两个算法**互相验证**下，本文使用 NSGA-II 算法得到的结果（$-1.1184°$，$31.1970°$）得出太阳直射辐射日均总能量为 **9534.2903**W，以及每日平均太阳辐射时长为 **9.9735**h。

本文给出模型的误差分析和优化建议，首先基于 **ephem 库**对于本文计算的太阳相关角进行误差分析，在问题一的结果上计算误差率，发现均小于 **0.05**，证明本文的相关角度计算效果良好，接着本文考虑了**晴雨天分析**，构建**晴雨天影响因子**，对第二题求解发现可以使得角度更加偏向实际。最后本文讨论了大气衰减系数的调整，认为大气衰减系数可以结合云层厚度，湿度，风速等考虑。

关键字：　MPSO 算法　多目标规划　NSGA-II 算法　衰减系数　太阳能光伏板

目录

一、 问题重述

1.1 问题背景

改革开放以来，国家致力于推动绿色能源的发展，以减少对传统化石能源的依赖，降低碳排放，应对气候变化，保护环境，太阳能电池板作为绿色能源的重要组成部分，正是能源研究的热点。太阳能路灯由太阳能电池板组件部分（包括支架）、LED 灯头、控制箱（包含控制器、蓄电池）、市电辅助器和灯杆五部分共同组成。太阳能电池板通过支架固定在灯杆上端。太阳能电池板也称为光伏板，它利用光伏效应接收太阳辐射能并转化为电能输出，通过充放电控制器存储在蓄电池中。太阳能辐射由直射辐射和散射辐射组成，其中直射辐射对太阳能资源的利用具有重要影响。

1.2 题目信息

1. 已知 sheet1 中 2025 年 5 月 23 日从 6 点到 19 点的太阳地面直射辐射强度

2. 已知 sheet2 中 2025 年每个月平均的外太空太阳辐射强度

3. 已知大气层对太阳能直射辐射的衰减变化量与其辐射强度、所穿过的大气层厚度成正比

4. 通常地球表面大气层厚度按 1000 公里计算，大气层可视为包裹地球的球壳。

5. 太阳光到达大气层外层上的平均太阳能辐射强度为 $1353W/m^2$。

1.3 待求解问题

问题一：建立一个太阳直射光伏板的模型，在固定光伏板太阳方位角朝南的情况下，对于 2025 年每个月 15 日分别计算水平倾角为 $20°$、$40°$、$60°$ 的情况下的最大太阳直射强度和太阳直射辐射总能量

问题二：假设光伏板受到的太阳直射辐射总能量最大时，可使路灯蓄电池储电量最大。求解使得光伏板在晴天条件下受到的太阳直射辐射日均总能量最大的光伏板朝向，此朝向包括光伏板方位角以及水平倾角。

问题三：综合考虑路灯蓄电池的储电效率高和储电量大这两个目标，设计出光伏板固定安装的最优朝向，并计算晴天条件下光伏板受到的太阳直射辐射日均总能量和太阳直射辐射（上午大于 $150W/m^2$ 、下午大于 $100W/m^2$ ）时长

二、 问题分析

2.1 问题一

问题一已知当地经纬度为北纬 30°35′，东经 114°19′，要求在光伏板固定向南的朝向情况下，调整水平倾角，分别求出不同角度的光伏板接收到的日太阳辐射能量总量和最大太阳辐射量。首先，我们利用附件 sheet1 中的数据，考虑光线经过不同的大气厚度，对于大气衰减系数关于时间进行最小二乘拟合。之后我们需要构建计算出在三维空间中太阳相对于光伏板的位置得到太阳光的三维向量，还需要构建出光伏板这个平面的三维模型。最后计算出每个时间点太阳在不同方向角情况下，经过大气衰减后到达地面的辐射，通过向量操作后得到垂直于光伏板平面的太阳辐射强度。得到每个时间点后的光伏板接受太阳辐射强度之后，将离散的点进行插值和拟合，最后对于时间进行积分和乘以面积的操作得到一天的总能量。

2.2 问题二

问题二要求在给定天气为晴天的条件下，求得合适光伏板方位角和水平倾角，使其接收的太阳直射辐射日均总能量最大。在同月中每日的太阳辐射强度近似不变的假设下，可以采用每月 15 日的太阳辐射数据代表整个月的太阳辐射数据，依据附件 sheet2 所给数据我们可以得到每月的大气层表层的太阳辐射强度，结合拟合出的地球表层的辐射强度函数，可以得到每月 15 日在该城区地表辐射强度随时间变化的函数。然后通过问题一中建立的空间三维复合关系模型求得射向地表的光线向量和光伏板的法线向量，从而可以得到每月 15 日光伏板接收辐射强度一天内随时间变化的函数，将其积分和平均处理后即可表示每月 15 日的辐射日均总能量，以此作为目标函数。然后将光伏板方位角和水平倾角作为决策变量，构建禁忌搜索结合变异的粒子群优化模型求解光伏板最优的方位角和水平倾角。

2.3 问题三

问题三要求在考虑路灯蓄电池的储电效率高和储电量大两个目标的情况下，调整光伏板的方位角和水平倾角。关于储电效率高这一目标，我们认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，这样可以使路灯蓄电池的储电效率更高。对于储电量大这个目标，我们认为光伏板收到的太阳直射辐射越大则储电量越大。

由于 sheet2 数据为 2025 年不同月份平均外太空太阳辐射强度，假设某个月份中不同天的太阳辐射基本一致，则使用每月 15 日的太阳辐射强度数据作为本月的代表。假设每个月日期数为 30 日。我们从全年的角度出发，以太阳直射强度上午大于 $150W/m^2$、下午大于 $100W/m^2$ 的时间，和日均太阳直射辐射能量为目标。

我们构建出 NSGA-II 多目标遗传优化算法模型和网格搜索多目标优化模型这两个模型，分别求解出最优值后进行对比。最终得到光伏板方位角和水平倾角后，计算得出光伏板受到的太阳直射辐射日均总能量和太阳直射辐射时长（上午大于 $150W/m^2$ 、下午大于 $100W/m^2$ ）。

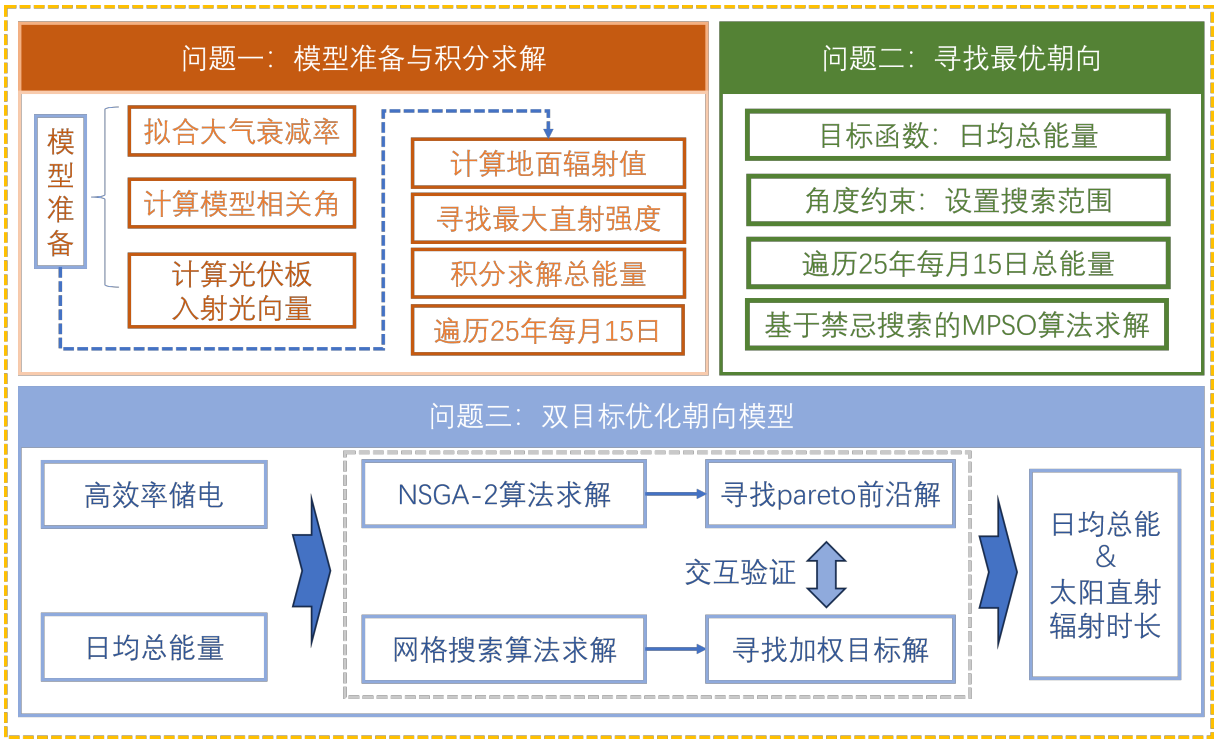


图 1　模型总流程图

三、 模型的假设

本文提出以下合理假设：

- 2025 年份的当地气候和过去几十年一致，无剧烈变化
- 假设 2025 年每个月平均日数为 30 日。
- 假设每个月 15 日中每天都是晴天。
- 假设大气衰减系数在晴天的条件下是恒定的。
- 假设光伏板只受到太阳直射辐射的影响。
- 假设不同月中每日的太阳辐射强度近似不变，可以用每月 15 日的太阳辐射数据代表整个月的太阳辐射数据。
- 如果一个光伏板朝向正南，那么它的方位角为零；如果一个光伏板朝向正东，那么它的方位角为 90°；如果一个光伏板朝向正西，那么它的方位角为 −90 °；水平仰角为光伏电池板平面与水平面的夹角。当太阳光线和光伏板的法线方向一致时，光伏板瞬时受到的太阳照射能量最大，否则会有余弦损失。

四、 符号说明

符号	意义	单位
θ	高度角	度
h	时角	度
ϕ	太阳方位角	度
δ	太阳赤纬	度
A	光伏板水平倾角	度
B	光伏板方位角	度
Φ	当地纬度	度
lng	当地经度	度
S	穿过大气层厚度	km
τ	衰减系数	$W/(m^2 \cdot km)$
d	某天相对于 1 月 1 日的日期数	–
mon	某天所在的月份	–
day	某天的日期	–
t_1	当天光照起始时间	时，分
t_2	当天光照起始时间	时，分
$hour$	某小时北京时间	时，分
ST	某小时当地时间	时，分
I	一天中任意时间点光伏板收到辐射强度	W/m^2
I_0	平均太阳能辐射强度	W/m^2
I_1	大气层表层辐射强度	W/m^2
I_2	经过大气衰减后的辐射强度	W/m^2
I_3	某小时直射在光伏板上太阳能辐射强度	W/m^2

五、 问题一的模型建立与求解

5.1 模型准备

5.1.1 赤纬角计算

$$\delta = 23.45^\circ \times \sin(\frac{360}{365} \times (d - 81)) \tag{1}$$

在 2025 年 5 月 23 日，$d = 142$，可以计算出当天的 $\delta = 20.34185^\circ$。

5.1.2 太阳高度角计算

$$\sin\theta = \sin\Phi \cdot \sin\delta + \cos\Phi \cdot \cos\delta \cdot \cos h \tag{2}$$

θ 是太阳高度角, Φ 是观察者的地理纬度, δ 是太阳的赤纬角, h 是太阳的时角。

5.1.3 太阳时角计算

$$ST = hour + \frac{(lng - 120^\circ)}{15^\circ} \tag{3}$$

$$h = 15\times(ST - 12) \tag{4}$$

其中 ST 为真太阳时，hour 是北京时间,lng 是当地经度，h 是时角。

5.2 模型建立

5.2.1 最小二乘法拟合大气衰减系数

1）大气表层到地球表层过程光线强度衰减 大气层对太阳能直射辐射的衰减变化量与其辐射强度、所穿过的大气层厚度成正比，其中 $\Delta_{attenuation}$ 为衰减变化量，τ 为衰减系数，S 为太阳光线穿过大气层的厚度，I_1 为大气层表层辐射强度:

$$\Delta_{attenuation} = S \times I_1 \times \tau \tag{5}$$

2）到地球表层的辐射量的时间变化误差通过对附件一中该城区 2023 年 5 月 23 日的地表辐射量随时间变化数据进行分析，发现一个基于时间的多次误差项，故将其作为影响辐射衰减的因素。

结合上述两个建立的关系式构建经过大气衰减后的太阳辐射随时间变化的函数，将数据集划分为训练集和测试集，采用最小二乘法拟合参数。衰减系数的拟合结果 τ 为 -8.6833×10^{-6}，拟合数据的整体效果如下表1所示，拟合的可视化见附件8。

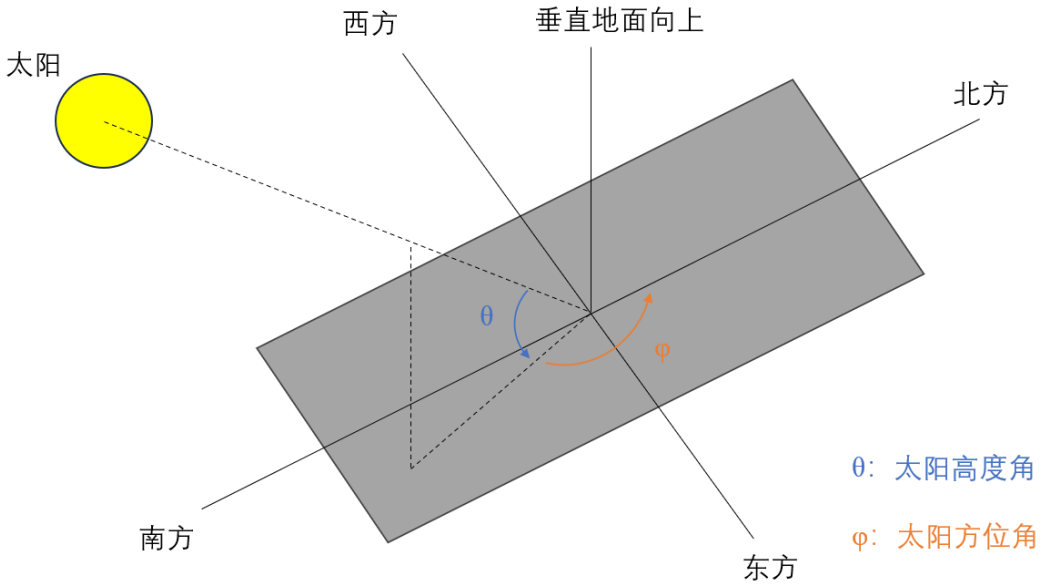
表 1　拟合效果

数值	训练集	测试集
决定系数 (R^2)	0.9919	0.9563
均方差 (MSE)	539.7114	3028.6632
均方根误差 (RMSE)	23.2317	55.0332

训练集决定系数为 0.9919，测试集决定系数为 0.9563，决定系数贴近 1 值，表明模型拟合效果良好。

5.2.2 太阳方位角计算

图 2　太阳方位角



$$\sin\phi = -\frac{\sin h \cdot \cos\delta}{\cos\theta} \tag{6}$$

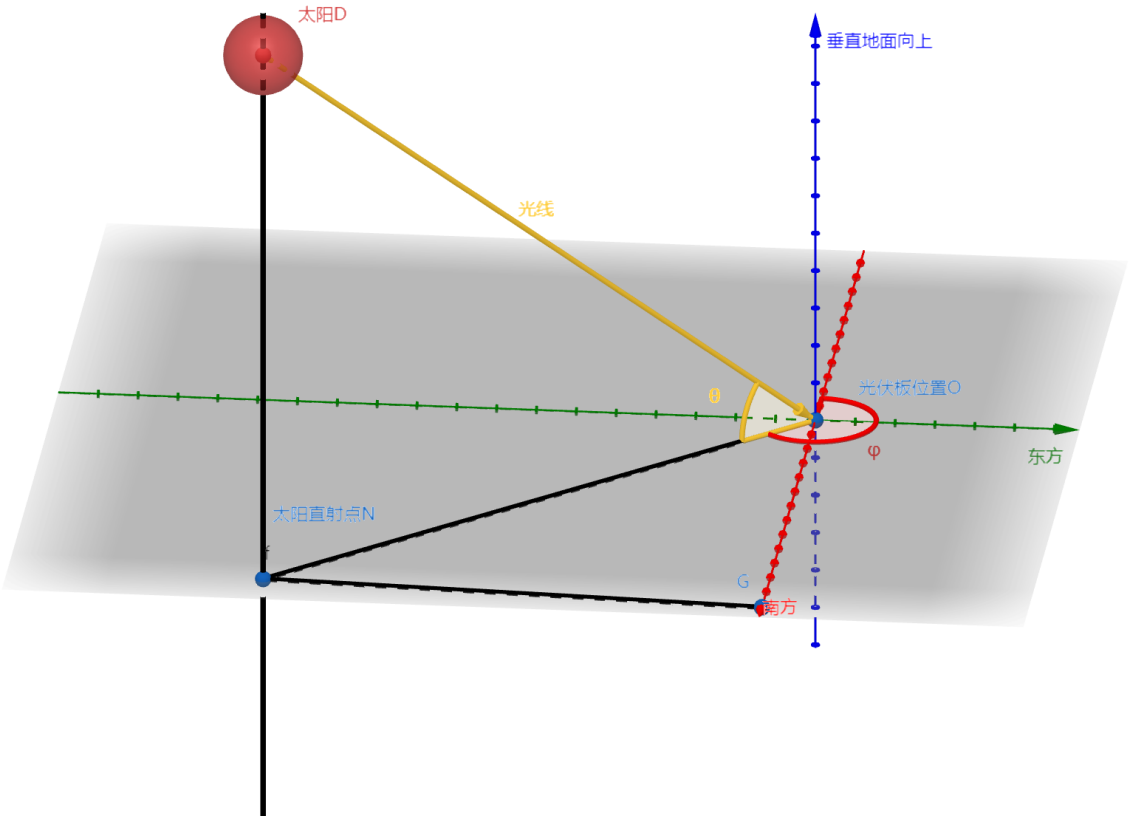
ϕ 是太阳的方位角，θ 是太阳高度角，h 是计算时间的时角，δ 是当时的太阳赤纬，

5.2.3 构建太阳光线三维坐标

在某一瞬时时刻，将太阳对光伏板光线简化为三维坐标里的直线，这个空间以光伏板位置作为坐标原点（由于光线是平行的，因此坐标原点的设置并不影响），以正南为

x 轴方向，以正东为 y 轴方向，垂直地平面向上为 z 轴方向，建立平面直角坐标系对其进行描述，如图3所示：

图 3　太阳光线向量三维立体图



图中 θ 为太阳高度角，ϕ 为太阳方位角，以此计算得光线向量 $\vec{a}$ 表达式：

$$\vec{a} = (\cos\phi, -\sin\phi, -\tan\theta) \tag{7}$$

5.2.4 构建光伏板三维坐标平面

为了方便对题目进行数学分析，故以正南为 x 轴方向，以正东为 y 轴方向，垂直地平面向上为 z 轴方向，建立平面直角坐标系，在直角坐标系中对转动的光伏板进行描述，如图4所示：

图 4　光伏板三维立体图

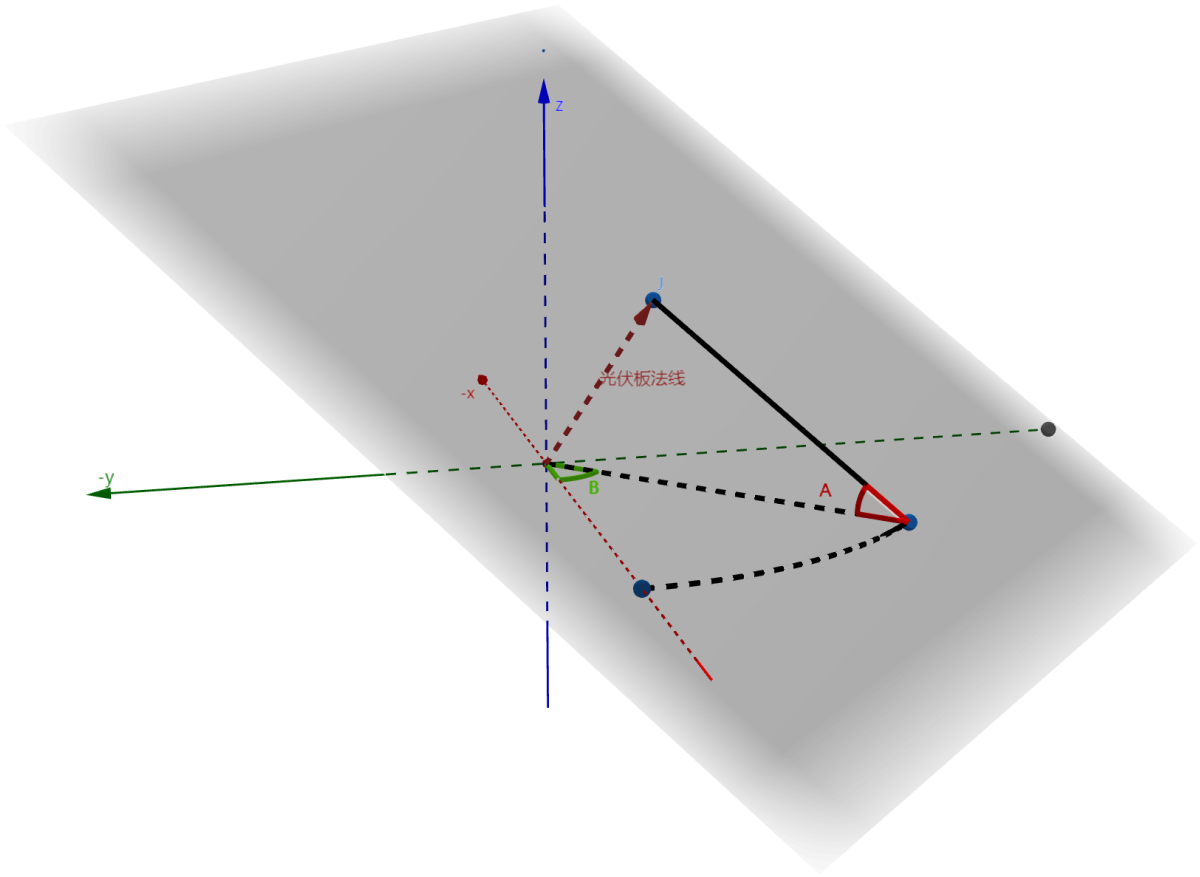
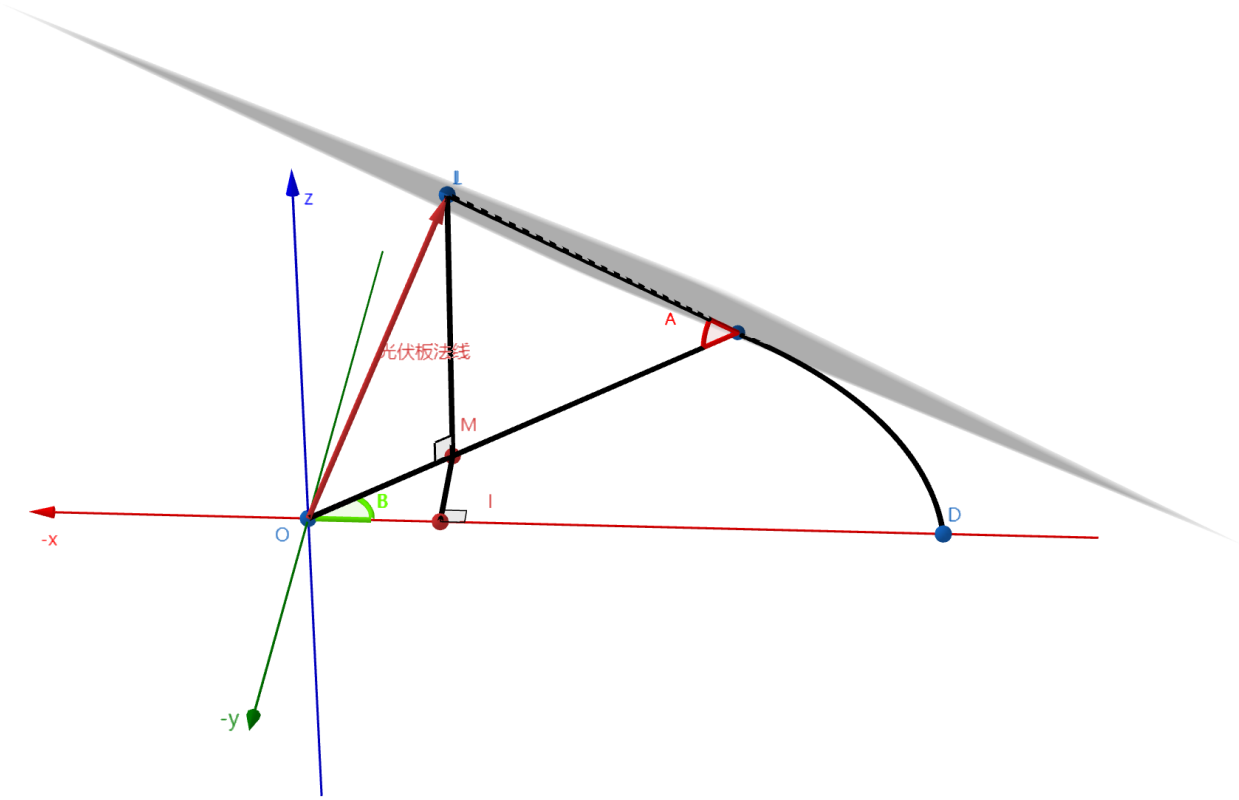


图4中黑色平面为光伏板，光伏板中心为点 L，A 角为光伏板的水平倾角，B 角为光伏板的方位角，红色箭头为光伏板中心的法线。然后对图4做辅助线对光伏板的法线进行计算，在图4的光伏板中心作对地面的垂线，交于 M，然后在点 M 作关于 x 轴的垂线交于点 I，得剖析图5。

图 5　三维剖析图



依据此数学三维模型进行推导，证明过程如下：

$$\begin{aligned}|\vec{x}|:|\vec{y}|:|\vec{z}| =& OI:IM:ML \\ =& OM\cos B:OM\sin B:\frac{OM}{\tan A} \\ =& \cos B:\sin B:\frac{1}{\tan A}\end{aligned}$$

故在三维坐标系里，在已知水平倾角 A 和朝向角 B 的条件下，光伏板法线可以表示为式8。

$$|\vec{x}|:|\vec{y}|:|\vec{z}| = \cos B:\sin B:\frac{1}{\tan A} \tag{8}$$

5.2.5 三维坐标系求解点到平面距离

- 假设两个三维向量分别为：$a = (x_1, y_1, z_1), b = (x_2, y_2, z_2)$。
- 向量 a 的模：$|a| = \sqrt{(x_1^2 + y_1^2 + z_1^2)}$。
- 向量 b 的模：$|b| = \sqrt{(x_2^2 + y_2^2 + z_2^2)}$。
- 两个向量的点乘：$a \cdot b = (x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2)$。
- 设两个向量的夹角为 w，则有：$\cos w = \frac{(x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2)}{\sqrt{(_1^2 + y_1^2 + z_1^2)x} \times \sqrt{(x_2^2 + y_2^2 + z_2^2)}}$

假设向量到平面的垂直分量为 $distance$, 假设向量的模为 $module$, 该向量与平面的法向量之间的夹角为 w

$$distance = module \times |\cos w| \tag{9}$$

5.2.6 光伏板接收辐射计算

针对 2025 年 5 月 23 日，在固定正南方朝向 $B = 0°$，调整水平倾角 A 分别为 $20°, 40°, 60°$ 的情况下

我们先求解某个小时光伏板接收到的太阳辐射 I_3，通过计算太阳高度角 θ，太阳时角 h, 太阳方位角 ϕ, 太阳的赤纬角 β，我们以当地作为原点，水平面作为 xy 面，可以构建出一个三维空间坐标系，分别得到太阳光线的向量方向 $(\cos\phi, -\sin\phi, -\tan\theta)$ 以及光伏板的平面法向量为 $(\cos B : \sin B : \frac{1}{\tan A})$，我们将经过大气衰减后的太阳辐射强度 I_2 作为太阳光线向量的模，利用公式9，将其与光伏板的平面法向量计算得到 I_2 在垂直光伏板方向的分量大小 I_3

计算出 2025 年 5 月 23 日中每个时间点的太阳辐射 $(I_3)_{t_1}, (I_3)_{t_1+1}, \ldots, (I_3)_{t_2}$，我们将其中最大的值列为最大太阳直射强度。

得到 I_3 进行插值和拟合得到当天每一时刻地球表层辐射强度 $I = f(t) \quad t \in t_1 \to t_2$，通过与光伏板面积相乘以及对时间进行积分可以得到当天光伏板接收到的太阳辐射总能量

5.3 结果分析

使用上述模型，在 2025 年的每个月份的 15 日进行模型求解，每次光伏板的朝向为东西方向不变，水平倾角分别为 20 度，40 度，60 度，得到的结果单日的太阳直射幅度最大值和太阳直射总能量如下表2所示。

表 2　太阳直射辐射数据

月份	20°C	40°C	60°C
1	446.35	391.02	288.53
2	532.06	466.11	343.94
3	638.03	558.94	412.44
4	749.36	656.47	484.40
5	825.64	723.30	533.71
6	847.17	742.16	547.63
7	826.16	723.75	534.04
8	762.68	668.14	493.01
9	659.44	577.70	426.28
10	549.37	481.28	355.13
11	456.12	399.58	294.84
12	416.37	364.75	269.15

表 3　太阳直射辐射总能量数据

月份	20°C	40°C	60°C
1	5128.84	4354.48	3058.14
2	6181.86	5232.37	3664.60
3	7442.43	6282.91	4390.66
4	8673.48	7308.20	5099.79
5	9425.97	7932.91	5531.14
6	9591.24	8068.03	5623.35
7	9366.53	7882.45	5496.17
8	8746.95	7369.39	5142.94
9	7640.74	6448.93	4506.65
10	6363.38	5384.57	3770.78
11	5234.19	4443.10	3120.09
12	4748.42	4037.67	2839.68

六、 问题二的模型建立与求解

6.1 模型准备

6.1.1 变异粒子群算法 [4]

传统的粒子群算法（PSO）通过一群微粒的协作和个体之间的信息交流来寻找最优解。在一个 D 维的空间中进行目标搜索，一个粒子群包含 n 个微粒，每个微粒是 D 维的向量，每个微粒在这个搜索空间中的位置为 $x_i = (x_{i1}, x_{i2}, ..., x_{iD}), i = 1, 2, ..., n$。微粒在空间中的位置代表着目标优化问题中的一个解，将该解代入**适应度函数**中后可以得到适应度的值，每个微粒适应度值的大小对应着该微粒所代表解对于目标函数的优劣程度；第 i 个微粒的飞行速度可以用一个 D 维的向量进行描述，记为 $v_i = (v_{i1}, v_{i2}, ..., v_{iD})$；第 i 个微粒在空间搜索过程中会记录下自己搜索过的具有最好适应度的位置，这个位置也是个体历史所搜索到的最好的解，记为 $p_i = (p_{i1}, p_{i2}, ..., p_{iD})$，整个粒子群在搜索过程中具有最好适应度的解称为全局历史最优解，记为 $p_g = (p_{g1}, p_{g2}, ..., p_{gD})$，粒子群的迭代搜索方程可以描述为：

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)(p_{ij}(t) - x_{ij}(t)) + c_2 r_2(t)(p_{gj}(t) - x_{ij}(t)) \tag{10}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t) \tag{11}$$

其中: 下标 j 表示微粒的第 j 维，下标 i 表示微粒 i，t 表示第 t 代，c_1，c_2 为加速常量，通常在 (0,2) 间取值，$r_1 \sim$ U(0,1)，$r_2 \sim$ U(0,1) 为两个相互独立的随机函数。从上述方程中可以知道，c1 常量会调整微粒向自身最优解靠近的步长，c2 常量会调整微粒向全局最优解靠近的步长。通过得到合适的 c1 和 c2 可以**加快收敛和避免陷入局部最优解**。

变异粒子群算法会在传统粒子群算法的基础上会进行自适应惯性权重 (Adaptive Inertia, AI) 和自适应种群规模 (Adaptive Population, AP) 这两个改变。

1）自适应惯性权重 (Adaptive Inertia, AI):

在传统 pso 算法中，惯性权重通常是一个固定的常数，它平衡了粒子的历史速度和局部/全局最优解之间的权衡关系。然而，在不同阶段或不同的问题中，固定的惯性权重可能无法取得最佳的优化效果。通过 AI 可以让传统算法中的惯性权重随着搜索状态发生改变, 因此改进后的算法具有**更高的搜索效率和更快的收敛速度**。

2）自适应种群规模 (Adaptive Population, AP):

传统算法在搜索后期种群多样性会减少从而不利于查找全局最优解，而自适应种群规模使传统算法的种群规模也随着搜索状态发生改变, 使得搜索过程**保持良好的种群多样性**，所以可以提高算法搜索过程中的种群多样性, 并有助于算法跳出可能的局部最优。

6.1.2 禁忌搜索算法 [3]

禁忌搜索算法是一种启发式算法，可以用于解决各种优化问题。在最优解的搜索过程中，经常会陷入局部最优解的陷阱。**禁忌搜索算法**首先会随机生成一个初始解作为当前解，然后在当前解的邻域中搜索若干个解，并选择其中的最佳解作为新的当前解，并且在搜索过程中会使用禁忌表记录已搜索的局部最优解的历史信息，以此防止算法重复选择已经探索过的局部最优解。如果一个解被加入禁忌表，算法在一定的禁忌期内不会再次选择这个解作为当前解。通过不断**更新和维护禁忌表**，算法能够**更广泛地探索搜索空间**，有助于找到更优的解决方案。

用禁忌搜索算法求解全局优化问题时，其求解步骤可以如下所示：

第一步：选定一个初始解 x^{now}; 令禁忌表 $H = \phi$;

第二步：若满足终止准则，转第四步；否则，在 x^{now} 的邻域 $N(x^{now})$ 中选出满足禁忌要求的候选集 $C - N(x^{now})$, 转第三步；

第三步：在 $C - N(x^{now})$ 中选一个评价值最好的解 x^{best}, 令 $x^{now} = x^{best}$, 更新禁忌表 H，转第二步；

第四步：得到最优解结果，终止算法。

6.2 模型建立

6.2.1 全局优化

在算法中，可以将每个优化问题的潜在解看作是 D 维搜索空间上的一个点，即假定没有体积和质量的“粒子”，并有被目标函数所决定适应度及其相应的位置和速度，然后粒子根据个体和群体的历史最优解进行**动态调整**。

step1：选择决策变量：

在固定光伏板的要求下，光伏板的水平倾角 A 和方位角 B 都可能影响每小时接受的太阳辐射强度，所以选择 A，B 作为决策变量。故粒子群算法搜索空间维度确定为 2 维空间，每个粒子是由 A，B 组成的 2 维向量，由 A，B 共同确定粒子在搜索空间中的位置。

step2：建立约束条件：

考虑到光伏板所处的地理位置，设定水平倾角 A 的范围为 0∼70 度，方位角 B 的范围为-40∼40 度。以此**缩小算法搜索的空间大小**，以此**提高算法执行的效率**。

$$\text{s.t.} \quad \begin{cases} 0^\circ \leq A \leq 70^\circ \\ -40^\circ \leq B \leq 40^\circ \end{cases}$$

step3：建立目标函数：

针对题目要求光伏板在晴天条件下受到的太阳直射辐射日均总能量尽可能的大，可以建立以太阳直射辐射日均总能量的目标函数作为为粒子群适应度函数，以此对应每次粒子搜索解的优劣程度，使得粒子在迭代中不断查找全局最优解。

以第一问中准备的**三维复合关系模型**作为基础，可以的得到地表光线向量与太阳高度角 θ 和太阳方位角 ϕ 的关系如式7所示，以及光伏板法线与光伏板水平倾角 A 和光伏板方位角 B 的关系如式8所示。通过计算可以得到地表太阳光线向量 $\vec{a}$ 垂直入射到光伏板的**光线分量** $\vec{a_0}$。同时通过第一问模型中由**最小二乘法**所拟合后的函数可推导求得某一天每一时刻地球表层辐射强度 $I = f(t)$ 如式12，其中 t 为当天的时间点，c_1、c_2、c_3 为常数项，S 为太阳光线穿过大气层的距离，即太阳光线到大气层表层入射点与地球表面入射点之间的直线距离，I_1 为当天 t 时刻大气层表层光照辐射强度，可由附件 sheet2 得,τ 为衰减系数，假设一天内大气层表层光照辐射 I_1 强度保持不变，可做如下推导：

$$\Delta_{attenuation} = S \times I_1 \times \tau$$

$$I(t) = c_1 t + c_2 t^2 + c_3 t^3 + \Delta_{attenuation}$$

由此可求得每小时光伏板收到太阳直射辐射强度结果为：

$$I(t) = c_1 t + c_2 t^2 + c_3 t^3 + S \times I_1 \times \tau \tag{12}$$

假设射向光伏板的光线全被转化，可得一天**光伏板接收总辐射能量** E 的表达式为式13所示，并将该式作为目标函数。

$$E = \int_{t_1}^{t_2} \frac{|\vec{a_0}|}{|\vec{a}|} I(t) \times 1m^2 \tag{13}$$

step4：参数设置：

- 迭代次数：100 次
- 搜索空间设置：水平倾角 0 到 70 度，东西方向-40 度到 40 度
- 禁忌搜索禁忌长度：5

6.2.2 禁忌搜索结合变异粒子群优化算法

在变异粒子群优化算法基础上，通过禁忌表记录粒子群搜索过程中的已经搜索到的局部最优解信息，在不断迭代搜索中会**更新和维护禁忌表**，每一个被记录的局部最优解信息会在一定的禁忌期限内无法被变异粒子群算法选择作为当前解，通过这种方式来避免变异粒子群算法陷入局部最优解，有助于得到全局最优解。

6.3 模型解答

基于禁忌搜索的 mpso 算法计算出来最优的向南倾角是 24.1474 度，最优的东西方向倾角是向西边旋转 2.8694 度，基于这个光伏板的朝向，遍历 12 月份的 15 日得到共计 12 天的总能量为 87677.5258W，即最终的晴天条件下，得到的日均太阳总能量为 7306W。

6.4 结论分析

小组调研了相关文献说明（这里补充论文）最优的向南倾角是 24.1474 度，接近该地区的维度，于是但是没有达到该地区的维度原因有：

- 只考虑了太阳直射的影响，使得夏天时模型的权重变高，导致模型求解过程会最大幅度的满足夏天时的最优解。
- 只考虑晴天情况，夏天雨天多，冬天雨天少，进而导致本题中相等的大气衰减系数不符合真正的大气衰减系数，导致模型求解过程会最大幅度的满足夏天时的最优解。

最优的东西方向倾角是是向西边旋转 2.8694 度，实际上是转动幅度较小，从一定程度上可以认为是向正南方向放置，我们认为这个结果是可靠的，也符合地理实际，即北半球的地区将太阳能板向正南方放置可以获得最大的太阳能量积累。

七、 问题三的模型建立与求解

7.1 变量定义

相关变量定义

定义变量 B 为光伏板方位角，定义变量 A 为光伏板与水平倾角, 某月份每日中每个小时的日照辐射为 I_3。该月份中每日的光伏板吸收能量为 E_1, 该月份光伏板吸收的太阳辐射能量为 E_2，2025 年全年吸收的太阳辐射能量为 E_3,K 为光伏板接收辐射强度与光伏板吸收能量之间的转换率。因为光电转化率受到温度，太阳辐射强度，散热系数等多方面影响，所以 K 为未知量，考虑到 I_3 随 E_0 增大而增大，因此在多目标优化中将 K 作为一个常量参与计算。

在得到每个小时的太阳辐射强度数据 I_3 后，我们对其进行插值拟合后得到直射在光伏板上的太阳辐射强度 $I(t)$, 其中 I 是一天中任意时刻的直射在光伏板上的太阳辐射强度

$$E_1 = \int_{t=t_1}^{t=t_2} ((I(t)) \times K) \times 1m^2 \, dt \tag{14}$$

在下列公式中 j 仅仅代表遍历不同的序列。

$$\begin{cases} E_2 = \sum_{j=1}^{j=30} (E_1)_j \\ E_3 = \sum_{j=1}^{j=12} (E_2)_j \end{cases} \tag{15}$$

将一天从 t_1 日出到 t_2 日落划分为多个时间段, 总共有 $1 \to \max(j)$ 个时间段，因此可以定义第 i 天中第 j 个时间段如下

$$C_{ij} = \begin{cases} 1 & , (I_3)_j > 150 \text{ W/m}^2 \text{ and j 处于上午时间段} \\ 1 & , (I_3)_j > 100 \text{ W/m}^2 \text{ and j 处于下午时间段} \\ 0 & , else \end{cases}$$

决策变量

在目标函数中 E_3 受到输入变量 A 和 B 的影响，A 是光伏板水平倾角，B 是光伏板朝向角。

7.2 目标函数及约束条件

$$
\begin{aligned}
&\min \quad f_1 = -E_3 \\
&\min \quad f_2 = -\sum_{i=1}^{360} \sum_{j=1}^{\max(j)} C_{ij} \\
&\text{s.t.} \quad \begin{cases} 0^\circ \leq A \leq 90^\circ \\ -90^\circ \leq B \leq 90^\circ \end{cases}
\end{aligned} \tag{16}
$$

7.3 NSGA-II 多目标遗传算法模型 [2]

7.3.1 模型定义

本文采用多目标遗传算法 NSGA-II。相对于传统的多目标遗传算法，NSGA-II 使用了**支配排序技术和拥挤度距离排序算法**。该算法通过选择，交叉，变异的方式获得下一代。在具体实现中，本文选择的交叉算子是“**模拟二进制交叉（SBX）**”，选择的变异算子是**“多项式变异”(PM)**。

SBX（Simulated Binary Crossover）算子表示为：

$$
\text{SBX}(x_1, x_2) = \begin{cases} x_1' = 0.5\left((1+\beta)x_1 + (1-\beta)x_2\right) \\ x_2' = 0.5\left((1-\beta)x_1 + (1+\beta)x_2\right) \end{cases}
$$

其中，x_1 和 x_2 是父代染色体，β 是分布指数。

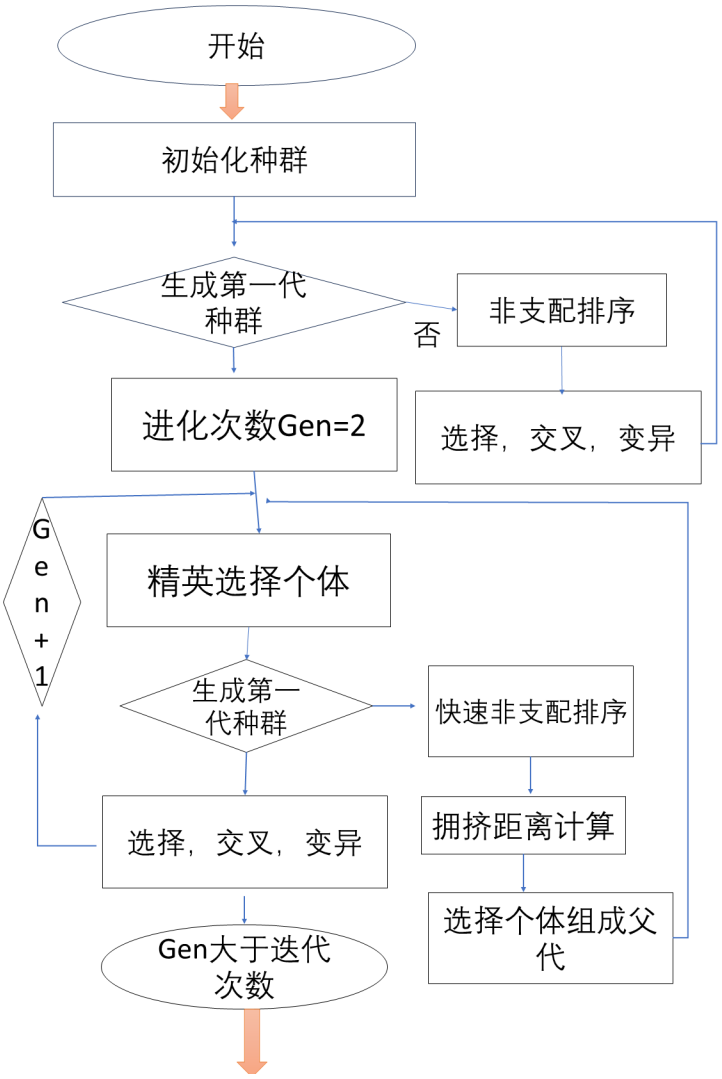
多项式变异 (PM) 的表示为：

$$
x_j = x_j + \Delta_j \tag{17}
$$

$$
\Delta_j = \begin{cases} (2\mu_i)^{\frac{1}{\eta}+1} & \text{if } \mu_i < 0.5 \\ 1 - [2(1-\mu_i)]^{\frac{1}{\eta}+1} & \text{if } \mu_i \geq 0.5 \end{cases} \tag{18}
$$

μ_i 是满足 (0,1) 均匀分布的随机数，η 是变异分布参数

图 6　NSGA-II 流程图



7.3.2 模型求解

我们构建出 NSGA-II 多目标遗传算法，并且定义**迭代次数为 100 次，种群大小为 50**，进行不断地迭代，选择，交叉，变异之后最终在 Pareto 前沿解中确定了最优值。考虑全年情况下，认为每个月份太阳辐射强度能够通过每个月的 15 日情况进行模拟，设定每个月长度为 30 日，在 NSGA-II 多目标遗传算法中我们得到光伏板的最优朝向是**水平倾角 31.1970 度，向西边偏 1.1184 度（即光伏板方位角为-1.1183 度）**，

7.4 网格搜索多目标优化

采用网格搜索的方法，在给定的朝向范围内对光伏板朝向进行搜索。具体步骤如下：

- 确定光伏板朝向的搜索范围，光伏板方位角跨度 180 度（$-90° \rightarrow 90°$），水平倾角跨

度 90 度（$0° \to 90°$）范围内的所有可能朝向。将步长设置为 1 度，即搜索 180*90 个解。

- 遍历所有可能的朝向，并计算每个朝向下在晴天条件下的太阳直射辐射日均总能量和太阳直射辐射时长。
- 综合考虑两个目标，选择使得太阳直射辐射时长和日均总能量都较大的朝向作为最优朝向。
- 输出最优朝向，并且和 NSGA-II 多目标遗传算法的结构进行对比说明

7.5 模型结果及分析

我们使用上述两个模型，在 12 个月份中的 15 日的情况进行模拟，在 NSGA-II 多目标遗传算法中我们得到光伏板的最优朝向是水平倾角 31.1970 度，向西边偏 1.1184 度（即光伏板方位角为-1.1184 度），同时在网格多目标搜索算法中，我们得到模型的最优解是向南方偏 30 度（即水平倾角为 0 度），东西方向不发生偏转（即光伏板方位角为 0 度），我们在两个模型中得到了相似的结果，同时查阅相关资料，我们认为我们的结果具有合理性。

我们使用 NSGA-II 多目标遗传算法中得到的最优朝向，水平倾角为 31.1970 度，向西边偏 1.1184 度（即光伏板方位角为-1.1184 度），在 25 年的每月 15 号中进行模型，得到每日光伏板受到太阳直射强度上午大于 150 W/m2、下午大于 $100W/m^2$ 的时间的平均长度是 **9.9735**h，日均的太阳直射强度为：**9534.2903**W。

八、 优缺点分析及模型优化

8.1 ephem 分析

在计算当地某小时的太阳高度角，太阳赤纬角，太阳方向角时我们采用了原始的公式进行近似求解，因此与精准的角度有部分误差。为验证这部分误差在可接受范围内，我们采用 python 中的包 ephem 作为精确计算的工具作对比。ephem 库内置了大量天体的基本数据，例如行星、恒星等的轨道参数、位置和运动信息。同时使用球面三角法等算法计算出天体在天球上的位置。针对一些影响天体位置的因素，如大气折射、地球章动等，ephem 库可能会进行一些额外的修正计算，以提高计算精度。因此调用 ephem 计算出来的角度比原始公式得出的角度更为精准。

在问题一中，我们使用我们的推导的角度和 ephem 库计算出来的精确角度做差之后除以精确角度，具体结果如下表4：

我们发现残差率均小于 0.05，证明我们推导的角度在可接受范围内。

表 4　推导角度和精确角度计算出来的第一题结果的残差率

	20 度	40 度	60 度
残差率	0.0317	0.0313	0.049

8.2 晴天分布分析

考虑到当地天气并不是一个月份中每天都是晴天，而是存在雨天，多云，晴天等多种天气，因此我们简化的假设两种天气，晴天和非晴天。其中晴天可以认为大气衰减系数保持恒定，雨天可以认为当地此时的太阳辐射强度为 0。

通过经纬度可以锁定当地位置为武汉市洪山区，额外调查可以发现武汉市每个月的晴天和雨天天数如下图

图 7　武汉每日降水机率

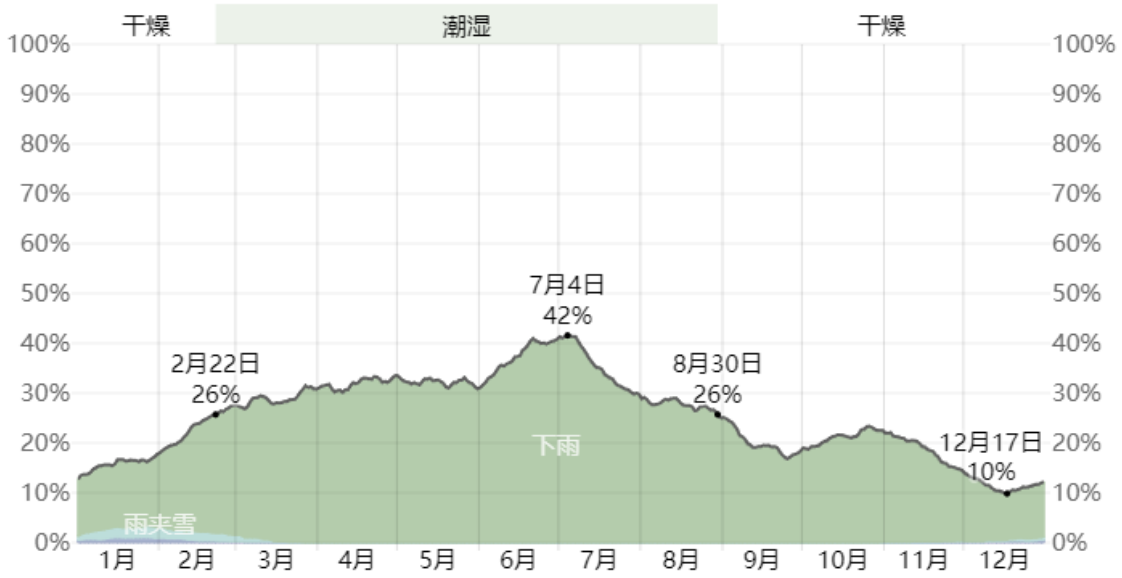


表 5　每月天气情况

天数	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
下雨	4.0 天	6.1 天	8.8 天	9.6 天	10.0 天	11.2 天	11.0 天	8.6 天	6.0 天	6.6 天	5.6 天	3.4 天
雨夹雪	0.6 天	0.5 天	0.1 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.1 天
下雪	0.3 天	0.1 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.0 天	0.1 天
任何	4.9 天	6.7 天	9.0 天	9.6 天	10.0 天	11.2 天	11.0 天	8.6 天	6.0 天	6.6 天	5.6 天	3.6 天

上述图片和表格资料均来源于 Weather Spark，网址地址为 [1]

我们假设任何非晴天天气，如下雨，雨夹雪，下雪等为一类，在这些天气下当地的太阳辐射强度为 0。例如在 5 月份，总共有 10 天非晴天天气，假设一个月平均有 30 天，则 5 月份有 20 天的晴天天数。

我们对于第二问进行改良实验，将模型每个月得到的能量乘以修正系数 Idx，修正系数 Idx 如下

$$Idx = \frac{D_{permonth} - D_{abnormal}}{D_{permonth}} \tag{19}$$

其中 $D_{permonth}$ 表示月份的天数，$D_{abnormal}$ 表示是一个月中非晴天情况的天数。

加上修正系数之后我们的模型在第二问中的目标函数得到的单日太阳辐射总能量上乘上修正系数。和第二题相同的实验条件下跑出的结果为：向西边旋转 1.3113 度（即光伏板方位角为-1.3113 度），水平倾角为 33.3380 度，题目中地点的维度小于这个答案的水平倾角，同时大于全部为晴天状态下的水平倾角，这说明实际上的最优纬度在我们两次极端模拟的答案之间（不考虑雨天和非晴天太阳直射强度为 0），可以验证我们的模型对于纬度的搜索符合实际。

8.3 大气衰减系数分析

之前问题中我们假设大气衰减系数 τ 是一个恒定的常数, 实际上大气衰减系数会因为天气，云层厚度，湿度，风速等原因，而不断变化。同时我们假设经过大气过程中太阳辐射的衰减量为 $\Delta_{attenuation} = S \times I_1 \times \tau$, 认为太阳辐射衰减量与外太阳太阳辐射强度、所穿过的大气层厚度成正比，实际上由于大气中成分复杂不能作为一个整体看待，因此实际上太阳辐射衰减量是一个复杂的表达式。

8.4 模型优缺点分析

我们的模型具有以下优点：

1. 我们的模型结果符合实际，水平倾角的取值在当地纬度相近。在模型改进方面，我们引入了对雨天的考虑。然而，当地纬度在两种极端条件下（不考虑雨天和雨天太阳直射强度置为 0）给出的最适合水平倾角之间存在差异，但仍需要确定具体的雨天系数。
2. 我们在问题二中建立了基于禁忌搜索的 MPSO 算法和在问题三种给出的 NSGA2 算法都具有良好的全局搜索能力，不会落入局部最优解，同时还使用网格搜索算法验证了这一点。
3. 我们在拟合大气衰减系数上给出了一个基于小时的残差项，很好地拟合了大气衰减系数，交叉验证地结果达到了 96%。

4. 我们找到了 python 库中 ephem 库，对于问题一中的结果进行了验证，发现残差率都小于 0.05，说明我们在具体太阳角度上计算效果良好。

我们的模型仍然存在一些不足之处：

1. 对于过高的太阳直射强度对储能效率的影响的约束条件简单。缺乏具体的方程来提供进一步的约束。
2. 模型搜索算法所需的时间开销较大。

参考文献

[1] Weather spark. https://zh.weatherspark.com/y/128408/%E4%B8%AD%E5%9B%BD%E3%80%81%E6%AD%A6%E6%B1%89%E7%9A%84%E5%85%A8%E5%B9%B4%E5%B9%B3%E5%9D%87%E5%A4%A9%E6%B0%94. Accessed: 2024-04-21.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation, 6(2):182–197, 2002.

[3] 张震, 魏鹏, 李玉峰, 兰巨龙, 徐萍, and 陈博. 改进粒子群联合禁忌搜索的特征选择算法. Journal on Communication/Tongxin Xuebao, 39(12), 2018.

[4] 陈金辉, 陈辰, and 董飚. 基于自适应策略的改进粒子群算法. 计算机仿真, 32(3):298–303, 2015.

附录 A 拟合过程图片

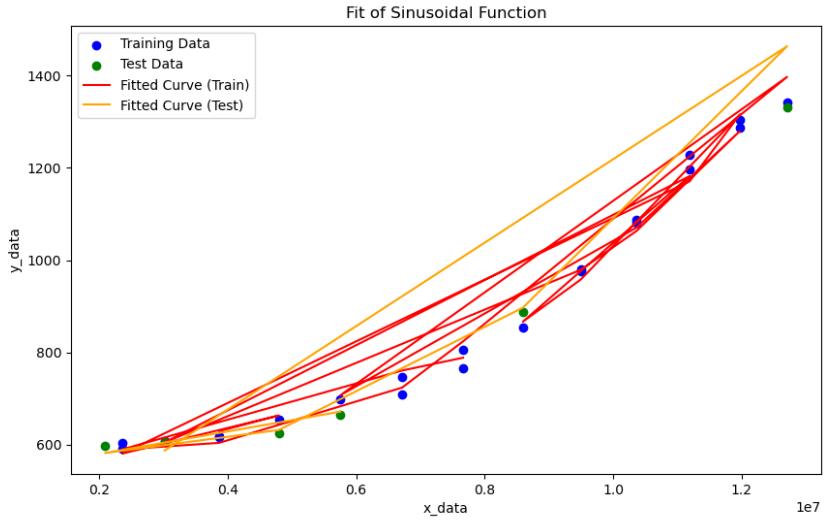


图 8　τ 的拟合可视化

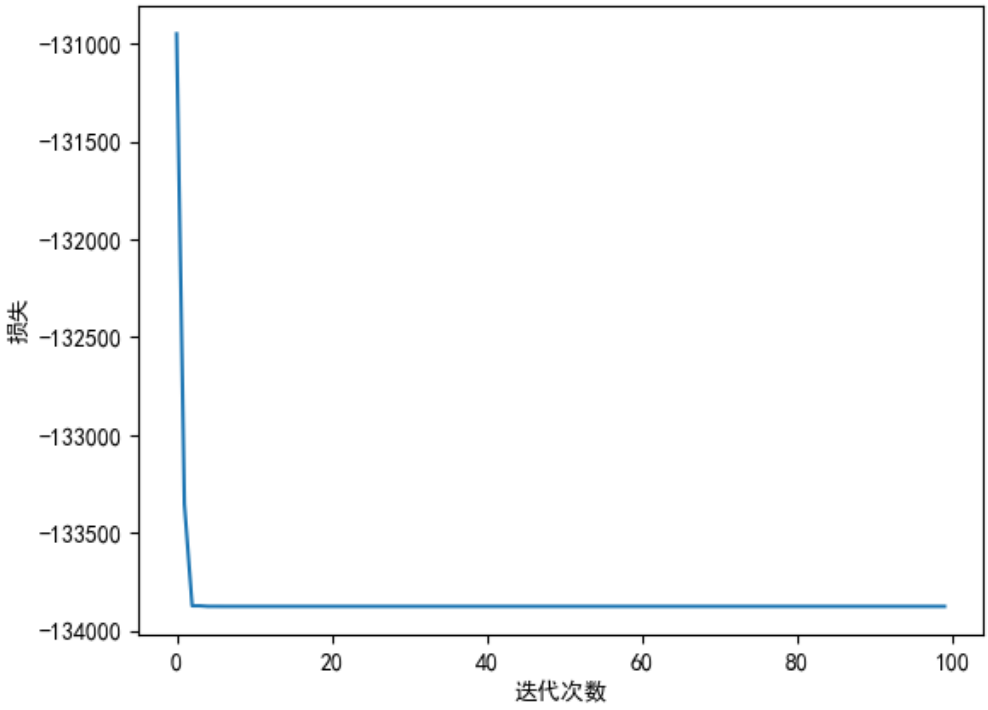


图 9　第二题中基于禁忌搜索的 mpso 收敛效果图，如图所示，结果收敛

表 6　文件名及描述

文件名	描述
test.py	增加上晴雨天的改进系数实现
main.py	用于生成第一题的 CSV 文件
getK.py	计算大气消磨系数
mpso.py	基于禁忌搜索的多目标粒子群优化算法的实现
nsga2.py	针对第三题的 NSGA-II 算法的实现
grid.py	针对第三题的网格搜索算法的实现

附录 B python Code

Listing 1: test.py

```
import numpy as np
from goal_for_pro2 import calculate_results
class Particle:
    def __init__(self, dim, min_values, max_values):
        self.position = np.random.uniform(min_values, max_values, dim)
        self.velocity = np.random.rand(dim)
        self.best_position = self.position.copy()
        self.best_score = float('inf')

def objective_function(x,y):
    alpha, beta = x, y
    sum = 0
    days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    abnormal_days = [4.9, 6.7, 9.0, 9.6, 10.0, 11.2, 11.0, 8.6, 6.0, 6.6, 5.6, 3.6]
    idx = []
    for i in range(12):
        idx.append((days_in_month[i] - abnormal_days[i]) / days_in_month[i])
    for i in range(1, 13):
        if i in range(5, 9):
            sum += idx[i-1]*calculate_results(2025, i, 15, alpha, beta)
        else:
            sum += idx[i-1]*calculate_results(2025, i, 15, alpha, beta)
    return -sum

import matplotlib.pyplot as plt
```

```
def pso(objective_function, dim, min_values, max_values, num_particles, num_iterations):
    particles = [Particle(dim, min_values, max_values) for _ in range(num_particles)]
    global_best_position = None
    global_best_score = float('inf')
    loss_history = []

    for _ in range(num_iterations):
        for particle in particles:
            score = objective_function(*particle.position)
            if score < particle.best_score:
                particle.best_position = particle.position.copy()
                particle.best_score = score

            if score < global_best_score:
                global_best_position = particle.position.copy()
                global_best_score = score

        for particle in particles:
            # 更新速度和位置
            inertia_weight = 0.7
            cognitive_weight = 1.5
            social_weight = 1.5
            particle.velocity = (inertia_weight * particle.velocity +
                                 cognitive_weight * np.random.rand(dim) * (particle.best_position -
                                     particle.position) +
                                 social_weight * np.random.rand(dim) * (global_best_position -
                                     particle.position))
            particle.position += particle.velocity
            # 边界检查
            particle.position = np.clip(particle.position, min_values, max_values)

        # 记录当前迭代的损失值
        loss_history.append(global_best_score)

    return global_best_position, global_best_score, loss_history

# 定义目标函数的参数范围
min_values = np.array([-90, -90]) # 参数的最小值
max_values = np.array([90, 90]) # 参数的最大值

# 调用 PSO 算法来最小化目标函数
best_position, best_score, loss_history = pso(objective_function, dim=2,
    min_values=min_values, max_values=max_values,
                                          num_particles=30, num_iterations=100)

print("最优参数:", best_position)
```

```
print("最优解:", best_score)

# 绘制损失函数的变化
plt.plot(loss_history)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss Function over Iterations')
plt.show()
```

Listing 2: main.py

```
import math
import numpy as np
import pandas as pd
from scipy.integrate import simps
from scipy.interpolate import interp1d
from matplotlib import pyplot as plt
from get_sun_altitude import get_sun_alti, calculate_sun_altitude
from get_left_warm import get_left_warm
from calculate_normal_vector import calculate_normal_vector
from get_sun_dir import get_sun_dir, calculate_sun_azimuth
import ephem

# 解决中文乱码
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["font.family"] = "sans-serif"
# 解决负号无法显示的问题
plt.rcParams['axes.unicode_minus'] = False

# 定义城市的纬度和经度
latitude = 30.5833 # 北纬30°35'
longitude = 114.3167 # 东经114°19'


def solar_altitude_angle(latitude, declination_angle, hour_angle):
    """
    计算太阳高度角
    :param latitude: 城市的纬度
    :param declination_angle: 太阳赤纬角
    :param hour_angle: 太阳时角
    :return: 太阳高度角
    """
    return math.asin(math.sin(math.radians(latitude)) * math.sin(declination_angle) +
                     math.cos(math.radians(latitude)) * math.cos(declination_angle) *
                         math.cos(hour_angle))
```

```python
def solar_declination_angle(day):
    """
    计算太阳赤纬角
    :param day: 一年中的第几天
    :return: 太阳赤纬角
    """
    return 23.45 * math.sin(math.radians(360 / 365 * (day - 81)))


def solar_hour_angle(hour):
    """
    计算太阳时角
    :param hour: 小时
    :return: 太阳时角
    """
    return math.radians(15 * (hour - 12))


def solar_radiation(solar_intensity, solar_altitude_angle):
    """
    计算太阳直射辐射能量
    :param solar_intensity: 太阳直射强度值 (W/m^2)
    :param solar_altitude_angle: 太阳高度角
    :return: 太阳直射辐射能量
    """
    return solar_intensity * math.sin(solar_altitude_angle)


def normal_vector(azimuth_angle, tilt_angle):
    """
    计算法向量
    :param azimuth_angle: 方位角
    :param tilt_angle: 倾斜角
    :return: 法向量
    """
    azimuth_rad = math.radians(180 - azimuth_angle)
    tilt_rad = math.radians(tilt_angle)

    return [math.cos(azimuth_rad) * math.cos(tilt_rad),
            math.sin(azimuth_rad) * math.cos(tilt_rad),
            math.sin(tilt_rad)]


def find_solar_declination(date):
    """
    查找太阳赤纬角
    :param date: 日期
```

```
    :return: 太阳赤纬角
    """
    obs = ephem.Observer()
    obs.date = date
    sun = ephem.Sun()
    sun.compute(obs)
    solar_declination = float(sun.dec)
    return solar_declination * (180 / math.pi)


def calculate_cosine_of_angle(normal_vector1, normal_vector2):
    """
    计算两个向量的夹角余弦值
    :param normal_vector1: 向量1
    :param normal_vector2: 向量2
    :return: 夹角余弦值
    """
    dot_product = sum(a * b for a, b in zip(normal_vector1, normal_vector2))
    length1 = math.sqrt(sum(a ** 2 for a in normal_vector1))
    length2 = math.sqrt(sum(b ** 2 for b in normal_vector2))
    cosine_of_angle = dot_product / (length1 * length2)
    return cosine_of_angle


def calculate_results(year, month, day, east_angle, south_angle):
    """
    计算结果
    :param year: 年份
    :param month: 月份
    :param day: 日份
    :param east_angle: 东方向角度
    :param south_angle: 南方向角度
    :return: 最大能量, 总能量
    """
    left_warm = get_left_warm(year, month, day)
    print("************")
    date = f"{year}/{month}/{day}"
    declination_angle = find_solar_declination(date)
    latitude = 30.5833
    max_energy = 0
    energy = []
    cos = []
    for index, hour in enumerate(range(12, 38)):
        hour_float = hour / 2
        hour_angle = solar_hour_angle(hour_float)
        altitude_angle = get_sun_alti(year, month, day, hour_float)
        altitude_angle = get_sun_alti(2023, 5, 23, hour_float)
```

```
        azimuth = get_sun_dir(year, month, day, hour_float)
        azimuth = get_sun_dir(2023, 5, 23, hour_float)
        band_vector = calculate_normal_vector(south_angle, east_angle)
        cosine_num = calculate_cosine_of_angle(band_vector, normal_vector(azimuth,
            altitude_angle))
        if cosine_num < 0:
            cosine_num = 0
        cos.append(cosine_num)
        now_warm = left_warm[index] * cosine_num
        energy.append(now_warm)
        if now_warm > max_energy:
            max_energy = now_warm

    print()
    print(month, east_angle, south_angle)
    print("最大的太阳直射辐射能量是: ", max_energy)
    print(cos)
    print(energy)

    time_interval = 0.5
    time = np.arange(len(energy))
    f = interp1d(time, energy, kind='cubic')
    time_new = np.linspace(time.min(), time.max(), 1000)
    energy_new = f(time_new)
    total_energy = simps(energy_new, dx=(time_new[1] - time_new[0]))

    print(east_angle, south_angle)
    print("总能量为:", total_energy)
    return max_energy, total_energy


max_energy_df = pd.DataFrame(columns=['month', 'Max Energy (20°C)', 'Max Energy (40°C)', 'Max
    Energy (60°C)',
                                      'Total Energy (20°C)', 'Total Energy (40°C)', 'Total Energy
                                          (60°C)'])
for i in range(1, 2):
    max_energy_list = []
    total_energy_list = []
    for temperature in [20, 40, 60]:
        print(1)
        max_energy, total_energy = calculate_results(2025, i, 15, 0, temperature)
        max_energy_list.append(max_energy)
        total_energy_list.append(total_energy)

    max_energy_df.loc[i-1] = [i] + max_energy_list + total_energy_list

max_energy_df.to_csv('max_energy_results.csv', index=False, encoding='gbk')
```

Listing 3: getK.py

```
import math
from get_sum_dir import find_solar_declination

def great_circle_distance(lat1, lon1, lat2, lon2, R1, R2):
    """
    计算两个地点之间的大圆距离
    :param lat1: 第一个地点的纬度（度）
    :param lon1: 第一个地点的经度（度）
    :param lat2: 第二个地点的纬度（度）
    :param lon2: 第二个地点的经度（度）
    :param R1: 第一个地点的半径（单位：任意长度单位）
    :param R2: 第二个地点的半径（单位：任意长度单位）
    :return: 两个地点之间的大圆距离（单位：与地球半径相同的长度单位）
    """
    # 将角度转换为弧度
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    # 计算经度差
    delta_lon = lon2_rad - lon1_rad

    # 使用大圆距离公式计算距离
    distance = math.sqrt(
        (R1 * math.cos(lat1_rad) * math.cos(lon1_rad) - R2 * math.cos(lat2_rad) *
            math.cos(lon2_rad)) ** 2 +
        (R1 * math.cos(lat1_rad) * math.sin(lon1_rad) - R2 * math.cos(lat2_rad) *
            math.sin(lon2_rad)) ** 2 +
        (R1 * math.sin(lat1_rad) - R2 * math.sin(lat2_rad)) ** 2)

    return distance

depth = 1000
R1 = 6371 # 地球半径，单位为千米
R2 = 6371 + depth # 地球半径，单位为千米

# 第一个地点的纬度和经度
lat1 = 30.35 # 地点1的纬度，单位为度
lon1 = 114.19 # 地点1的经度，单位为度

# 第二个地点的纬度和经度
# 计算直射维度
```

```
date = "2023/05/23"
# 计算太阳直射点
declination = find_solar_declination(date)

lat2 = declination # 地点2的维度，单位为度

# 生成lon2数组
lon2 = [lon1 + offset * 7.5 for offset in range(-13, 14)]
print(lon2)

distances = []
for lon in lon2:
    distance = great_circle_distance(lat1, lon1, lat2, lon, R1, R2)
    print("两个地点之间的距离为:", distance, "千米")
    distances.append(distance)
print(distances)

import pandas as pd
import numpy as np

# 读取xlsx文件
df = pd.read_excel('附件/附件.xlsx', sheet_name='Sheet1')

# 显示前几行数据
print(df.head())

left_warm = np.array(df["地面直射辐射量W/m2"].values)

E = 1353

# 计算出delta
delta_warm = E - left_warm

import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split

# 定义拟合函数
def sinusoidal_function(x, a, b, c, d, K):
    x_data, time = x
    return a * time**3 + b * time**2 + c * time + d + K * x_data

# 构造数据
x_data = np.array(distances) * E
```

```
time = np.linspace(6, 19, 27)
y_data = delta_warm

# 将 x_data 和 time 合并成一个二维数组
X = np.vstack((x_data, time)).T

# 将数据集分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y_data, test_size=0.2, random_state=42)

# 定义初始猜测值
initial_guess = [1, 1, 0, 1, 1] # 例如 [a, b, c, d, K]

# 使用 curve_fit 拟合训练集数据
popt, pcov = curve_fit(sinusoidal_function, (X_train[:, 0], X_train[:, 1]), y_train,
    p0=initial_guess)

# 计算拟合的函数值
y_fit_train = sinusoidal_function((X_train[:, 0], X_train[:, 1]), *popt)
y_fit_test = sinusoidal_function((X_test[:, 0], X_test[:, 1]), *popt)

# 绘制拟合图像
plt.figure(figsize=(10, 6))
plt.scatter(X_train[:, 0], y_train, color='blue', label='Training Data')
plt.scatter(X_test[:, 0], y_test, color='green', label='Test Data')
plt.plot(X_train[:, 0], y_fit_train, color='red', label='Fitted Curve (Train)')
plt.plot(X_test[:, 0], y_fit_test, color='orange', label='Fitted Curve (Test)')
plt.xlabel('x_data')
plt.ylabel('y_data')
plt.title('Fit of Sinusoidal Function')
plt.legend()
plt.savefig("result/K拟合.png")
plt.show()

# 计算训练集和测试集上的 R^2 值
r2_train = r2_score(y_train, y_fit_train)
r2_test = r2_score(y_test, y_fit_test)

# 计算训练集和测试集上的均方差 MSE
mse_train = mean_squared_error(y_train, y_fit_train)
mse_test = mean_squared_error(y_test, y_fit_test)

# 计算训练集和测试集上的均方根误差 RMSE
rmse_train = np.sqrt(mse_train)
rmse_test = np.sqrt(mse_test)

print("训练集 R^2 值:", r2_train)
print("测试集 R^2 值:", r2_test)
```

```
print("训练集均方差 (MSE):", mse_train)
print("测试集均方差 (MSE):", mse_test)
print("训练集均方根误差 (RMSE):", rmse_train)
print("测试集均方根误差 (RMSE):", rmse_test)
print("拟合的参数:", popt)
```

Listing 4: mpso.py

```
import numpy as np
from goal_for_pro2 import calculate_results # 假设这个模块包含你的 'calculate_results' 函数

# 定义目标函数
def objective_function(x):
    # 这个目标函数示例是基于Rosenbrock函数的
    alpha, beta = x
    sum = 0
    for i in range(1, 13):
        sum += calculate_results(2025, i, 15, alpha, beta)
    return sum

# 定义搜索空间范围
lb = [-5, -5] # 搜索空间下限
ub = [90, 90] # 搜索空间上限

# 禁忌搜索参数
tabu_tenure = 5 # 禁忌长度
tabu_list = [] # 禁忌列表

# 初始化粒子群
def initialize_swarm(n_particles, n_dimensions, lb, ub):
    return np.random.uniform(lb, ub, (n_particles, n_dimensions))

# 更新粒子位置和速度
def update_particles(particles, velocities, global_best, omega, phi_p, phi_g, lb, ub):
    r_p = np.random.uniform(0, 1, particles.shape)
    r_g = np.random.uniform(0, 1, global_best.shape)

    new_velocities = omega * velocities + phi_p * r_p * (global_best - particles) + phi_g * r_g
        * (global_best - particles)
    new_particles = particles + new_velocities

    # 边界处理
    new_particles = np.clip(new_particles, lb, ub)

    return new_particles, new_velocities

# 禁忌搜索函数
```

```python
def tabu_search(particle, objective_function, lb, ub):
    global tabu_list

    # 如果当前粒子位置在禁忌列表中，则随机移动到非禁忌区域
    while particle.tolist() in tabu_list:
        particle = np.random.uniform(lb, ub)

    # 将当前解加入禁忌列表
    tabu_list.append(particle.tolist())

    # 如果禁忌列表长度超过禁忌长度，则移除最老的禁忌解
    if len(tabu_list) > tabu_tenure:
        tabu_list.pop(0)

    return particle

# 使用禁忌搜索的变异粒子群优化（MPSO）
def mpso_tabu_search(objective_function, lb, ub, n_particles=30, max_iter=100, omega=0.5,
     phi_p=0.5, phi_g=0.5):
    n_dimensions = len(lb)

    # 初始化粒子位置和速度
    particles = initialize_swarm(n_particles, n_dimensions, lb, ub)
    velocities = np.random.uniform(-1, 1, (n_particles, n_dimensions))

    # 初始化全局最优解
    global_best = particles[np.argmin([objective_function(p) for p in particles])]

    # 迭代优化
    for _ in range(max_iter):
        particles, velocities = update_particles(particles, velocities, global_best, omega,
            phi_p, phi_g, lb, ub)

        # 使用禁忌搜索处理局部最优解
        for i in range(n_particles):
            particles[i] = tabu_search(particles[i], objective_function, lb, ub)

        # 更新全局最优解
        current_best = particles[np.argmin([objective_function(p) for p in particles])]
        if objective_function(current_best) < objective_function(global_best):
            global_best = current_best

    return global_best, objective_function(global_best)

# 运行带禁忌搜索的MPSO算法
best_solution, best_value = mpso_tabu_search(objective_function, lb, ub)
print("最优解:", best_solution) # 最优解
```

```
print("最优值:", best_value) # 最优值
```

Listing 5: nsga2.py

```
from collections import defaultdict
import numpy as np
import random
import matplotlib.pyplot as plt
import math
from goal_for_pro3 import *
class Individual(object):
    def __init__(self):
        self.solution = None # 实际赋值中是一个 nparray 类型，方便进行四则运算
        self.objective = defaultdict()

        self.n = 0 # 解p被几个解所支配，是一个数值（左下部分点的个数）
        self.rank = 0 # 解所在第几层
        self.S = [] # 解p支配哪些解，是一个解集合（右上部分点的内容）
        self.distance = 0 # 拥挤度距离

    def bound_process(self, bound_min, bound_max):
        """
        对解向量 solution
            中的每个分量进行定义域判断，超过最大值，将其赋值为最大值；小于最小值，赋值为最小值
        :param bound_min: 定义域下限
        :param bound_max:定义域上限
        :return:
        """
        for i, item in enumerate(self.solution):

            if item > bound_max[i]:
                self.solution[i] = bound_max[i]
            elif item < bound_min[i]:
                self.solution[i] = bound_min[i]

    def calculate_objective(self, objective_fun):
        self.objective = objective_fun(self.solution)

    # 重载小于号 “<”
    def __lt__(self, other):
        v1 = list(self.objective.values())
        v2 = list(other.objective.values())
        for i in range(len(v1)):
            if v1[i] > v2[i]:
                return 0 # 但凡有一个位置是 v1大于v2的 直接返回0,如果相等的话比较下一个目标值
        return 1
```

```
# TODO 如果目标函数是3个及以上呢，如果比较的支配方向不是越小越好呢？
def main():
    # 初始化/参数设置
    generations = 25 # 迭代次数
    popnum = 35 # 种群大小
    eta = 1 # 变异分布参数，该值越大则产生的后代个体逼近父代的概率越大。Deb建议设为 1

    # poplength = 30 # 单个个体解向量的维数
    # bound_min = 0 # 定义域
    # bound_max = 1
    # objective_fun = ZDT1

    poplength = 2 # 单个个体解向量的维数
    bound_min = [-90,-90] # 定义域
    bound_max = [90,90]
    objective_fun = KUR

    # 生成第一代种群
    P = []
    for i in range(popnum):
        P.append(Individual())
        P[i].solution = np.random.rand(poplength) * (np.array(bound_max) - np.array(bound_min))
            + np.array(bound_min) # 随机生成个体可行解
        P[i].bound_process(bound_min, bound_max) # 定义域越界处理
        P[i].calculate_objective(objective_fun) # 计算目标函数值

    # 否 -> 非支配排序
    fast_non_dominated_sort(P)
    Q = make_new_pop(P, eta, bound_min, bound_max, objective_fun)

    P_t = P # 当前这一届的父代种群
    Q_t = Q # 当前这一届的子代种群

    for gen_cur in range(generations):
        R_t = P_t + Q_t # combine parent and offspring population
        F = fast_non_dominated_sort(R_t)

        P_n = [] # 即为P_t+1,表示下一届的父代
        i = 1
        while len(P_n) + len(F[i]) < popnum: # until the parent population is filled
            crowding_distance_assignment(F[i]) # calculate crowding-distance in F_i
            P_n = P_n + F[i] # include ith non dominated front in the parent pop
            i = i + 1 # check the next front for inclusion
        F[i].sort(key=lambda x: x.distance) # sort in descending order using
            <n，因为本身就在同一层，所以相当于直接比拥挤距离
        P_n = P_n + F[i][:popnum - len(P_n)]
```

```
        Q_n = make_new_pop(P_n, eta, bound_min, bound_max,
                           objective_fun) # use selection,crossover and mutation to create a new
                               population Q_n

        # 求得下一届的父代和子代成为当前届的父代和子代，，进入下一次迭代 《=》 t = t + 1
        # 选择出最好的前10个个体
        best_individuals = sorted(P_t, key=lambda x: x.rank)[:1]

        # 打印或存储这些个体的解向量和目标函数值
        for idx, ind in enumerate(best_individuals):
            print(f"Best individual {idx + 1}:")
            print(f"Solution: {ind.solution}")
            print(f"Objective values: {ind.objective}")
        P_t = P_n
        Q_t = Q_n

        # 绘图
        plt.clf()
        plt.title('current generation:' + str(gen_cur + 1))
        plot_P(P_t)
        plt.pause(0.1)



    plt.show()

    return 0


def fast_non_dominated_sort(P):
    """
    非支配排序
    :param P: 种群 P
    :return F: F=(F_1, F_2, ...) 将种群 P 分为了不同的层， 返回值类型是dict，键为层号，值为 List
        类型，存放着该层的个体
    """
    F = defaultdict(list)
    for p in P:
        p.S = []
        p.n = 0
        for q in P:
            if p < q: # if p dominate q
                p.S.append(q) # Add q to the set of solutions dominated by p
            elif q < p:
                p.n += 1 # Increment the domination counter of p
        if p.n == 0:
            p.rank = 1
```

```
            F[1].append(p)

    i = 1
    while F[i]:
        Q = []
        for p in F[i]:
            for q in p.S:
                q.n = q.n - 1
                if q.n == 0:
                    q.rank = i + 1
                    Q.append(q)
        i = i + 1
        F[i] = Q

    return F


def crowding_distance_assignment(L):
    """ 传进来的参数应该是L = F(i), 类型是List"""
    l = len(L) # number of solution in F

    for i in range(l):
        L[i].distance = 0 # initialize distance

    for m in L[0].objective.keys():
        L.sort(key=lambda x: x.objective[m]) # sort using each objective value
        L[0].distance = float('inf')
        L[l - 1].distance = float('inf') # so that boundary points are always selected

        # 排序是由小到大的, 所以最大值和最小值分别是 L[l-1] 和 L[0]
        f_max = L[l - 1].objective[m]
        f_min = L[0].objective[m]


        # 当某一个目标方向上的最大值和最小值相同时, 此时会发生除零错, 这里采用异常处理机制来解决
        try:
            for i in range(1, l - 1): # for all other points
                L[i].distance = L[i].distance + (L[i + 1].objective[m] - L[i - 1].objective[m]) /
                    (f_max - f_min)
        except Exception:
            print(str(m) + "目标方向上, 最大值为" + str(f_max) + "最小值为" + str(f_min))


def binary_tournament(ind1, ind2):
    """
    二元锦标赛
    :param ind1:个体1号
```

```
    :param ind2: 个体2号
    :return:返回较优的个体
    """
    if ind1.rank != ind2.rank: # 如果两个个体有支配关系，即在两个不同的rank中，选择rank小的
        return ind1 if ind1.rank < ind2.rank else ind2
    elif ind1.distance != ind2.distance: # 如果两个个体rank相同，比较拥挤度距离，选择拥挤读距离大的
        return ind1 if ind1.distance > ind2.distance else ind2
    else: # 如果rank和拥挤度都相同，返回任意一个都可以
        return ind1


# TODO 真想把 eta, bound_min, bound_max, objective_fun 设为全局变量
def make_new_pop(P, eta, bound_min, bound_max, objective_fun):
    """
    use select,crossover and mutation to create a new population Q
    :param P: 父代种群
    :param eta: 变异分布参数，该值越大则产生的后代个体逼近父代的概率越大。Deb建议设为 1
    :param bound_min: 定义域下限
    :param bound_max: 定义域上限
    :param objective_fun: 目标函数
    :return Q : 子代种群
    """
    popnum = len(P)
    Q = []
    # binary tournament selection
    for i in range(int(popnum / 2)):
        # 从种群中随机选择两个个体，进行二元锦标赛，选择出一个 parent1
        i = random.randint(0, popnum - 1)
        j = random.randint(0, popnum - 1)
        parent1 = binary_tournament(P[i], P[j])

        # 从种群中随机选择两个个体，进行二元锦标赛，选择出一个 parent2
        i = random.randint(0, popnum - 1)
        j = random.randint(0, popnum - 1)
        parent2 = binary_tournament(P[i], P[j])

        while (parent1.solution == parent2.solution).all(): #
            如果选择到的两个父代完全一样，则重选另一个
            i = random.randint(0, popnum - 1)
            j = random.randint(0, popnum - 1)
            parent2 = binary_tournament(P[i], P[j])

        # parent1 和 parent1 进行交叉，变异 产生 2 个子代
        Two_offspring = crossover_mutation(parent1, parent2, eta, bound_min, bound_max,
            objective_fun)

        # 产生的子代进入子代种群
```

```
            Q.append(Two_offspring[0])
            Q.append(Two_offspring[1])
        return Q


def crossover_mutation(parent1, parent2, eta, bound_min, bound_max, objective_fun):
    """
    交叉方式使用二进制交叉算子（SBX），变异方式采用多项式变异（PM）
    :param parent1: 父代1
    :param parent2: 父代2
    :param eta: 变异分布参数，该值越大则产生的后代个体逼近父代的概率越大。Deb建议设为 1
    :param bound_min: 定义域下限
    :param bound_max: 定义域上限
    :param objective_fun: 目标函数
    :return: 2 个子代
    """
    poplength = len(parent1.solution)

    offspring1 = Individual()
    offspring2 = Individual()
    offspring1.solution = np.empty(poplength)
    offspring2.solution = np.empty(poplength)

    # 二进制交叉
    for i in range(poplength):
        rand = random.random()
        beta = (rand * 2) ** (1 / (eta + 1)) if rand < 0.5 else (1 / (2 * (1 - rand))) ** (1.0 /
            (eta + 1))
        offspring1.solution[i] = 0.5 * ((1 + beta) * parent1.solution[i] + (1 - beta) *
            parent2.solution[i])
        offspring2.solution[i] = 0.5 * ((1 - beta) * parent1.solution[i] + (1 + beta) *
            parent2.solution[i])

    # 多项式变异
    # TODO 变异的时候只变异一个，不要两个都变，不然要么出现早熟现象，要么收敛速度巨慢 why？
    for i in range(poplength):
        mu = random.random()
        delta = 2 * mu ** (1 / (eta + 1)) if mu < 0.5 else (1 - (2 * (1 - mu)) ** (1 / (eta +
            1)))
        offspring1.solution[i] = offspring1.solution[i] + delta

    # 定义域越界处理
    offspring1.bound_process(bound_min, bound_max)
    offspring2.bound_process(bound_min, bound_max)

    # 计算目标函数值
    offspring1.calculate_objective(objective_fun)
```

```
    offspring2.calculate_objective(objective_fun)

    return [offspring1, offspring2]


def ZDT1(x):
    """
    测试函数——ZDT1
    :parameter
    :param x: 为 m 维向量，表示个体的具体解向量
    :return f: 为两个目标方向上的函数值
    """
    poplength = len(x)
    f = defaultdict(float)

    g = 1 + 9 * sum(x[1:poplength]) / (poplength - 1)
    f[1] = x[0]
    f[2] = g * (1 - pow(x[0] / g, 0.5))

    return f


def ZDT2(x):
    poplength = len(x)
    f = defaultdict(float)

    g = 1 + 9 * sum(x[1:poplength]) / (poplength - 1)
    f[1] = x[0]
    f[2] = g * (1 - (x[0] / g) ** 2)

    return f


def KUR(x):
    f = defaultdict(float)
    poplength = len(x)
    east_angle, south_angle = x
    f[1] = 0
    f[2] = 0
    m1 = 0
    m2 = 0
    year = 2025
    day = 15
    for i in range(1,13):
        month = i
        m1, m2 = calculate_results(year, month, day, east_angle, south_angle)
        f[1] -= m1 # 上下午大于目标值的次数
```

```
        f[2] -= m2 # 总的能量
    return f


def plot_P(P):
    """
    假设目标就俩,给个种群绘图
    :param P:
    :return:
    """
    X = []
    Y = []
    for ind in P:
        X.append(ind.objective[1])
        Y.append(ind.objective[2])

    plt.xlabel('F1')
    plt.ylabel('F2')
    plt.scatter(X, Y)


def show_some_ind(P):
    # 测试使用
    for i in P:
        print(i.solution)


if __name__ == '__main__':
    main()
```

Listing 6: grid.py

```
from collections import defaultdict
import itertools
import math
import numpy as np
from matplotlib import pyplot as plt
from goal_for_pro3 import * # Assuming this module contains your 'calculate_results' function

# 解决中文乱码
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["font.family"] = "sans-serif"
# 解决负号无法显示的问题
plt.rcParams['axes.unicode_minus'] = False

# Define the fitness function K
def K(x, weights):
```

```
    try:
        f = defaultdict(float)
        poplength = len(x)
        east_angle, south_angle = x
        f[1] = 0
        f[2] = 0
        m1 = 0
        m2 = 0
        year = 2025
        day = 15
        for i in range(1, 13):
            month = i
            m1, m2 = calculate_results(year, month, day, east_angle, south_angle)
            f[1] -= m1 # 上下午大于目标值的次数
            f[2] -= m2 / 10000 # 总的能量
        return weights[0] * f[1] + weights[1] * f[2]
    except:
        return 0

# Define variable ranges (bounds)
bounds = [(-4, 30), (20, 40)] # For each variable, set a range (min, max)

# Define weights for multi-objective optimization
weights = [5, 0.5] # Weight for each objective function

# Define step size
step = 1 # Step size is 1 degree

best_score = 0

# Define the grid search range
angles_range = [range(bounds[i][0], bounds[i][1] + 1, step) if bounds[i][1] != 0 else [0] for
    i in range(len(bounds))]
print(angles_range)

# Iterate over all combinations of angles in the search range
for angles in itertools.product(*angles_range):
    score = K(angles, weights)
    if score > best_score:
        best_score = score
        best_solution = angles

# Print the results
print("最优解:", best_solution) # Best solution
print("目标函数值:", best_score) # Objective function value
```