**八、附录**

代码一：

% 太阳角度  
clear;close all;clc;  
  
% N = [21 52 80 111 141 172 202 233 264 294 325 355];  
% b = 2\*pi.\*(N - 1)/365;  
% weidujiao = 0.006918 - 0.399912.\*cos(b) + 0.070257.\*sin(b)...  
%     - 0.006758.\*cos(2.\*b) + 0.000907.\*cos(3.\*b) + 0.00148.\*sin(3.\*b);  
% ST = [9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.5 14.0 14.5 15.0];  
% w = pi.\*(ST - 12)/12;  
% A1 = cos(weidujiao)\*cos(39.4).\*cos(w) + sin(weidujiao)\*sin(39.4);  
% A2 = (sin(weidujiao) - A1\*sin(39.4)) / (sqrt(1-A1.^2))\*cos(39.4);  
  
  
D = [-58 -28 0 31 61 92 122 153 184 214 245 275];  
A = sin(2\*pi.\*D/365)\*sin(2\*pi\*23.45/360);  
ST = [9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.5 14.0 14.5 15.0];  
w = pi.\*(ST - 12)/12;  
A1 = sqrt(1-A.^2)\*cos(39.4).\*cos(w) + A\*sin(39.4);  
A2 = (A - A1\*sin(39.4)) / (sqrt(1-A1.^2))\*cos(39.4);

代码二：

% 对表格数据进行处理，求出距离原点的长度  
  
clear;close all;clc;  
  
data=readmatrix("D:\Race\数学建模\CUMCM2023Problems\A题\附件.xlsx","Range","A2:B1746");  
  
P=data(:,1);  
E=data(:,2);  
% 距离原点的距离RO  
RO = sqrt(P.^2 + E.^2);  
% 镜面中心到集热器中心的距离DHR  
DHR = sqrt(RO.^2 + 80\*80);  
% 大气透射率  
NAT = 0.99321 - 0.0001176.\*DHR + 1.97\*10^(-8).\*DHR.^2;  
  
% 添加新的两列数据  
new\_data = [data, RO, DHR,NAT];  
  
% 写入 Excel 文件  
xlswrite('处理后数据.xlsx', new\_data);

代码三：

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
def RED(FPath):  
    return pd.read\_excel(FPath)  
  
def CMB(center, normal):  
    halfSize = 3.0  
    if not (normal[0] == 0 and normal[1] == 0):  
        PV1 = np.cross(normal, [0, 0, 1])  
    else:  
        PV1 = np.cross(normal, [0, 1, 0])  
    PV1 = PV1 / np.linalg.norm(PV1)  
    PV2 = np.cross(normal, PV1)  
    PV2 = PV2 / np.linalg.norm(PV2)  
  
    p1 = center + halfSize \* PV1 + halfSize \* PV2  
    p2 = center - halfSize \* PV1 + halfSize \* PV2  
    p3 = center - halfSize \* PV1 - halfSize \* PV2  
    p4 = center + halfSize \* PV1 - halfSize \* PV2  
  
    return [p1, p2, p3, p4]  
  
def RPI(rayOrigin, rayDir, planePoint, planeNormal):  
    dot\_product = np.dot(rayDir, planeNormal)  
    if abs(dot\_product) < 1e-6:  
        return None  
    t = np.dot(planePoint - rayOrigin, planeNormal) / dot\_product  
    if t < 0:  
        return None  
   IP = rayOrigin + t \* rayDir  
    returnIP  
  
def isPIP(pt, polygon):  
    NV = len(polygon)  
    inter = 0  
    for i in range(NV):  
        p1, p2 = polygon[i], polygon[(i + 1) % NV]  
        if (pt[1] > min(p1[1], p2[1])) and (pt[1] <= max(p1[1], p2[1])):  
            x\_inters = (pt[1] - p1[1]) \* (p2[0] - p1[0]) / (p2[1] - p1[1]) + p1[0]  
            if pt[0] < x\_inters:  
                inter += 1  
    return inter % 2 == 1  
  
def CO(data):  
    occlusion = []  
    for idx, row in data.iterrows():  
        center = np.array([row["x坐标 (m)"], row["y坐标 (m)"], row["z坐标 (m)"]])  
        normal = np.array([row["Cosine X"], row["Cosine Y"], row["Cosine Z"]])  
        boundaries = CMB(center, normal)  
        occluded = False  
        for j, other\_row in data.iterrows():  
            if idx != j:  
                other\_center = np.array([other\_row["x坐标 (m)"], other\_row["y坐标 (m)"], other\_row["z坐标 (m)"]])  
                other\_normal = np.array([other\_row["Cosine X"], other\_row["Cosine Y"], other\_row["Cosine Z"]])  
                other\_boundaries = CMB(other\_center, other\_normal)  
                intersection = RPI(center, normal, other\_center, other\_normal)  
                if intersection is not None and isPIP(intersection, other\_boundaries):  
                    occluded = True  
                    break  
        occlusion.append(0 if occluded else 1)  
    return occlusion  
  
# 3D可视化  
def V3d(data, occlusionResults):  
    x = data["x坐标 (m)"].values  
    y = data["y坐标 (m)"].values  
    z = data["z坐标 (m)"].values  
    colors = ["red" if occluded == 0 else "blue" for occluded in occlusionResults]  
    fig = plt.figure(figsize=(12, 8))  
    ax = fig.add\_subplot(111, projection='3d')  
    ax.scatter(x, y, z, c=colors, marker='o')  
    ax.set\_xlabel('X')  
    ax.set\_ylabel('Y')  
    ax.set\_zlabel('Z')  
    ax.set\_title('3D)')  
    plt.show()  
  
def main():  
    FPath = "附件.xlsx"  
    data = RED(FPath)  
    occlusionResults = CO(data)  
    data["Occluded"] = occlusionResults  
    updated\_FPath = "附件更新.xlsx"  
    data.to\_excel(updated\_FPath, index=False)  
V3d(data, occlusionResults)

代码四：

import numpy as np  
import pandas as pd  
  
def CR(center, cosine, size=6):  
    DV = np.array(cosine)  
    HD = (size / 2) \* (DV / np.linalg.norm(DV))  
    vertex1 = center + HD  
    vertex2 = center - HD  
    vertex3 = center + [-HD[0], HD[1], HD[2]]  
    vertex4 = center - [-HD[0], HD[1], HD[2]]  
    return vertex1, vertex2, vertex3, vertex4  
  
def IA(rect1, rect2):  
    rect1X = [point[0] for point in rect1]  
    rect1Y = [point[1] for point in rect1]  
    rect2X = [point[0] for point in rect2]  
    rect2Y = [point[1] for point in rect2]  
      
    overlapX = max(0, min(max(rect1X), max(rect2X)) - max(min(rect1X), min(rect2X)))  
    overlapY = max(0, min(max(rect1Y), max(rect2Y)) - max(min(rect1Y), min(rect2Y)))  
      
    return overlapX \* overlapY  
  
def PB(current, total, BL=50):  
    progress = (current / total)  
    arrow = '=' \* int(round(progress \* BL) - 1) + '>'  
    spaces = ' ' \* (BL - len(arrow))  
    print(f"\rProgress: [{arrow + spaces}] {int(round(progress \* 100))}%", end='')  
  
def COI(data, customCosine=None):  
    numPoints = len(data)  
    resultsMatrix = np.zeros((numPoints, numPoints))  
    rectangleArea = 6 \* 6  
      
    for idx in range(numPoints):  
        target\_point = data.iloc[idx]  
        target\_rect = CR(  
            np.array([target\_point["x坐标 (m)"], target\_point["y坐标 (m)"], target\_point["z坐标 (m)"]]),  
            customCosine if customCosine else [target\_point["Cosine X"], target\_point["Cosine Y"], target\_point["Cosine Z"]])  
          
        for j in range(idx + 1, numPoints):  
            row = data.iloc[j]  
            other\_rect = CR(  
                np.array([row["x坐标 (m)"], row["y坐标 (m)"], row["z坐标 (m)"]]),  
                customCosine if customCosine else [row["Cosine X"], row["Cosine Y"], row["Cosine Z"]])  
            area = IA(target\_rect, other\_rect) / rectangleArea  
            resultsMatrix[idx, j] = area  
            resultsMatrix[j, idx] = area  
          
        # 更新进度条  
        PB(idx + 1, numPoints)  
      
    print("\nDone!")  
    return resultsMatrix  
  
def main():  
    # 自定义方向余弦  
    customCosine = [0.5, 0.5, 0.5]   
      
    # 读取指定路径的Excel文件  
    dataPath = "附件更新.xlsx"  
    dataAll = pd.read\_excel(dataPath)  
      
    # 仅选取倒数第二列为0的点  
    dataSelected = dataAll[dataAll.iloc[:, -2] == 0]  
      
    # 对这些选定的点进行操作  
    IM = COI(dataSelected, customCosine)  
      
    intersectionDf = pd.DataFrame(IM, index=dataSelected.index, columns=dataSelected.index)  
    dataAll\_combined = pd.concat([dataAll, intersectionDf], axis=1, join="outer")  
      
    dataAll\_combined.to\_excel(dataPath, index=False)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()

代码五：

import numpy as np  
import pandas as pd  
  
def CR(center, cosineX, cosineY, size=6):  
    DV\_xy = np.array([cosineX, cosineY])  
    HD = (size / 2) \* (DV\_xy / np.linalg.norm(DV\_xy))  
    vertex1 = center + np.append(HD, 0)  
    vertex2 = center - np.append(HD, 0)  
    vertex3 = center + np.append([-HD[0], HD[1]], 0)  
    vertex4 = center - np.append([-HD[0], HD[1]], 0)  
    return vertex1, vertex2, vertex3, vertex4  
  
def IA(rect1, rect2):  
    rect1X = [point[0] for point in rect1]  
    rect1Y = [point[1] for point in rect1]  
    rect2X = [point[0] for point in rect2]  
    rect2Y = [point[1] for point in rect2]  
      
    overlapX = max(0, min(max(rect1X), max(rect2X)) - max(min(rect1X), min(rect2X)))  
    overlapY = max(0, min(max(rect1Y), max(rect2Y)) - max(min(rect1Y), min(rect2Y)))  
      
    return overlapX \* overlapY  
  
def PB(current, total, BL=50):  
    progress = (current / total)  
    arrow = '=' \* int(round(progress \* BL) - 1) + '>'  
    spaces = ' ' \* (BL - len(arrow))  
    print(f"\rProgress: [{arrow + spaces}] {int(round(progress \* 100))}%", end='')  
  
def COI(data):  
    numPoints = len(data)  
    resultsMatrix = np.zeros((numPoints, numPoints))  
    rectangleArea = 6 \* 6  
      
    for idx in range(numPoints):  
        target\_point = data.iloc[idx]  
        target\_rect = CR(  
            np.array([target\_point["x坐标 (m)"], target\_point["y坐标 (m)"], target\_point["z坐标 (m)"]]),  
            target\_point["Cosine X"], target\_point["Cosine Y"])  
          
        for j in range(idx + 1, numPoints):  
            row = data.iloc[j]  
            other\_rect = CR(  
                np.array([row["x坐标 (m)"], row["y坐标 (m)"], row["z坐标 (m)"]]),  
                row["Cosine X"], row["Cosine Y"])  
            area = IA(target\_rect, other\_rect) / rectangleArea  
            resultsMatrix[idx, j] = area  
            resultsMatrix[j, idx] = area  
          
        PB(idx + 1, numPoints)  
      
    print("\nDone!")  
    return resultsMatrix  
  
def main():  
    dataPath = "附件更新.xlsx"  
    dataAll = pd.read\_excel(dataPath)  
      
    dataSelected = dataAll[dataAll.iloc[:, -2] == 0]  
      
    IM = COI(dataSelected)  
      
    intersectionDf = pd.DataFrame(IM, index=dataSelected.index, columns=dataSelected.index)  
    dataAll\_combined = pd.concat([dataAll, intersectionDf], axis=1, join="outer")  
      
    dataAll\_combined.to\_excel(dataPath, index=False)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()