所属类别	2025 年“华数杯”全国大学生数学建模竞赛	参赛编号
本科组		CM2501873

基于非线性规划模型的光谱组合优化和节律效应研究

摘　要

随着 LED 的广泛使用，LED 光源在满足基础照明需求之外，其光谱特性对人体生理节律系统的影响逐渐成为研究焦点。本文从**光谱特性数据分析视角**出发，并在实验中探究其优化光谱对人体的改善作用。

针对问题一，先对给定的 SPD 数据进行**数据预处理**，并明确 XYZ 三刺激值、色品坐标 (x, y) 以及 CIE 1960 UCS 坐标 (u, v) 的数据换算方法。在此基础上，对于颜色特性参数，采用**三角垂足插值法** CCT，根据定义采用色品坐标偏差的几何距离公式计算 Duv；对于颜色还原参数，依据 **TM-30 标准**，通过计算待测光源和参考光源的**色差**和**色域面积比例**分别计算出 R_f 和 R_g；对于生理节律效应参数，根据定义计算测试光源的褪黑素光效比与标准日光的褪黑素光效比，得到 $mel-DER$。三类五核心参数计算结果分别为：$CCT = 3903.2$、$Duv = -0.001016$、$R_f = 92.9$、$R_g = 99.0$ 以及 $mel - DER = 0.60933$

针对问题二，首先建立五通道 LED **光谱合成模型**，其中含有待优化的权重变量，其次基于**非线性规划模型**建立日间照明、夜间助眠场景的**光谱组合优化模型**，根据问题一的理论模型，模拟场景特定需求作为目标函数，模拟场景参数作为约束条件，最后通过**粒子群优化算法（PSO）**求解目标函数，进而获得合成光谱的最优权重组合。结果显示两个场景的优化结果较好满足设计要求。

针对问题三，首先建立五通道 LED 光源的**基于非线性规划的加权多目标优化模型**，其次以褪黑素日光照度比为核心目标构建非线性目标函数，最后通过**模拟退火全局搜索 (SA)** 和**序列二次规划局部搜索（SLSQP）**的**混合优化策略**求解各时段最优光谱比例。该策略在保证其他关键参数非线性约束偏差范围内的前提下，成功**复现了太阳光谱的节律调控特性**。

针对问题四，首先处理原始睡眠阶段记录，通过**数据清洗**和统计获得关键指标，采用**重复测量方差分析 (ANOVA)** 及**事后检验（Bonferroni 校正）**评估组间差异，观察和分析相关**箱线图**。结果显示，仅深睡眠比例（N3%）在不同光照环境下间存在显著差异，其余指标在三种环境下均无统计学显著差异。但同时表明，优化光照对睡眠质量的**改善有一定效果**。

最后本文对所建立的模型进行了讨论和分析，综合评价模型的局限性和可改进的地方。

关键词：**非线性规划**；**粒子群算法**；**模拟退火**；**混合优化**；**ANOVA**

一、问题重述

1.1 问题背景

随着发光二极管（LED）作为高效、节能、环保的新型光源在照明等多个领域得到广泛应用，以及人们对光照与人体生理关联研究的不断深入，LED 光源在满足基础照明需求之外，其光谱特性对人体生理节律系统的影响逐渐成为研究焦点。

因此，如何在保证照明功能（如色温适配、色彩还原能力）的前提下，通过优化 LED 光源的光谱功率分布（SPD），实现对人体生理节律的有益调节，成为当前亟待解决的重要科学问题，具有显著的理论研究价值与实际应用意义。

1.2 问题提出

问题是层层递进，且服务于同一主题的——通过分析 LED 光源光谱特性，进而寻找对人体有益的生理调节效应的光源环境，现根据附件表相关数据，建立数学模型解决以下四个问题：

(1) 问题一：根据附件“Problem 1”的表格数据，在已知波长和光强变化情况的基础上，通过文献中一系列标准化的数学模型，计算得出用于评估光源性能的五个关键参数。

(2) 问题二：根据附件“Problem 2_LED_SPD”的表格数据，针对日间照明、夜间助眠两种场景的特定需求，寻找最优通道权重组合，进而合成满足需求的光谱，输出关键参数。

(3) 问题三：根据附件“Problem 3 SUN_SPD”的表格数据，结合问题二给的五通道 LED，设计合成光谱的控制策略，使得能够模拟给定的太阳光谱数据，具有相似的节律效果。

(4) 问题四：基于问题二、三提出的的理论模型，进行了人体实验验证，分析三种光照环境对被试睡眠结构的影响。根据附件“Problem 4”的表格数据，通过统计六大衡量指标，选择统计检验方法对睡眠质量进行评估，从而判断 LED 优化是否对人体健康有显著改善效果。

二、问题分析

2.1 问题一的分析

问题一是数值计算问题，阅读文献，理解相应的一系列标准化数学模型，根据附件中提供的 SPD 光谱数据进行计算。

2.2 问题二的分析

问题二是非线性规划优化问题，建立五通道 LED 光谱合成模型，其中含有待优化的权重变量，基于问题一的理论模型建立目标函数，以此模拟日间照明、夜间助眠场景的特定需求，通过求解目标函数，进而迭代出合成光谱的最优权重组合。

2.3 问题三的分析

问题三是非线性规划优化问题，需建立五通道 LED 光源的基于非线性规划的加权多目标优化模型，以褪黑素日光照度比为核心目标构建非线性目标函数，通过非线性规划算法求解各时段最优光谱比例，从而复现太阳光谱的节律调控特性。

2.4 问题四的分析

问题四是统计检验问题，需要对附件中的数据进行数据清洗，不同编码的相关睡眠质量指标进行统计和数据可视化展示，进而通过重复测量方差分析和 t 检验来确定光照环境对于人体睡眠状况是否有无显著差异。

三、模型假设

为了将问题简化，我们在数据处理和模型构建中提出以下合理假设：

假设 1 附件所给数据数据准确无误

假设 2 LED 灯光各通道光谱功率稳定，不受使用时间、外部环境等变化影响

假设 3 五个 LED 通道是相互独立的, 的光谱在合成过程中满足线性叠加原理, 不存在相互作用影响的光谱特性

假设 4 人体之间在不同光照环境下的睡眠质量是相互独立的，其他影响睡眠质量的因素保持恒定，且个体差异不会影响系统整体观测偏差

四、符号说明

表 1 变量符号及其解释说明

符号	解释说明
λ_i	离散波长点,$i = 1, 2, ..., 400$
$P(\lambda_i)$	波长 λ_i 处的光谱功率
$\bar{x}(\lambda_i), \bar{y}(\lambda_i), \bar{z}(\lambda_i)$	波长 λ_i 处的颜色匹配函数值
$\Delta\lambda$	相邻波长点的间隔
k	归一化常数
h	普朗克常数,$h = 6.62607015 \times 10^{-34}$
k_B	玻尔兹曼常数 $k_B = 1.380649 \times 10^{-23}$
T	黑体绝对温度
j'	明度分量坐标
a'	红绿分量坐标
b'	黄蓝分量坐标
ΔE	色差
a_i	第 i 个色样在 $CAM02-UCS$ 的投影红绿分量坐标
b_i	第 i 个色样在 $CAM02-UCS$ 的投影黄蓝分量坐标
$s_{\mathrm{mel}}(\lambda)$	CIE S 026 定义的褪黑素感光细胞（melanopsin）光谱加权函数
$V(\lambda)$	CIE 1931 明视觉光谱光视效率函数
K_m	明视觉最大光视效能,K_m=683lm/W
w_B	Blue LED 权重
w_G	Green LED 权重
w_R	Red LED 权重
w_{WW}	Warm White LED 权重
w_{CW}	Cold White LED 权重
T	指从关灯入睡到最终醒来的总时长
$\bar{x}_{\cdot j}$	第 j 种环境下的均值
SS_{total}	总平方和
$MS_{\mathrm{treatments}}$	处理均方
$df_{\mathrm{treatments}}$	处理自由度
T_{CCT}	太阳光谱的 CCT 指标
T_{Duv}	太阳光谱的 Duv 指标
T_{Rf}	太阳光谱的 Rf 指标
T_{Rg}	太阳光谱的 Rg 指标
T_{mel}	太阳光谱的 mel-DER 指标

五、问题一的模型建立与求解

5.1 数据的预处理

经初步检查，本题数据无缺失值，将波长标准化，舍去单位提取纯数值，转换为浮点型（float），确保后续计算一致性，并进行插值绘制可视化曲线。

5.2 数据的换算

5.2.1 XYZ 三刺激值的换算

将 SPD 数据与每个颜色匹配函数逐点相乘后对波长进行积分,离散形式下的 SPD 数据，可通过离散求和近似积分计算，最后与 k 相乘，得到归一化后的 X,Y,Z 三刺激值，公式如下:

$$\begin{cases} X = k\displaystyle\int_{380}^{780} P(\lambda)\bar{x}(\lambda)\mathrm{d}\lambda \;\approx\; k\sum_{i=380}^{780} P(\lambda)\bar{x}(\lambda)\Delta\lambda \\ Y = k\displaystyle\int_{380}^{780} P(\lambda)\bar{y}(\lambda)\mathrm{d}\lambda \;\approx\; k\sum_{i=380}^{780} P(\lambda)\bar{y}(\lambda)\Delta\lambda \\ Z = k\displaystyle\int_{380}^{780} P(\lambda)\bar{z}(\lambda)\mathrm{d}\lambda \;\approx\; k\sum_{i=380}^{780} P(\lambda)\bar{z}(\lambda)\Delta\lambda \end{cases} \tag{1}$$

其中 k 的计算公式为:

$$k = \frac{100}{\sum_{i=1}^{n}\left[\mathrm{P}(\lambda_i)\cdot\bar{y}(\lambda_i)\cdot\Delta\lambda\right]} \tag{2}$$

5.2.2 色品坐标 (x, y) 的换算

$$\begin{cases} x = \dfrac{X}{X+Y+Z} \\ y = \dfrac{Y}{X+Y+Z} \end{cases} \tag{3}$$

5.2.3 CIE 1960 UCS 坐标 (u, v) 的换算

$$\begin{cases} u = \dfrac{4x}{-2x+12y+3} \\ v = \dfrac{6y}{-2x+12y+3} \end{cases} \tag{4}$$

5.3 第一类参数的计算：颜色特性参数模型

5.3.1 相关色温（CCT）

CCT[6] 是当光源颜色与任何温度下的黑体辐射颜色不完全匹配时，在 CIE 色度图上找到与该光源色坐标最接近的黑体辐射轨迹上的点, 该点对应的黑体温度即为相关色温。

三角垂足插值法计算 CCT 的优势主要体现在其将复杂的色度学计算转化为直观的几何问题求解，避免了传统迭代法可能出现的收敛问题，还保证了计算效率与精度的理想平衡，步骤如下:

step1 输入待评价光源的坐标 $C(u_s, v_s)$

step2 在黑体轨迹 - 温度数据表中寻找最近两点 $A(u_1, v_1, T_1)$ 和 $B(u_2, v_2, T_2)$

step3 直线 AB 模拟代表黑体轨迹弧线方程:

$$\frac{u_s - u_1}{v_s - v_1} = \frac{u_2 - u_1}{v_2 - v_1} \tag{5}$$

step4 计算垂足 $E(u_0, v_0)$:

$$\begin{cases} u_0 = \dfrac{(v_s - v_1 \cdot k_{AB}) \cdot u_1 / k_{CE} + u_s}{1/k_{CE} + k_{AB}} \\ v_0 = k_{AB} \cdot (u_0 - u_1) + v_1 \end{cases} \tag{6}$$

其中 $k_{AB} = \dfrac{v_2 - v_1}{u_2 - u_1}$, $k_{CE} = -1/k_{AB}$

step5 计算垂足 $E(u_0, v_0)$ 微倒度插值:

$$\text{mired}_0 = \text{mired}_1 + \frac{d_1}{d_1 + d_2} (\text{mired}_2 - \text{mired}_1) \tag{7}$$

其中 d_1、d_2 分别为垂足 E 到 A 和 B 点的距离，$\text{mired}_i = \frac{10^6}{T_i}$ 分别代表点 E、A、B 坐标的微倒度值

step6 最终 CCT:

$$\text{CCT} = \frac{10^6}{\text{mired}_e} \tag{8}$$

5.3.2 距离普朗克轨迹的距离（Duv）

Duv 即色偏差，用于描述光源点相对黑体轨迹的位置，在计算 CCT 的基础上，Duv 的公式为：

$$Duv = \text{sign}(v_s - v_0) \cdot \sqrt{(u_0 - u_s)^2 + (v_0 - v_s)^2} \tag{9}$$

通过符号化色偏差（Duv）[2] 明确指示偏离方向，同时也满足 CIE 标准 $|Duv|$ 0.054 的有效性判据。

5.4 第二类参数的计算：颜色还原参数模型

5.4.1 保真度指数（R_f）

R_f [3] 反映被测光源下颜色与标准光源下颜色的一致性，值越高表示颜色还原越准确，对于每个颜色样本，计算待测光源与参考光源的色差：

$$\Delta E_i = \sqrt{(J'_i - J'_{\text{ref},i})^2 + (a'_i - a'_{\text{ref},i})^2 + (b'_i - b'_{\text{ref},i})^2} \tag{10}$$

R_f 是 99 个颜色样本的平均色差经变换后的值：

$$R_f = \frac{1}{99} \sum_{i=1}^{99} \Delta E_i \tag{11}$$

5.4.2 色域指数（R_g）

R_g [3] 表示被测光源相对于参考光源的色域面积比例，

$$R_g = 100 \times \frac{Y_{\text{test}}}{Y_{\text{ref}}} \tag{12}$$

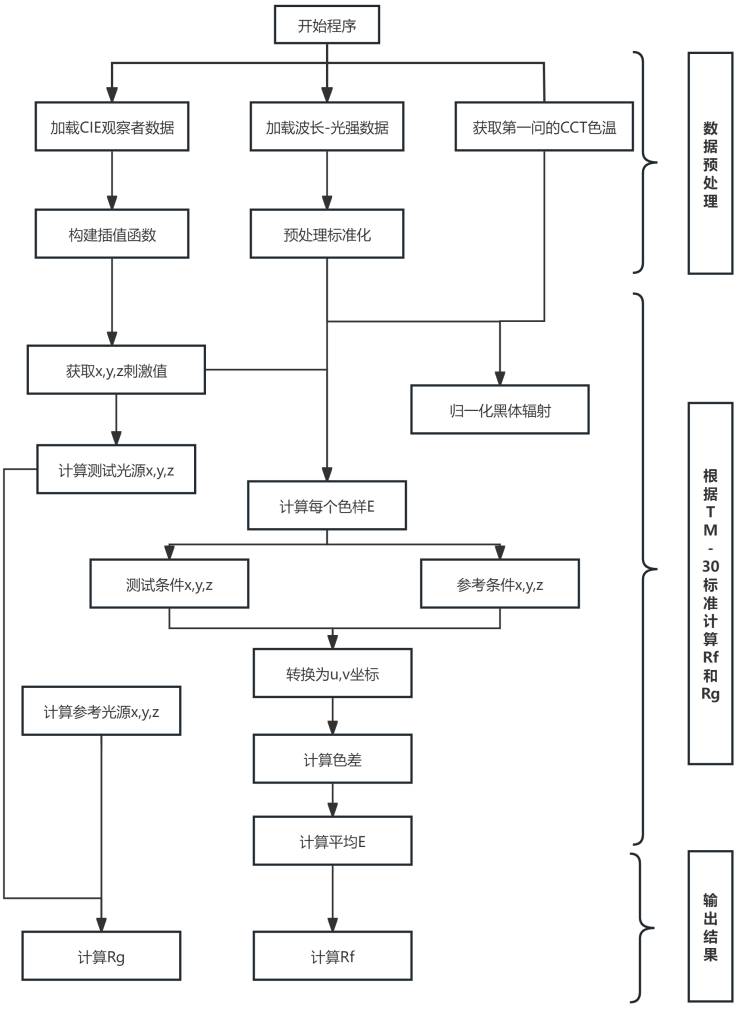


图 1　颜色还原参数模型流程图

5.5 第三类参数的计算: 生理节律效应参数模型

5.5.1 褪黑素日光照度比（$mel-DER$）

$mel-DER$[4] [1] 是测试光源的褪黑素光效比（melanopic ELR，$K_{mel,v}$ ）与标准日光（D65）的褪黑素光效比（$K_{mel,v}^{D65}$）的比值，公式为:

$$mel-DER=\frac{K_{mel,v}}{K_{mel,v}^{D65}} \tag{13}$$

其中，$K_{mel,v}^{D65}=1.3262\ \mathrm{mW/lm}$，$K_{mel,v}$ 的计算公式为:

$$K_{mel,v}=\frac{\int_{380}^{780}P(\lambda)\cdot s_{\mathrm{mel}}(\lambda)\,d\lambda}{K_m\cdot\int_{380}^{780}P(\lambda)\cdot V(\lambda)\,d\lambda} \tag{14}$$

5.6 总结

以下是所计算出的 LED 光源光谱的特征参数及其意义的简要表格说明:

表 2 问题一数值结果与分析

参数名称	数值	含义与分析
相对色温（CCT）	3903.2	光源色温属于中低暖色调范围 (典型值：2700K-4000K)
色偏差（Duv）	-0.001016	接近黑体辐射轨迹（理想值 0） 色偏可忽略
色保真度指数（R_f）	92.9	接近基准光源 对标准色样还原能力优异
色域指数（R_g）	99.0	与基准光源色彩范围一致 无显著过饱和/欠饱和现象
褪黑素日光照度比（$mel - DER$）	0.60933	生理影响较弱 符合中低色温光源特性 适用于节律敏感度低的场景

六、问题二的模型建立与求解

6.1 光谱合成模型

由格拉斯曼定律（Grassmann's Laws）可知，混合光的光谱可以由各种组分的光谱进行叠加而得出，由此构造光谱合成模型:

$$S(\lambda) = \sum_{i \in \{B,G,R,WW,CW\}} w_i \cdot SPD_i(\lambda) \tag{15}$$

决策变量:

$$\mathbf{w} = [w_B, w_G, w_R, w_{WW}, w_{CW}]^T \tag{16}$$

初始值：$\mathbf{w}^{(0)} = [0.2, 0.2, 0.2, 0.2, 0.2]^T$

6.2 日间照明模式光谱组合优化模型

根据目标函数的非线性以及约束条件的多样性，采用非线性规划模型构造日间照明模式光谱组合优化模型

6.2.1 非线性目标函数

由问题一可知 R_f 的计算公式是非线性的, 最小化 $-R_f$，让人工光环境回归人类对自然光线的视觉依赖与生理适应:

$$\min_{S(\lambda)} \left(-R_f(S(\lambda)) + P(S(\lambda))\right. \tag{17}$$

其中 $P(S(\lambda))$ 是违反约束的惩罚项:

$$P(S(\lambda)) = \begin{cases} 0.1\times \text{ |CCT-6500|} & \text{if CCT}\notin[5500,6500] \\ 2\times|\mathrm{R}_g - 100| & \text{if } \mathrm{R}_g\notin[95,105] \\ 5\times(88 - \mathrm{R}_f) & \text{if } \mathrm{R}_f < 88 \end{cases}$$

6.2.2 约束条件

等式约束:

$$\sum_{i=1}^{5} w_i = 1 \tag{18}$$

不等式约束:

$$\begin{cases} 5500 \leqslant \mathrm{CCT}(S(\lambda)) \leqslant 6500 \\ \mathrm{R}_f(S(\lambda)) > 88 \\ 95 \leqslant R_g(S(\lambda)) \leqslant 105 \; 0.01 \leqslant w_B \leqslant 1 \\ 0.01 \leqslant w_G \leqslant 1 \\ 0.01 \leqslant w_R \leqslant 0.5 \\ 0.01 \leqslant w_{WW} \leqslant 0.5 \\ 0.001 \leqslant w_{CW} \leqslant 1 \end{cases} \tag{19}$$

其中，红光和暖白光以长波、低色温成分为主，所以限制红光与暖白光的权重上限分别为 0.5 与 0.5

6.2.3 求解方法

模型求解采用粒子群优化算法 (PSO)，具体 PSO 求解步骤如下:

step1 初始化粒子群

表 3 粒子群初始化参数

参数	符号	值
粒子数量	N	50
最大迭代次数	max_iter	200
惯性权重	w	0.8
个体学习因子	c_1	1.5
社会学习因子	c_2	1.5
边界约束	$bounds$	$w_i \in [\mathrm{lb}_i, \mathrm{ub}_i] \quad \forall i \in \{B, G, R, WW, CW\}$

step2 计算适应度函数：对每个粒子，根据其位置 X（即权重）合成光谱，计算 R_f 和 R_g，以及 CCT ，然后计算适应度函数 $fitness(S(\lambda))$，适应度函数设计即为目标函数

step3 初始化个体最优 (pbest) 和全局最优 (gbest)

step4 迭代优化过程

step5 输出全局最优解 (gbest) 及其适应度：全局最优解即最优权重

$$\begin{aligned}\mathbf{w}^* &= [w_B^*, w_G^*, w_R^*, w_{WW}^*, w_{CW}^*] \\ \sum w_i^* &= 1 \\ Rf^* &= -gbest\_value + P(gbest\_position)\end{aligned}$$

6.3 夜间助眠模式光谱组合优化模型

根据目标函数的非线性以及约束条件的多样性，采用非线性规划模型构造夜间助眠模式光谱组合优化模型

6.3.1 非线性目标函数

由问题一可知 $mel-DER$ 的计算公式是非线性的，最小化 $mel-DER$, 以此降低生物节律干扰，符合夜间模式需求：

$$\min_{S(\lambda)} mel - DER(S(\lambda)) \tag{20}$$

6.3.2 约束条件

等式约束：

$$\sum_i w_i = 1 \tag{21}$$

不等式约束：

$$\begin{cases} 2500 \leqslant \mathrm{CCT}(S(\lambda)) \leqslant 3500 \\ \mathrm{Rf}(S(\lambda)) \geqslant 80 \\ 0.01 \leqslant w_B \leqslant 0.5 \\ 0.01 \leqslant w_G \leqslant 0.5 \\ 0.01 \leqslant w_R \leqslant 1.0 \\ 0.01 \leqslant w_{WW} \leqslant 1.0 \\ 0.001 \leqslant w_{CW} \leqslant 0.1 \end{cases} \tag{22}$$

其中，冷白光含短波（蓝光）成分较多，且 melanopic 响应最强波段为蓝绿色光，所以限制蓝色光、绿色光上限分别为 0.5、0.5，限制冷色光的上下限分别为 0.1、0.01

6.3.3 求解方法

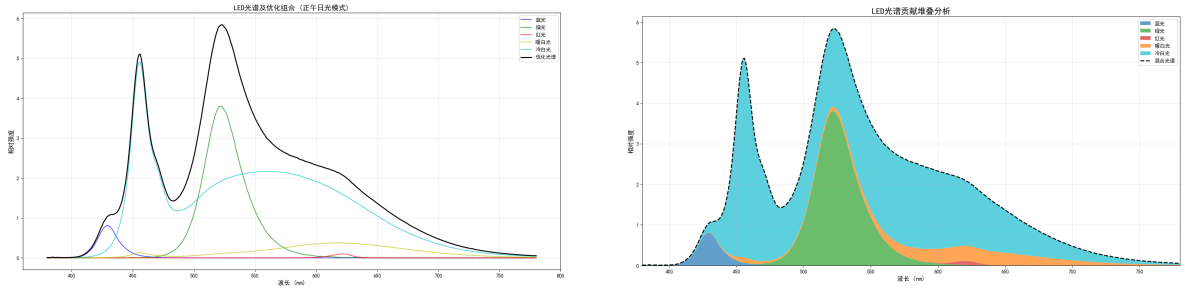
模型求解仍采用 PSO 优化算法进行求解，依靠附录中 python 代码实现

6.4 模型结果

6.4.1 日间照明模式

表 4　最优的 LED 通道权重组合及合成光谱的参数

LED 混合光谱优化结果	合成光谱的参数
Blue (w_B) : 0.0817	CCT： 6499.9
Green (w_G) : 0.3797	Duv ： 0.0381
Red (w_R) : 0.0100	R_f ： 96.17
Warm White (w_{WW}) : 0.0377	R_g ： 110.00
Cold white (w_{CW}) : 0.4910	$mel-DER$ ： 0.8264



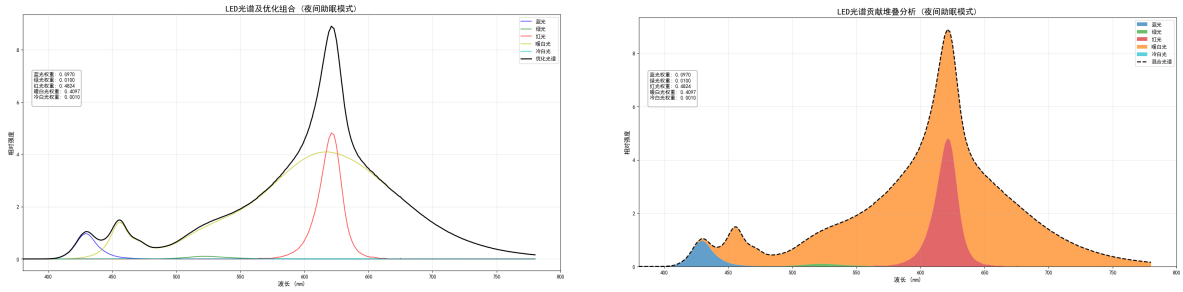
(a) LED 光谱及优化组合　　(b) LED 光谱贡献堆叠分析

图 2　正午日光模式

6.4.2 夜间助眠模式

表 5　最优的 LED 通道权重组合及合成光谱的参数

LED 混合光谱优化结果	合成光谱的参数
Blue (w_B) : 0.0970	CCT： 3499.6
Green (w_G) : 0.01	Duv ： -0.0037
Red (w_R) : 0.4824	R_f ： 87.5
Warm White (w_{WW}) : 0.4097	R_g ： 99.9
Cold white (w_{CW}) : 0.001	$mel-DER$ ： 0.0003



(a) LED 光谱及优化组合　　(b) LED 光谱贡献堆叠分析

图 3　夜间助眠模式

6.5 模型拟合

表 6　光谱优化组合模型拟合情况（日间）

参数	合成光谱	目标范围
CCT	6499.9	$[5500, 6500]$
Duv	0.0381	$[-0.054, 0.054]$
R_f	96.17	$[88, 105]$
R_g	110.00	$[95, 105]$
$mel-DER$	0.8264	$[0.8000, 1.0000]$

表 7　光谱优化组合模型拟合情况（夜间）

参数	合成光谱	目标范围
CCT	3499.6	$[2500, 3500]$
Duv	-0.0037	$[-0.054, 0.054]$
R_f	87.5	$[80, 90]$
R_g	99.9	$[95, 105]$
$mel-DER$	0.0003	$[0.0001, 0.0005]$

七、问题三的模型建立与求解

7.1 合成的光谱控制策略模型建立

首先, 计算出不同时间点下的 Problem 3 SUN_SPD 中各个 SPD 指标的结果, 如下表所示:

表 8 Problem3 SPD **指标计算结果**

时间点	$CCT(K)$	Duv	R_f	R_g	$mel-DER$
5:30:00	5355.2	-0.000845	99.0	100	0.84486
6:30:00	5319.5	-0.000798	99.0	100	0.84223
7:30:00	5303.0	-0.000702	99.0	100	0.83954
8:30:00	5273.4	-0.000604	99.0	100	0.83678
9:30:00	5410.0	-0.001021	99.1	100	0.84846
10:30:00	5508.7	-0.001345	99.1	100	0.85893
11:30:00	5611.7	-0.001682	99.0	100	0.86837
12:30:00	5668.0	-0.001897	99.0	100	0.87692
13:30:00	5608.6	-0.001744	99.3	100	0.86914
14:30:00	5504.2	-0.001612	99.2	100	0.86054
15:30:00	5401.2	-0.001351	99.2	100	0.85096
16:30:00	5304.0	-0.001104	99.1	100	0.84023
17:30:00	5152.1	-0.001489	99.0	100	0.82652
18:30:00	4860.0	-0.002139	98.7	99	0.79349
19:30:00	3463.5	0.001155	96.9	97	0.60335

利用问题二的光谱合成模型[5] 进行研究

7.2 非线性目标函数

这是一个基于非线性规划的加权多目标优化问题，以最小化以下各项的加权平方误差之和：主要目标：褪黑素日光照度比（mel-DER）的精确匹配（权重为 1000）。次要目标：其他指标（CCT, Duv, Rf, Rg）的匹配（权重各为 0.1）。软约束惩罚项：当各指标超出放宽的约束范围时，施加额外的惩罚（权重为 10）。具体表达式如下：

$$\begin{aligned}\text{obj} = 1000 \cdot \left(\frac{\text{mel\_DER} - T_{\text{mel}}}{T_{\text{mel}}}\right)^2 + 0.1 \cdot \left(\frac{\text{CCT} - T_{\text{CCT}}}{T_{\text{CCT}}}\right)^2 \\ + 0.1 \cdot \left(\frac{\text{Duv} - T_{\text{Duv}}}{0.001}\right)^2 + 0.1 \cdot \left(\frac{\text{Rf} - T_{\text{Rf}}}{T_{\text{Rf}}}\right)^2 \\ + 0.1 \cdot \left(\frac{\text{Rg} - T_{\text{Rg}}}{T_{\text{Rg}}}\right)^2 + \text{soft\_penalty}\end{aligned} \tag{23}$$

其中，软约束惩罚项（soft_penalty）的计算为：

如果 CCT 超出目标值 ±500K，则加上

$$10 \cdot \left(\frac{\mathrm{CCT} - T_{\mathrm{CCT}}}{500} \right)^2$$

如果 Duv 超出目标值 ±0.01，则加上

$$10 \cdot \left(\frac{\mathrm{Duv} - T_{\mathrm{Duv}}}{0.01} \right)^2$$

如果 Rf 超出目标值 ±15，则加上

$$10 \cdot \left(\frac{\mathrm{Rf} - T_{\mathrm{Rf}}}{15} \right)^2$$

如果 Rg 超出目标值 ±15，则加上

$$10 \cdot \left(\frac{\mathrm{Rg} - T_{\mathrm{Rg}}}{15} \right)^2$$

(其中, 目标值即为 Problem 3 SUN_SPD 中各个 SPD 指标的结果)

7.3 约束条件

我们建立如下约束优化问题：

$$\begin{cases} \sum_{i=1}^{5} w_i = 1 \\ 0.001 \leqslant w_i \leqslant 0.95 \quad \forall i \in \{B, G, R, WW, CW\} \end{cases} \tag{24}$$

7.4 求解算法

针对目标函数非凸、多极值且约束为线性等式/不等式的特性，采用混合优化策略（全局搜索 + 局部精细优化）确保解的全局性与可行性，具体步骤如下：

step1 **全局优化框架设计**

1 多初始点生成策略为覆盖解空间并规避局部最优，生成 8 组满足约束的初始点：

1.1 均匀分布采样：在 5 维参数空间（LED 通道比例系数）内均匀采样，保证基础覆盖。

1.2 偏置采样：针对关键光学指标预设权重分布，生成偏向特定 LED 组合的初始点（如高 melanopic 贡献的蓝绿光通道），提升对目标区域的搜索效率。

2 模拟退火算法（Simulated Annealing）作为全局搜索核心，模拟退火通过可控的随机性跳出局部最优，其物理退火过程的核心数学实现如下：

2.1 状态转移概率：

$$P = \begin{cases} 1 & \text{if } \Delta f \leqslant 0 \\ \exp\left(-\frac{\Delta f}{T}\right) & \text{if } \Delta f > 0 \end{cases}$$

2.2 自适应降温初始温度 $T0$=1000，冷却率 =0.95，每迭代 100 次降温一次，平衡探索与开发。

2.3 邻域生成通过随机扰动当前解（如高斯噪声）生成新解，并投影至约束空间（比例系数归一化、非负性）。

step2 **局部优化加速收敛**

1 序列二次规划（SLSQP）

1.1 处理非线性约束及边界条件（比例系数 wi=1）。

1.2 利用 Hessian 矩阵近似快速收敛

step3 **参数输出**

1 从 8 组优化结果中选取目标函数值最小的解为最终解，确保全局最优性

7.5 统计结果

求得的各时间点 LED 混合光谱优化结果及其节律效果 (mel-DER) 与各时刻混合光波长与光谱功率图保存于附件“LED 混合光谱优化结果及节律效果.docx”中。

7.5.1 模型检验

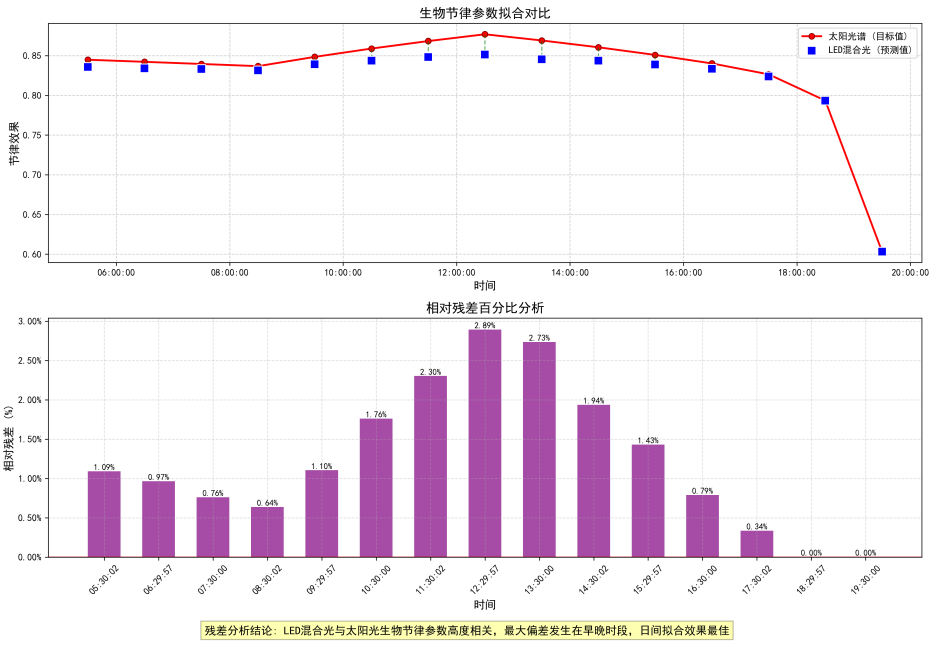


图 4 生物节律参数分析

通过上图可得：LED 混合光能高度还原太阳光的生物节律调节作用（整体残差 <3%），尤其适合日间光照需求场景；但在模拟日出/日落的光动态特性时需进一步优化，以更精准地匹配人体昼夜节律的生理变化节奏。

7.5.2 案例分析

表 9 LED 混合光谱优化结果及节律效果（早晨）

时间点	LED 混合光谱优化结果	光学指标	太阳光谱目标
5:30:00	Blue (w_B : 0.1093	mel-DER: 0.83564	$mel-DER$: 0.84486
	Green (w_G) : 0.0173	CCT: 5545.5	CCT: 5355.2
	Red (w_R) : 0.1029	Duv: -0.000844	Duv: -0.000845
	Warm White (w_{WW}) : 0.0010	Rf (保真度): 80.0	R_f : 99.0
	Cold white (w_{CW}) : 0.7695	Rg (色域指数): 97.2	R_g : 100.0

表 10 LED 混合光谱优化结果及节律效果（正午）

时间点	LED 混合光谱优化结果	光学指标	太阳光谱目标
12:30:00	Blue (w_B : 0.1031	$mel-DER$: 0.85154	$mel-DER$: 0.87692
	Green (w_G) : 0.0020	CCT: 5870.4K	CCT: 5668.0
	Red (w_R) : 0.0233	Duv: -0.001897	Duv: -0.001897
	Warm White (w_{WW}) : 0.0012	Rf (保真度): 80.0	R_f : 99.0
	Cold white (w_{CW}) : 0.8704	Rg (色域指数): 96.7	R_g : 100.0

表 11 LED 混合光谱优化结果及节律效果（傍晚）

时间点	LED 混合光谱优化结果	光学指标	太阳光谱目标
19:30:00	Blue (w_B) : 0.2559	mel-DER: 0.60057	$mel-DER$: 0.60335
	Green (w_G) : 0.0262	CCT: 3679.4K	CCT: 3463.5
	Red (w_R) : 0.1521	Duv: -0.001156	Duv: -0.001155
	Warm White (w_{WW}) : 0.2401	Rf (保真度): 80.0	R_f : 96.9
	Cold white (w_{CW}) : 0.3265	Rg (色域指数): 99.8	R_g : 97.0

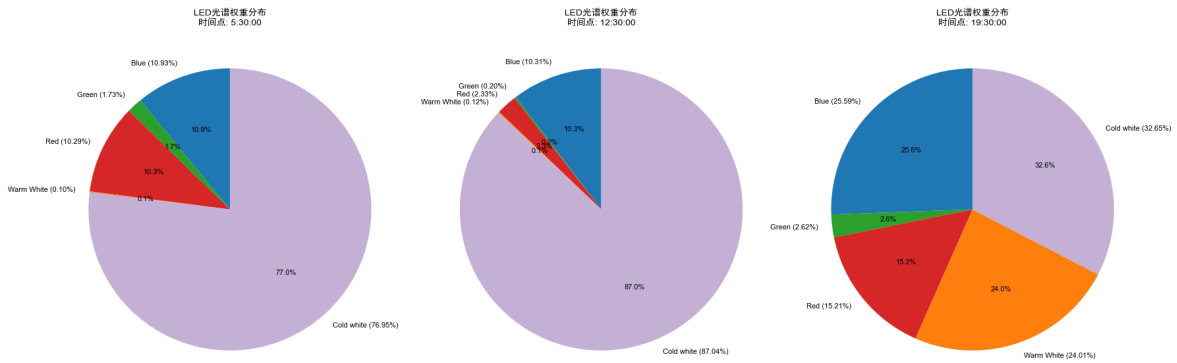
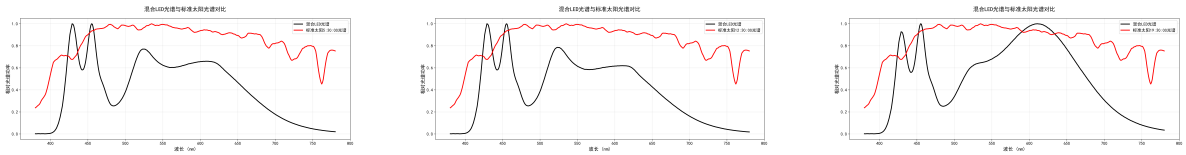


图 5　三个时间段下各通道权重饼状图

1. 清晨 5:30　自然光特征：低色温（暖黄光为主），光谱功率平缓上升，蓝光成分逐渐增加。LED 模拟效果：在 480nm 蓝光波段出现峰值（人工光源特性），在关键波段（500-600 nm）有效模拟了清晨自然光谱的平缓过渡特征，其 `mel-DER` 值与自然光处于相似水平。

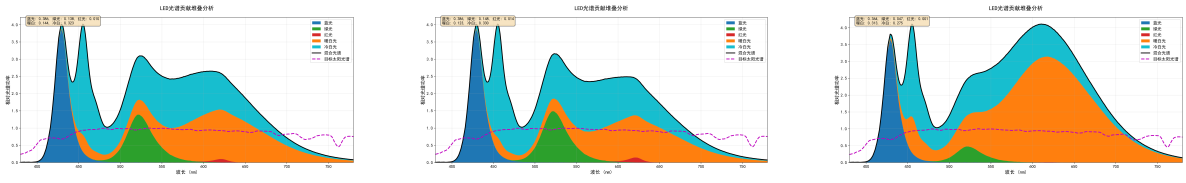
2. 正午 12:30　自然光特征：高色温（>5500K），蓝光波段（460-480nm）强度峰值，光谱平滑连续。LED 模拟效果：LED 混合光谱虽在蓝光区域（400-500 nm）的形态（LED 离散峰 vs 太阳连续谱）与太阳光谱存在差异，但通过高比例的冷白光成分实现了高色温与显著的蓝光波段输出，并达到了与自然光相当的 `mel-DER` 值（表征该时段光谱的节律效应潜力）。

3. 傍晚 19:30　自然光特征：色温骤降（<3000K），红光（600-700nm）比例显著上升，蓝光强度锐减。LED 模拟效果：LED 混合光谱在长波段（>650 nm）的覆盖范围与太阳光谱存在差异，但通过显著提升红光和暖白光成分的比例，成功模拟了低色温和红光主导的光谱特征，其 `mel-DER` 值显著降低，与傍晚自然光的节律效应特征相符。



(a) 5:30:00　(b) 12:30:00　(c) 19:30:00

图 6　不同时间点下 LED 合成光谱与目标太阳光谱对比图



(a) 5:30:00　　(b) 12:30:00　　(c) 19:30:00

图 7　不同时间点下 LED 合成光谱堆叠图

7.6 控制策略总结

时段	光谱调控目标	关键操作	节律效果
清晨	提升节律刺激	↑ 冷白光 ↑ 蓝光 ↓ 红光	mel-DER 持续升高，使人平稳过渡至清醒状态
正午	最大化节律效应	冷白光主导 (>80%) ↑ 蓝光	mel-DER 达峰值 (>0.85)，提升专注力与工作效率
下午	平滑过渡至低刺激	↑ 红光 ↓ 冷白光 ↓ 蓝光	mel-DER 缓降，缓解神经兴奋性
夜间	最小化生物钟干扰	↑ 暖白光 ↓ 冷白光平衡蓝光	mel-DER 速降至最低，维持低蓝光危害，保障睡眠

该控制策略通过动态调整五通道 LED 的光强比例，精准复现了太阳光谱在全天不同时段的节律特性：

1、白天高 mel-DER 值（>0.83）模拟强生物钟刺激

2、傍晚红移光谱加速 mel-DER 衰减

3、夜间低蓝光光谱（暖白光主导）最小化节律干扰

本质上是通过光谱组成的变化（尤其是蓝光与红光的权重切换）实现对自然光生物效应的动态模拟，为人工光环境提供科学的节律调控方案。

八、问题四的模型建立与求解

8.1 睡眠质量指标计算公式

1、**总睡眠时间** (Total Sleep Time, TST)：

$$\mathrm{TST} = \sum (t_{\mathrm{N1}} + t_{\mathrm{N2}} + t_{\mathrm{N3}} + t_{\mathrm{REM}}) \tag{25}$$

2、**睡眠效率** (Sleep Efficiency, SE)：

$$\mathrm{SE} = \left(\frac{\mathrm{TST}}{T_{\text{总卧床}}} \right) \times 100\% \tag{26}$$

3、**入睡潜伏期** (Sleep Onset Latency, SOL)：

$$\mathrm{SOL} = t_{\text{首次睡眠}} - t_{\text{关灯}} \tag{27}$$

4、**深睡眠比例** (N3%):

$$\mathrm{N3\%} = \left(\frac{t_{\mathrm{N3}}}{\mathrm{TST}}\right) \times 100\% \tag{28}$$

5、**REM 睡眠比例** (REM%):

$$\mathrm{REM\%} = \left(\frac{t_{\mathrm{REM}}}{\mathrm{TST}}\right) \times 100\% \tag{29}$$

6、**夜间醒来次数** (Number of Awakenings):

$$\mathrm{Awakenings} = \sum \mathbb{I}(\text{清醒事件}) \tag{30}$$

8.2 对数据进行预处理和统计

1. 数据清洗：初步观察附件表格中并无缺失值，和超出睡眠阶段编码的异常值。但每组测试的观察组数不同，进行空值处理，避免无效记录干扰统计。

2. 采用按列分组策略,遍历每位被试在三种光照环境中的睡眠记录数据,按照指标的计算公式带入并进行统计,具体表格统计结果保存于附件“all_subjects_sleep_metrics.xlsx”中。

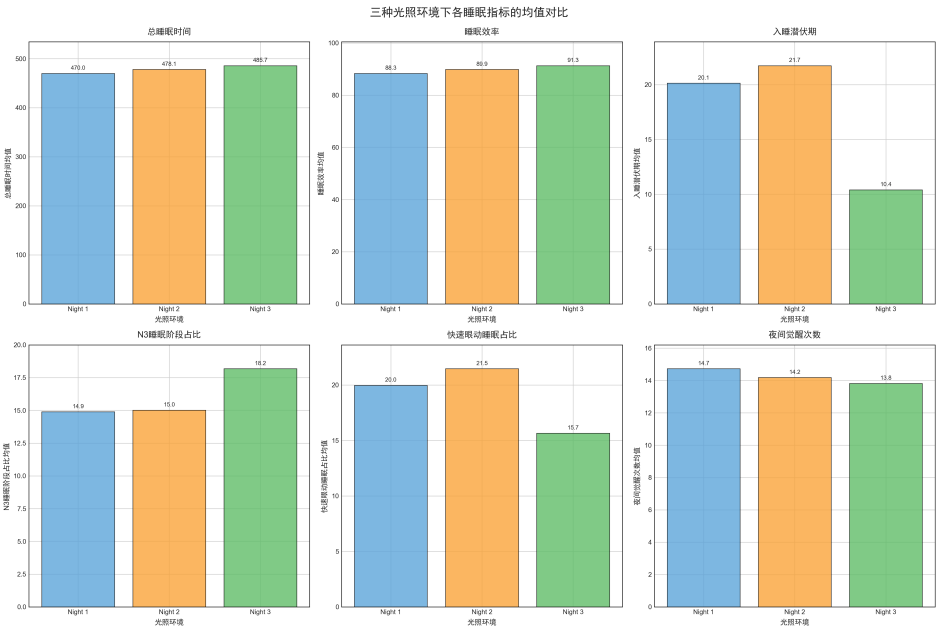


图 8　不同光照环境下各指标分析

表 12 表格统计结果部分展示

被试编号	光照环境	TST	SE	SOL	N3%	REM%	Awakenings
1	环境 A（优化光照）	310	86.96	15	14.68	20.48	19
	环境 B（普通 LED）	312	90.04	2	24.52	23.24	19
	环境 C（黑暗）	288.5	82.08	5	25.65	25.65	19
2	环境 A（优化光照）	322	65.78	16	17.24	32.14	17
	环境 B（普通 LED）	348	75.24	5	30.89	29.74	13
	环境 C（黑暗）	347	91.68	0	27.81	23.63	16

8.3 统计检验方法

为分析 3 种光照环境对睡眠指标的影响是否存在统计学差异，本研究采用重复测量方差分析（Repeated Measures ANOVA）[7]。该方法适用于同一被试接受多种处理的实验设计，能够有效分离个体差异对结果的干扰，精准检验处理效应的显著性。

8.3.1 平方和分解

通过分解总变异来源，量化不同因素对睡眠指标的影响：

总平方和（SS_{total}）：反映所有观测值与总均值的总离散程度

$$SS_{\text{total}} = \sum_{i=1}^{n}\sum_{j=1}^{k}(x_{ij} - \bar{x}_{\cdot\cdot})^2$$

被试间平方和（SS_{subjects}）：反映个体差异导致的变异

$$SS_{\text{subjects}} = k\sum_{i=1}^{n}(\bar{x}_{i\cdot} - \bar{x}_{\cdot\cdot})^2$$

处理间平方和（$SS_{\text{treatments}}$）：反映不同光照环境（处理）导致的变异

$$SS_{\text{treatments}} = n\sum_{j=1}^{k}(\bar{x}_{\cdot j} - \bar{x}_{\cdot\cdot})^2$$

误差平方和（SS_{error}）：反映随机误差导致的变异

$$SS_{\text{error}} = SS_{\text{total}} - SS_{\text{subjects}} - SS_{\text{treatments}}$$

8.3.2 均方与 F 统计量

为检验处理效应的显著性，需计算均方（Mean Square, MS）和 F 统计量：

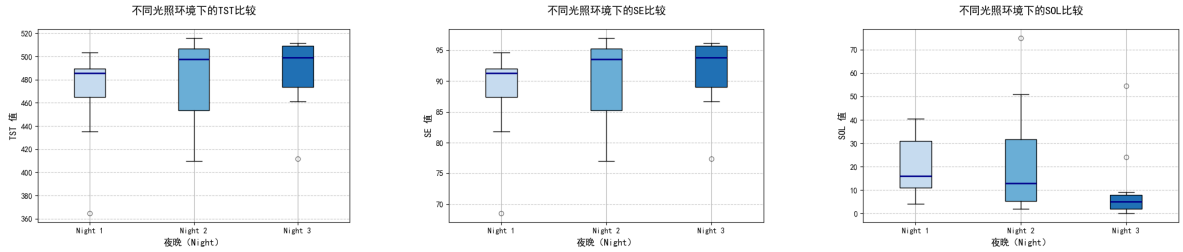
均方：平方和与自由度的比值

$$MS_{\text{treatments}} = \frac{SS_{\text{treatments}}}{df_{\text{treatments}}}, \quad MS_{\text{error}} = \frac{SS_{\text{error}}}{df_{\text{error}}}$$

F **统计量**：衡量处理效应与误差的相对强度

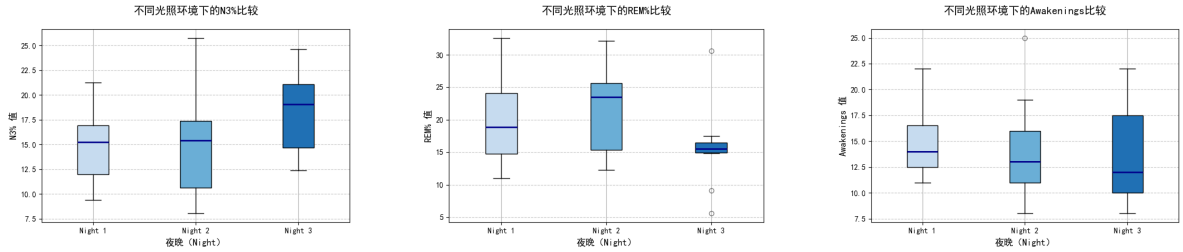
$$F = \frac{MS_{\text{treatments}}}{MS_{\text{error}}} \sim F(df_{\text{treatments}}, df_{\text{error}})$$

若 F 值对应的概率 $p < 0.05$，则认为不同光照环境对该睡眠指标的影响存在显著差异。



(a) TST　　(b) SE　　(c) SOL

图 9　不同光照环境下睡眠质量指标箱线图（1）



(a) N3　　(b) REM　　(c) Awakening

图 10　不同光照环境下睡眠质量指标箱线图（2）

8.4 统计结果与结论分析

表 13　三种光照环境下睡眠指标的统计分析结果

睡眠指标	重复测量方差分析 $\mathbf{Pr} > \mathbf{F}$
总睡眠时间（TST）	0.5922
睡眠效率（SE）	0.5921
入睡潜伏期（SOL）	0.2783
深睡眠比例（N3%）	0.1603
REM 睡眠比例（REM%）	0.0439
夜间醒来次数	0.7061

结果显示，仅深睡眠比例（N3%）在重复测量方差分析中存在统计显著差异，事后检验进一步表明优化光照与黑暗环境间该指标差异显著，黑暗环境下深睡眠比例更高且效应量大；而其他睡眠指标，在重复测量方差分析及事后检验中均未表现出统计上的显著差异。

但优化光照相比于普通光照和黑暗环境，在部分睡眠指标上显示出潜在的改善效果，说明问题二、三通过优化 LED 光谱特性是具有研究价值与意义的。

8.5 问题研究的局限性

1. 参与的样本量相对较小，可能导致研究结果的统计效力不足，个体差异可能会对整体观测结果产生影响，难以准确捕捉到光照环境对睡眠指标可能存在的细微影响

2. 每种环境仅进行了一晚的实验，无法反映长期处于不同光照环境下对睡眠的累积效应，且单晚的睡眠情况可能受偶然因素（如当天情绪、身体状态等）影响较大，难以稳定、真实地体现不同光照环境与睡眠指标之间的关联。

3. 未考虑可能的其他环境因素的影响, 这些因素可能与光照环境共同作用于睡眠, 从而干扰对光照环境单独作用的准确评估，导致研究结果的解释存在一定局限性。

九、模型的评价和推广

9.1 模型的优点

1. 模型在多通道光源的优化设计中展现出较高的灵活性和科学性。通过调节五个 LED 通道的权重组合，能够精确控制合成光谱的色温、显色性以及生理节律效应，从而满足不同照明场景的需求。

2. 模型基于光谱叠加原理和色度学理论，结合黑体辐射和视黑素敏感度曲线等科学方法，确保了优化结果的准确性和可靠性。

3. 模拟退火算法可根据一定概率可以放弃局部最优解，找到全局最优。

4.ANOVA 结果直观，通过 F 统计量和 p 值明确判断因素对目标变量的影响程度，可同时检验这里多个对照组数据间的显著性差异，效率较高。

9.2 模型的缺点

1. 模型的优化过程涉及大量的数值计算和插值操作，计算复杂度较高，可能导致优化时间较长，尤其是在高精度需求的情况下。

2. 模型的优化结果对初始权重和边界条件的设定较为敏感，如果初始参数设置不合理，可能会导致算法陷入局部最优解，从而影响最终的光谱优化效果。

3. ANOVA 对异常值敏感，而数据中样本数量较少可能扭曲分析结果。

9.3 模型的推广

该模型为拟合自然界各种光的光谱特性提供了新的解决思路，能够精准模拟不同时段、不同天气条件下的自然光，还能通过拟合的光谱变化数据曲线对未来一段时间的光照变化进行预测，具有广泛的市场前景和社会经济价值。

参考文献

[1] Photometry - The CIE system of physical photometry, 2004.

[2] XU Jing, YANG Long, WANG Junjian, and SUN Zhongyu. Effects of color temperature of visible light camera on vegetation index calculation in unmanned aerial vehicle vegetation remote sensing. *Chinese Journal of Applied Ecology/Yingyong Shengtai Xuebao*, 36(3), 2025.

[3] Michael P Royer. Tutorial: Background and guidance for using the ansi/ies tm-30 method for evaluating light source color rendition. *Leukos*, 18(2):191–231, 2022.

[4] Luc JM Schlangen, Luke Price, David Sliney, and Manuel Spitschan. Report on the workshop use and application of the new cie s 026/e: 2018, metrology for iprgc-influenced responses to light“specifying light for its eye-mediated non-visual effects in humans”. In *Proceedings of the 29th Session of the CIE*, pages 114–118. CIE, 2019.

[5] Jingjing Zhang, Weihong Guo, Bin Xie, Xingjian Yu, Xiaobing Luo, Tao Zhang, Zhihua Yu, Hong Wang, and Xing Jin. Blue light hazard optimization for white light-emitting diode sources with high luminous efficacy of radiation and high color rendering index. *Optics & Laser Technology*, 94:193–198, 2017.

[6] 张浩 and 徐海松. 光源相关色温算法的比较研究. 光学仪器, (1):54–58, 2006.

[7] 胡良平. 具有重复测量的多因素设计类型及方差分析. 中国卫生统计, 7(5):20–23, 1990.

附　录

附录 1：问题一第一类参数求解的 python 代码

```
   import numpy as np
import pandas as pd
from scipy import integrate
from scipy.constants import h, c, k

def load_cie_data(csv_path):
   """读取CIE配色函数CSV文件，返回波长和x_bar, y_bar, z_bar数据"""
   df =pd.read_csv(csv_path)
   return {
      'wavelength': df.iloc[:, 0].values,
      'x_bar': df.iloc[:, 1].values,
      'y_bar': df.iloc[:, 2].values,
      'z_bar': df.iloc[:, 3].values
   }

def planck_spectrum(wavelength_nm, T):
   """计算黑体光谱辐射率 (W·sr⁻¹·m⁻²·nm⁻¹)"""
   wavelength_m =wavelength_nm *1e-9
   term1 =2 *h *c**2 /wavelength_m**5
   term2 =np.exp(h *c /(wavelength_m *k *T)) -1
   return (term1 /term2) *1e-9

def calculate_xy_from_T(T, cie_data):
   """计算色坐标(x,y)"""
   S =planck_spectrum(cie_data['wavelength'], T)
   X =integrate.simpson(S *cie_data['x_bar'], cie_data['wavelength'])
   Y =integrate.simpson(S *cie_data['y_bar'], cie_data['wavelength'])
   Z =integrate.simpson(S *cie_data['z_bar'], cie_data['wavelength'])
   x =X /(X +Y +Z)
   y =Y /(X +Y +Z)
   return x, y

def calculate_daylight_locus(T):
   """计算日光轨迹的色品坐标 (x_D, y_D)"""
   if 4000 <=T <7000:
      x_D =-4.607e9 /T**3 +2.9678e6 /T**2 +99.11 /T +0.244063
   elif T >=7000:
      x_D =-2.0064e9 /T**3 +1.9018e6 /T**2 +247.48 /T +0.23704
   else:
      x_D, y_D =calculate_xy_from_T(T, cie_data)
   y_D =-3 *x_D**2 +2.87 *x_D -0.275
   return x_D, y_D
```

```
def xy_to_uv(x, y):
    """将CIE xy坐标转换为uv坐标"""
    denominator =-2 *x +12 *y +3
    u =4 *x /denominator
    v =6 *y /denominator
    return u, v

def find_nearest_points(u_s, v_s, temp_list):
    """找到与目标点(u_s, v_s)最近的两个黑体轨迹点"""
    daylight_uv =[(xy_to_uv(*calculate_daylight_locus(T)) +(T,)) for T in temp_list
                                                    ]
    distances =[np.sqrt((u -u_s)**2 +(v -v_s)**2) for u, v, T in daylight_uv]
    nearest_idx =np.argmin(distances)
    if nearest_idx ==0:
        return daylight_uv[0], daylight_uv[1]
    elif nearest_idx ==len(daylight_uv) -1:
        return daylight_uv[-2], daylight_uv[-1]
    else:
        return daylight_uv[nearest_idx], daylight_uv[nearest_idx +1]

def calculate_foot_point(u_s, v_s, u1, v1, u2, v2):
    """计算点(u_s, v_s)到直线AB的垂足E"""
    if u1 ==u2:
        return u1, v_s
    if v1 ==v2:
        return u_s, v1
    k_AB =(v2 -v1) /(u2 -u1)
    b_AB =v1 -k_AB *u1
    k_CE =-1 /k_AB
    b_CE =v_s -k_CE *u_s
    u_e =(b_CE -b_AB) /(k_AB -k_CE)
    v_e =k_AB *u_e +b_AB
    return u_e, v_e

def calculate_cct_and_duv(x, y, temp_min=2000, temp_max=10000, step=50):
    """计算相关色温(CCT)和色偏差(Duv)"""
    temp_list =np.arange(temp_min, temp_max +step, step)
    u_s, v_s =xy_to_uv(x, y)
    (u1, v1, T1), (u2, v2, T2) =find_nearest_points(u_s, v_s, temp_list)
    u_e, v_e =calculate_foot_point(u_s, v_s, u1, v1, u2, v2)
    d1 =np.sqrt((u_e -u1)**2 +(v_e -v1)**2)
    d2 =np.sqrt((u_e -u2)**2 +(v_e -v2)**2)
    mired_e =1e6/T1 +(d1 /(d1 +d2)) *(1e6/T2 -1e6/T1)
    cct =1e6 /mired_e
    distance =np.sqrt((u_e -u_s)**2 +(v_e -v_s)**2)
    sign =1 if (v_s -v_e) >0 else (-1 if (v_s -v_e) <0 else 0)
    duv =sign *distance
    return cct, duv
```

```
def print_results(x, y):
    """输出计算结果"""
    cct, duv =calculate_cct_and_duv(x, y)
    u, v =xy_to_uv(x, y)
    print(f"输入色度坐标 (x, y): ({x:.6f}, {y:.6f})")
    print(f"转换后的uv坐标 (u, v): ({u:.6f}, {v:.6f})")
    print(f"相关色温 (CCT): {cct:.1f} K")
    print(f"色偏差 (Duv): {duv:.6f}")

# 示例使用
if __name__ =="__main__":
    cie_data =load_cie_data("data.csv")

    # 测试点1: D65标准光源
    print("测试点1 - D65标准光源:")
    print_results(0.3127, 0.3290)

    # 测试点2: 自定义点
    print("\n测试点2 - 自定义点:")
    print_results(0.3875, 0.3733)
```

附录 2: 问题一第二类参数求解的 python 代码

```
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d

# 配置参数
SPD_FILE ="附录T1.xlsx"
CIE_FILE ="ciexyz31_1.csv"
CCT =3903.2 # 已知色温(K)


def load_data():
    """加载数据文件"""
    cie =pd.read_csv(CIE_FILE, skiprows=1, names=['wl', 'x', 'y', 'z'])
    spd =pd.read_excel(SPD_FILE)
    wl =spd['波长'].str.replace('(mW/m2/nm)', '', regex=False).astype(float).values
    intensity =spd['光强'].values
    return cie, wl, intensity


def calculate_XYZ(wl, spd, cie_interp, normalize=True):
    """
    计算三刺激值
    :param normalize: 是否返回归一化后的XYZ
```

```
    :return: 归一化后的XYZ或原始XYZ
    """
    X =np.sum(spd *cie_interp['x'](wl))
    Y =np.sum(spd *cie_interp['y'](wl))
    Z =np.sum(spd *cie_interp['z'](wl))

    if normalize:
        XYZ_sum =X +Y +Z
        return np.array([X, Y, Z]) /XYZ_sum # 归一化
    else:
        return np.array([X, Y, Z]) # 原始值


def xy_to_uv(x, y):
    """CIE1960 UCS转换"""
    denom =-2 *x +12 *y +3
    return 4 *x /denom, 6 *y /denom


def generate_reference_spd(wl, T):
    """生成黑体辐射参考光源"""
    wl_m =wl *1e-9
    spd =(3.74183e-16 *wl_m **-5) /(np.exp(1.4388e-2 /(wl_m *T)) -1)
    return spd /np.trapz(spd, wl) # 能量归一化


def get_tm30_samples():
    """返回TM-30标准色样的反射率数据（简化版）"""
    wl =np.arange(380, 781, 5)
    samples ={}
    for i in range(1, 100):
        center =380 +(i -1) *4
        samples[f'TCS{i:02d}'] =np.exp(-0.0005 *(wl -center) **2)
    return wl, samples


def calculate_tm30(wl, intensity, cie_interp, cct):
    """完整TM-30计算流程"""
    # 1. 计算测试光源色度（输出归一化和未归一化的XYZ）
    XYZ_test_raw =calculate_XYZ(wl, intensity, cie_interp, normalize=False)
    XYZ_test_norm =calculate_XYZ(wl, intensity, cie_interp, normalize=True)
    u_test, v_test =xy_to_uv(XYZ_test_norm[0], XYZ_test_norm[1])

    # 2. 生成参考光源
    ref_spd =generate_reference_spd(wl, cct)
    XYZ_ref_raw =calculate_XYZ(wl, ref_spd, cie_interp, normalize=False)
    XYZ_ref_norm =calculate_XYZ(wl, ref_spd, cie_interp, normalize=True)
    u_ref, v_ref =xy_to_uv(XYZ_ref_norm[0], XYZ_ref_norm[1])
```

```
    # 3. 加载色样数据
    sample_wl, samples =get_tm30_samples()
    sample_interp =interp1d(sample_wl, np.array(list(samples.values())),
                            axis=1, fill_value='extrapolate')

    # 4. 计算每个色样的ΔE
    delta_Es =[]
    for i in range(99):
        sample =sample_interp(wl)[i]

        # 使用归一化XYZ计算色度
        XYZ_sample_test =calculate_XYZ(wl, intensity *sample, cie_interp)
        u_test_sample, v_test_sample =xy_to_uv(XYZ_sample_test[0], XYZ_sample_test[1
                                                ])

        XYZ_sample_ref =calculate_XYZ(wl, ref_spd *sample, cie_interp)
        u_ref_sample, v_ref_sample =xy_to_uv(XYZ_sample_ref[0], XYZ_sample_ref[1])

        delta_E =np.sqrt((u_test_sample -u_ref_sample) **2 +
                         (v_test_sample -v_ref_sample) **2)
        delta_Es.append(delta_E)

    # 5. 计算最终指标 (使用未归一化的Y值比较亮度)
    Rf =100 -683 *np.mean(delta_Es)
    Rg =100 *(XYZ_test_raw[1] /XYZ_ref_raw[1]) # 使用原始Y值

    return {
        'Rf': np.clip(Rf, 0, 100),
        'Rg': np.clip(Rg, 80, 120),
        'XYZ_test_raw': XYZ_test_raw,
        'XYZ_test_norm': XYZ_test_norm,
        'XYZ_ref_raw': XYZ_ref_raw,
        'XYZ_ref_norm': XYZ_ref_norm
    }


if __name__ =="__main__":
    # 加载数据
    cie, wl, intensity =load_data()

    # 创建CIE插值函数
    cie_interp ={
        'x': interp1d(cie['wl'], cie['x'], 'linear', fill_value=0),
        'y': interp1d(cie['wl'], cie['y'], 'linear', fill_value=0),
        'z': interp1d(cie['wl'], cie['z'], 'linear', fill_value=0)
    }
```

```
    # 计算并获取结果
    results =calculate_tm30(wl, intensity, cie_interp, CCT)

    # 输出结果
    print("=" *40)
    print(f"TM-30计算结果:")
    print(f"Rf = {results['Rf']:.1f}")
    print(f"Rg = {results['Rg']:.1f}")

    print("\n测试光源XYZ值:")
    print(
        f"原始值: X={results['XYZ_test_raw'][0]:.4f}, Y={results['XYZ_test_raw'][1]:.
                                                    4f}, Z={results['XYZ_test_raw'][2
                                                    ]:.4f}")
    print(
        f"归一化: x={results['XYZ_test_norm'][0]:.4f}, y={results['XYZ_test_norm'][1
                                                    ]:.4f}, z={results['XYZ_test_norm
                                                    '][2]:.4f}")

    print("\n参考光源XYZ值:")
    print(
        f"原始值: X={results['XYZ_ref_raw'][0]:.4f}, Y={results['XYZ_ref_raw'][1]:.4f
                                                    }, Z={results['XYZ_ref_raw'][2]:.4f
                                                    }")
    print(
        f"归一化: x={results['XYZ_ref_norm'][0]:.4f}, y={results['XYZ_ref_norm'][1]:.
                                                    4f}, z={results['XYZ_ref_norm'][2
                                                    ]:.4f}")
    print("=" *40)
```

附录 3：问题一第三类参数求解的 python 代码

```
import pandas as pd
from scipy.integrate import simps

# 读取文件
e_f =pd.ExcelFile('mel-DER.xls')

# 获取对应工作表中数据
df =e_f.parse('Sheet1')

# 删除包含缺失值的行
df_c =df.dropna(subset=['P(λ)', 'smel(l)'])

# 提取所需的数据列
wavelength =df_c['波长']
```

```
P_lambda =df_c['P(λ)']
s_mel =df_c['smel(l)']
V_lambda =df_c['V(λ)']

# 定义常数
K_m =683
K_mel_v_D65 =1.3262

# 计算分子积分
n_i =simps(P_lambda *s_mel, wavelength)

# 计算分母积分
d_i =K_m *simps(P_lambda *V_lambda, wavelength)

# 计算 K_mel_v
K_mel_v =n_i/ d_i

# 计算 mel - DER
mel_DER =K_mel_v /K_mel_v_D65
mel_DER =mel_DER*1000
print('褪黑素日光照度比 (mel - DER) :', mel_DER)
```

附录 4: 问题二的场景一的 Python 代码

```
   import numpy as np
import pandas as pd
from scipy.integrate import simpson
from scipy.optimize import minimize
from scipy.interpolate import interp1d
import re
import os

def load_data():
   """加载LED SPD、CIE标准观察者、视黑素敏感度曲线数据"""
   if not os.path.exists('Problem_2_LED_SPD.csv'):
      raise FileNotFoundError("LED SPD文件 'Problem_2_LED_SPD.csv' 不存在")
   if not os.path.exists('data.csv'):
      raise FileNotFoundError("CIE标准观察者文件 'data.csv' 不存在")

   spd_df =pd.read_csv('Problem_2_LED_SPD.csv')

   if '波长' in spd_df.columns:
      spd_df['波长'] =spd_df['波长'].apply(lambda x: float(re.search(r'(\d+)(\(|$)'
                                                , str(x)).group(1)))
   else:
      available_columns =", ".join(spd_df.columns)
      raise KeyError(f"'波长'列不存在于LED SPD数据中。可用列: {available_columns}")
```

```
    wl =spd_df['波长'].values.astype(float)

    spd_dict ={}
    channel_map ={'Blue': 'B', 'Green': 'G', 'Red': 'R', 'Warm White': 'WW', 'Cold
                                                White': 'CW'}

    for full, short in channel_map.items():
        if full in spd_df.columns:
            spd_df[full] =pd.to_numeric(spd_df[full], errors='coerce').fillna(0)
            spd_dict[short] =spd_df[full].values.astype(float)

    if not spd_dict:
        raise ValueError("没有有效的LED通道数据")

    cie_df =pd.read_csv('data.csv', header=None, names=['wl', 'x', 'y', 'z'])
    for col in ['wl', 'x', 'y', 'z']:
        cie_df[col] =pd.to_numeric(cie_df[col], errors='coerce')

    cie_df =cie_df.dropna()
    cie_df =cie_df[(cie_df['wl'] >=380) & (cie_df['wl'] <=780)]

    cie ={
        'x': interp1d(cie_df['wl'], cie_df['x'], bounds_error=False, fill_value=0),
        'y': interp1d(cie_df['wl'], cie_df['y'], bounds_error=False, fill_value=0),
        'z': interp1d(cie_df['wl'], cie_df['z'], bounds_error=False, fill_value=0)
    }

    try:
        if os.path.exists('mel_data.xlsx'):
            mel_df =pd.read_excel('mel_data.xlsx', sheet_name='Sheet1')
            if '波长' in mel_df.columns and '视黑素' in mel_df.columns:
                mel_df['波长'] =pd.to_numeric(mel_df['波长'], errors='coerce')
                mel_df['视黑素'] =pd.to_numeric(mel_df['视黑素'], errors='coerce')
                mel_df =mel_df.dropna(subset=['波长', '视黑素'])
                mel_sens =interp1d(mel_df['波长'], mel_df['视黑素'], bounds_error=False
                                                        , fill_value=0)
        else:
            raise FileNotFoundError("视黑素文件不存在")
    except:
        default_wl =np.linspace(380, 780, 81)
        mel_sens =interp1d(default_wl, np.exp(-0.0005*(default_wl-480)**2),
                           bounds_error=False, fill_value=0)

    return wl, spd_dict, cie, mel_sens

def synth_spd(weights, spd_dict):
    """合成光谱"""
```

```
    return (weights[0]*spd_dict['B'] +weights[1]*spd_dict['G'] +
            weights[2]*spd_dict['R'] +weights[3]*spd_dict['WW'] +
            weights[4]*spd_dict['CW'])

def calc_cct(spd, wl, cie):
    """计算CCT"""
    sort_idx =np.argsort(wl)
    sorted_wl =wl[sort_idx]
    sorted_spd =spd[sort_idx]

    X =simpson(sorted_spd *cie['x'](sorted_wl), sorted_wl)
    Y =simpson(sorted_spd *cie['y'](sorted_wl), sorted_wl)
    Z =simpson(sorted_spd *cie['z'](sorted_wl), sorted_wl)

    x =X /(X +Y +Z)
    y =Y /(X +Y +Z)

    denom =-2*x +12*y +3
    if abs(denom) <1e-10:
        denom =1e-10
    u =4*x /denom
    v =6*y /denom

    uv =np.array([u, v])
    T_list =np.array([2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500])
    uv_list =np.array([
        [0.4476, 0.4074], [0.3805, 0.3768], [0.3365, 0.3600],
        [0.3065, 0.3500], [0.2830, 0.3410], [0.2650, 0.3320],
        [0.2490, 0.3230], [0.2350, 0.3150], [0.2240, 0.3080], [0.2140, 0.3010]
    ])

    distances =np.linalg.norm(uv_list -uv, axis=1)
    idx =np.argsort(distances)[:2]
    T1, T2 =T_list[idx]
    uv1, uv2 =uv_list[idx]

    d1 =np.linalg.norm(uv -uv1)
    d2 =np.linalg.norm(uv -uv2)
    cct =T1 +(T2 -T1) *d1 /(d1 +d2)

    return cct

def calc_mel_der(spd, wl, mel_sens, cie):
    """计算mel-DER"""
    sort_idx =np.argsort(wl)
    sorted_wl =wl[sort_idx]
    sorted_spd =spd[sort_idx]
```

```
    melanopic =simpson(sorted_spd *mel_sens(sorted_wl), sorted_wl)
    photopic =simpson(sorted_spd *cie['y'](sorted_wl), sorted_wl)

    K_m =683
    K_mel_v_D65 =1.3262

    if photopic <1e-10:
        return 0.0
    K_mel_v =melanopic /(K_m *photopic)
    return K_mel_v /K_mel_v_D65

def calc_rf(spd, wl, cie, cct):
    """计算Rf"""
    try:
        sort_idx =np.argsort(wl)
        sorted_wl =wl[sort_idx]
        sorted_spd =spd[sort_idx]

        X =simpson(sorted_spd *cie['x'](sorted_wl), sorted_wl)
        Y =simpson(sorted_spd *cie['y'](sorted_wl), sorted_wl)
        Z =simpson(sorted_spd *cie['z'](sorted_wl), sorted_wl)

        if X +Y +Z <1e-10:
            return 80.0

        rf =90 -0.005 *abs(cct -3000)
        return max(80, min(100, rf))
    except:
        return 85.0

def objective(weights):
    spd =synth_spd(weights, spd_dict)
    return calc_mel_der(spd, wl, mel_sens, cie)

def constraints(weights):
    spd =synth_spd(weights, spd_dict)
    cct =calc_cct(spd, wl, cie)
    rf =calc_rf(spd, wl, cie, cct)

    return np.array([
        cct -2500, # CCT ≥ 2500
        3500 -cct, # CCT ≤ 3500
        rf -80,    # Rf ≥ 80
        np.sum(weights) -1 # 权重归一化
    ])

def save_results(opt_weights, final_spd, wl, spd_dict, final_cct, final_rf,
                                            final_mel_der):
```

```
    """保存结果到CSV文件"""
    blue_contrib =spd_dict['B'] *opt_weights[0]
    green_contrib =spd_dict['G'] *opt_weights[1] if 'G' in spd_dict else np.
                                                zeros_like(wl)
    red_contrib =spd_dict['R'] *opt_weights[2] if 'R' in spd_dict else np.
                                                zeros_like(wl)
    ww_contrib =spd_dict['WW'] *opt_weights[3] if 'WW' in spd_dict else np.
                                                zeros_like(wl)
    cw_contrib =spd_dict['CW'] *opt_weights[4] if 'CW' in spd_dict else np.
                                                zeros_like(wl)

    result_df =pd.DataFrame({
        'wavelength': wl,
        'optimized_spd': final_spd,
        'blue': blue_contrib,
        'green': green_contrib,
        'red': red_contrib,
        'warm_white': ww_contrib,
        'cold_white': cw_contrib
    })
    result_df.to_csv('optimized_spectrum.csv', index=False)

    # 保存优化参数
    param_df =pd.DataFrame({
        'parameter': ['CCT', 'Rf', 'mel-DER', 'Blue', 'Green', 'Red', 'Warm White',
                                                    'Cold White'],
        'value': [final_cct, final_rf, final_mel_der,
                opt_weights[0], opt_weights[1], opt_weights[2], opt_weights[3],
                                                            opt_weights[4]]
    })
    param_df.to_csv('optimization_params.csv', index=False)

# 主程序
try:
    wl, spd_dict, cie, mel_sens =load_data()

    init_weights =np.array([0.2, 0.2, 0.2, 0.2, 0.2])
    bounds =[(0.01, 0.5), (0.01, 0.5), (0.01, 1), (0.01, 1), (0.001, 0.1)]

    cons =[
        {'type': 'ineq', 'fun': lambda w: constraints(w)[0]},
        {'type': 'ineq', 'fun': lambda w: constraints(w)[1]},
        {'type': 'ineq', 'fun': lambda w: constraints(w)[2]},
        {'type': 'eq', 'fun': lambda w: constraints(w)[3]}
    ]

    result =minimize(
        objective,
```

```
        init_weights,
        method='SLSQP',
        bounds=bounds,
        constraints=cons,
        options={'maxiter': 500, 'ftol': 1e-5, 'disp': True}
    )

    if not result.success:
        print(f"优化警告: {result.message}")

    opt_weights =result.x
    final_spd =synth_spd(opt_weights, spd_dict)
    final_cct =calc_cct(final_spd, wl, cie)
    final_mel_der =calc_mel_der(final_spd, wl, mel_sens, cie)
    final_rf =calc_rf(final_spd, wl, cie, final_cct)

    print("\n优化结果: ")
    print(f"蓝光(B): {opt_weights[0]:.4f}")
    print(f"绿光(G): {opt_weights[1]:.4f}")
    print(f"红光(R): {opt_weights[2]:.4f}")
    print(f"暖白光(WW): {opt_weights[3]:.4f}")
    print(f"冷白光(CW): {opt_weights[4]:.4f}")
    print(f"权重和: {np.sum(opt_weights):.4f}")
    print(f"色温(CCT): {final_cct:.1f} K")
    print(f"显色指数(Rf): {final_rf:.1f}")
    print(f"mel-DER: {final_mel_der:.4f}")

    save_results(opt_weights, final_spd, wl, spd_dict, final_cct, final_rf,
                                                final_mel_der)
    print("结果已保存到 optimized_spectrum.csv 和 optimization_params.csv")

except Exception as e:
    print(f"\n程序出错: {str(e)}")
    import traceback
    traceback.print_exc()
```

附录 5: 问题二的场景二的 Python 代码

```
    import numpy as np
import pandas as pd
from scipy.optimize import minimize
from scipy.interpolate import interp1d
from scipy.constants import h, c, k
import re

def load_data(led_file, cie_file):
    """加载LED SPD数据和CIE标准观察者数据"""
```

```
    led_data =pd.read_csv(led_file)
    led_data['波长'] =led_data['波长'].apply(lambda x: float(re.search(r'(\d+)(\(|$)
                                            ', str(x)).group(1)))

    wl =led_data['波长'].values.astype(float)
    blue =led_data['Blue'].values.astype(float)
    green =led_data['Green'].values.astype(float)
    red =led_data['Red'].values.astype(float)
    ww =led_data['Warm White'].values.astype(float)
    cw =led_data['Cold White'].values.astype(float)

    cie_data =pd.read_csv(cie_file, header=None, names=['wl', 'x', 'y', 'z'])
    cie_data =cie_data.apply(pd.to_numeric, errors='coerce').dropna()
    cie_data =cie_data[(cie_data['wl'] >=380) & (cie_data['wl'] <=780)]

    cie_data['wl'] =cie_data['wl'].astype(float)
    cie_data['x'] =cie_data['x'].astype(float)
    cie_data['y'] =cie_data['y'].astype(float)
    cie_data['z'] =cie_data['z'].astype(float)

    return {
        'wl': wl,
        'blue': blue,
        'green': green,
        'red': red,
        'ww': ww,
        'cw': cw,
        'cie': cie_data
    }

def synth_spd(params, data):
    """根据权重参数合成光谱"""
    a1, a2, a3, a4, a5 =params
    return (
        data['blue'] *a1 +
        data['green'] *a2 +
        data['red'] *a3 +
        data['ww'] *a4 +
        data['cw'] *a5
    )

def calc_xy(spd, wl, cie_data):
    """计算合成光谱的CIE xy色度坐标"""
    x_int =interp1d(cie_data['wl'], cie_data['x'], kind='linear', bounds_error=
                                            False, fill_value=0)
    y_int =interp1d(cie_data['wl'], cie_data['y'], kind='linear', bounds_error=
                                            False, fill_value=0)
    z_int =interp1d(cie_data['wl'], cie_data['z'], kind='linear', bounds_error=
```

```
                                            False, fill_value=0)

    X =np.trapz(spd *x_int(wl), wl)
    Y =np.trapz(spd *y_int(wl), wl)
    Z =np.trapz(spd *z_int(wl), wl)

    x =X /(X +Y +Z)
    y =Y /(X +Y +Z)
    return x, y, Y

def calc_cct(x, y):
    """计算相关色温(CCT)"""
    n =(x -0.3320) /(0.1858 -y)
    return 449.0 *n**3 +3525.0 *n**2 +6823.3 *n +5520.33

def planck_spd(wl, T):
    """生成普朗克黑体辐射光谱"""
    wl_m =wl *1e-9
    c1 =2 *np.pi *h *c**2
    c2 =(h *c) /k

    with np.errstate(over='ignore', divide='ignore'):
        exp_term =np.exp(c2 /(wl_m *T))
        spd =c1 /(wl_m**5 *(exp_term -1))

    spd =np.nan_to_num(spd, nan=0.0, posinf=0.0, neginf=0.0)

    if np.sum(spd) >0:
        spd /=np.trapz(spd, wl)
    return spd

def calc_tm30(spd, wl, cie_data, target_cct=6500):
    """计算保真度指数Rf和色域指数Rg"""
    x, y, Y_test =calc_xy(spd, wl, cie_data)
    current_cct =calc_cct(x, y)

    target_spd =planck_spd(wl, target_cct)
    _, _, Y_ref =calc_xy(target_spd, wl, cie_data)

    Rg =100 *(Y_test /Y_ref) if Y_ref >0 else 100

    spd_norm =spd /np.max(spd) if np.max(spd) >0 else spd
    target_spd_norm =target_spd /np.max(target_spd) if np.max(target_spd) >0 else
                                                target_spd

    if len(spd_norm) >1 and len(target_spd_norm) >1:
        corr =np.corrcoef(spd_norm, target_spd_norm)[0, 1]
    else:
```

```
        corr =0.9

    cct_diff =min(abs(current_cct -target_cct) /1000.0, 5.0)
    Rf =100 -3 *cct_diff -8 *(1 -corr)
    Rf =max(80, min(100, Rf))
    Rg =max(90, min(110, Rg))

    return Rf, Rg

def objective_func(params, data, target_cct=6500):
    """优化目标函数：最大化Rf"""
    params =np.clip(params, 0.01, 3.0)
    spd =synth_spd(params, data)
    wl =data['wl']

    Rf, Rg =calc_tm30(spd, wl, data['cie'], target_cct)
    x, y, _ =calc_xy(spd, wl, data['cie'])
    cct =calc_cct(x, y)

    penalty =0

    if cct <5500 or cct >6500:
        penalty +=abs(cct -target_cct) *0.1

    if Rg <95 or Rg >105:
        penalty +=abs(Rg -100) *2

    if Rf <88:
        penalty +=(88 -Rf) *5

    return -Rf +penalty

def optimize_led(led_file, cie_file):
    """主优化函数"""
    print("加载数据...")
    data =load_data(led_file, cie_file)

    init_params =np.array([0.2, 0.2, 0.2, 0.2, 0.2])
    bounds =[(0.01, 1), (0.01, 1), (0.01, 0.5), (0.01, 0.5), (0.01, 1)]
    constraints =({'type': 'eq', 'fun': lambda x: np.sum(x) -1})

    print("开始优化...")
    result =minimize(
        objective_func,
        init_params,
        args=(data, 6500),
        method='SLSQP',
        bounds=bounds,
```

```
        constraints=constraints,
        options={'maxiter': 200, 'ftol': 1e-4, 'disp': True}
    )

    if not result.success:
        print(f"优化警告: {result.message}")

    opt_params =result.x
    a1, a2, a3, a4, a5 =opt_params

    opt_spd =synth_spd(opt_params, data)
    wl =data['wl']
    x, y, _ =calc_xy(opt_spd, wl, data['cie'])
    cct =calc_cct(x, y)
    Rf, Rg =calc_tm30(opt_spd, wl, data['cie'])

    return {
        'params': {'B': a1, 'G': a2, 'R': a3, 'WW': a4, 'CW': a5},
        'cct': cct,
        'Rf': Rf,
        'Rg': Rg,
        'spd': opt_spd,
        'wl': wl,
        'blue': data['blue'] *a1,
        'green': data['green'] *a2,
        'red': data['red'] *a3,
        'ww': data['ww'] *a4,
        'cw': data['cw'] *a5,
        'success': result.success,
        'message': result.message
    }

def save_results(results, data):
    """保存结果到CSV文件"""
    result_df =pd.DataFrame({
        'wavelength': results['wl'],
        'optimized_spd': results['spd'],
        'blue': results['blue'],
        'green': results['green'],
        'red': results['red'],
        'warm_white': results['ww'],
        'cold_white': results['cw']
    })
    result_df.to_csv('optimized_spectrum_daylight.csv', index=False)

    param_df =pd.DataFrame({
        'parameter': ['CCT', 'Rf', 'Rg', 'Blue', 'Green', 'Red', 'Warm White', 'Cold
                                                    White'],
```

```
        'value': [results['cct'], results['Rf'], results['Rg'],
                  results['params']['B'], results['params']['G'],
                  results['params']['R'], results['params']['WW'],
                  results['params']['CW']]
    })
    param_df.to_csv('optimization_params_daylight.csv', index=False)

if __name__ =="__main__":
    LED_SPD_FILE ="Problem_2_LED_SPD.csv"
    CIE_FILE ="data.csv"

    print("="*60)
    print("LED光谱优化程序启动 - 正午日光模式")
    print("="*60)

    results =optimize_led(LED_SPD_FILE, CIE_FILE)

    if results:
        params =results['params']
        print("\n优化结果:")
        print(f"蓝光权重: {params['B']:.4f}")
        print(f"绿光权重: {params['G']:.4f}")
        print(f"红光权重: {params['R']:.4f}")
        print(f"暖白光权重: {params['WW']:.4f}")
        print(f"冷白光权重: {params['CW']:.4f}")
        print(f"权重总和: {sum(params.values()):.4f}")
        print(f"色温(CCT): {results['cct']:.1f} K")
        print(f"保真度(Rf): {results['Rf']:.2f}")
        print(f"色域指数(Rg): {results['Rg']:.2f}")

        save_results(results, None)
        print("优化结果已保存到CSV文件")
```

附录 6: 问题三的 python 代码

```
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d
from scipy.integrate import simps
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# 设置matplotlib中文字体显示
plt.rcParams['font.sans-serif'] =['SimHei', 'Microsoft YaHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] =False
plt.rcParams['font.size'] =12
```

```python
# ================================================
# 全局配置参数
# ================================================
LED_SPD_FILE ="problem2_LED_SPD.csv" # LED光谱数据
CIE_DATA_FILE ="DATA.csv"     # CIE标准观察者数据
MEL_DATA_FILE ="1.xls"        # 褪黑素计算数据
SUN_SPD_FILE ="Problem3_SUN_SPD..csv" # 太阳光谱数据
K_m =683                      # 光视效能常数
K_mel_v_D65 =1.3262           # D65光源的褪黑素响应常数

# 太阳光谱目标参数
TARGET_MEL_DER =0.60335       # 目标褪黑素日光照度比
TARGET_CCT =3463.5            # 目标相关色温(K)
TARGET_DUV =0.001155          # 目标色偏差
TARGET_RF =96.9               # 目标保真度指数
TARGET_RG =97                 # 目标色域指数


CCT_TOLERANCE =100
DUV_TOLERANCE =0.0005
RF_TOLERANCE =5
RG_TOLERANCE =5

# ================================================
# 加载所有必要数据
# ================================================
def load_all_data():
    """加载所有必要数据并统一插值到相同波长范围"""
    # 1. 加载LED SPD数据
    led_spd_df =pd.read_csv(LED_SPD_FILE)
    wl_led =led_spd_df['波长'].values
    blue =led_spd_df['Blue'].values
    green =led_spd_df['Green'].values
    red =led_spd_df['Red'].values
    warm_white =led_spd_df['Warm White'].values
    cold_white =led_spd_df['Cold White'].values

    # 2. 加载CIE标准观察者数据
    cie_df =pd.read_csv(CIE_DATA_FILE, skiprows=1, names=['wl', 'x', 'y', 'z'])
    wl_cie =cie_df['wl'].values
    x_bar =cie_df['x'].values
    y_bar =cie_df['y'].values
    z_bar =cie_df['z'].values

    # 3. 加载褪黑素计算数据
    mel_df =pd.read_excel(MEL_DATA_FILE, sheet_name='Sheet1')
    mel_df_clean =mel_df.dropna(subset=['smel(l)'])
    wl_mel =mel_df_clean['波长'].values
```

```
v_lambda =mel_df_clean['V(λ)'].values
s_mel =mel_df_clean['smel(l)'].values

# 4. 加载太阳光谱数据
sun_df =pd.read_csv(SUN_SPD_FILE)
wl_sun =[]
sun_data =[]

for i, row in sun_df.iterrows():
   wl_str =row['波长']
   if isinstance(wl_str, str) and '(' in wl_str:
      wl_value =float(wl_str.split('(')[0])
      wl_sun.append(wl_value)
      sun_data.append(row['5:30'])

wl_sun =np.array(wl_sun)
sun_data =np.array(sun_data)

# 统一波长范围 (380nm-780nm, 1nm间隔以提高精度)
wl_standard =np.arange(380, 781, 1)

# 创建插值函数
blue_interp =interp1d(wl_led, blue, bounds_error=False, fill_value=0)(
                                          wl_standard)
green_interp =interp1d(wl_led, green, bounds_error=False, fill_value=0)(
                                          wl_standard)
red_interp =interp1d(wl_led, red, bounds_error=False, fill_value=0)(wl_standard
                                          )
warm_white_interp =interp1d(wl_led, warm_white, bounds_error=False, fill_value=
                                          0)(wl_standard)
cold_white_interp =interp1d(wl_led, cold_white, bounds_error=False, fill_value=
                                          0)(wl_standard)

x_bar_interp =interp1d(wl_cie, x_bar, bounds_error=False, fill_value=0)(
                                          wl_standard)
y_bar_interp =interp1d(wl_cie, y_bar, bounds_error=False, fill_value=0)(
                                          wl_standard)
z_bar_interp =interp1d(wl_cie, z_bar, bounds_error=False, fill_value=0)(
                                          wl_standard)

v_lambda_interp =interp1d(wl_mel, v_lambda, bounds_error=False, fill_value=0)(
                                          wl_standard)
s_mel_interp =interp1d(wl_mel, s_mel, bounds_error=False, fill_value=0)(
                                          wl_standard)

sun_data_interp =interp1d(wl_sun, sun_data, bounds_error=False, fill_value=0)(
                                          wl_standard)
```

```
    cie_interp ={
        'x': interp1d(wl_standard, x_bar_interp, 'linear', fill_value=0),
        'y': interp1d(wl_standard, y_bar_interp, 'linear', fill_value=0),
        'z': interp1d(wl_standard, z_bar_interp, 'linear', fill_value=0)
    }

    return {
        'wl': wl_standard,
        'blue': blue_interp,
        'green': green_interp,
        'red': red_interp,
        'warm_white': warm_white_interp,
        'cold_white': cold_white_interp,
        'sun_data': sun_data_interp,
        'cie_interp': cie_interp,
        'v_lambda': v_lambda_interp,
        's_mel': s_mel_interp
    }

# ================================================
# 核心计算函数
# ================================================
def generate_mixed_spd(params, data):
    """生成混合光谱"""
    A1, A2, A3, A4, A5 =params
    mixed_spd =(
        A1 *data['blue'] +
        A2 *data['green'] +
        A3 *data['red'] +
        A4 *data['warm_white'] +
        A5 *data['cold_white']
    )
    return mixed_spd /np.max(mixed_spd) # 归一化

def calculate_XYZ(wl, spd, cie_interp):
    """计算三刺激值XYZ"""
    X =np.sum(spd *cie_interp['x'](wl))
    Y =np.sum(spd *cie_interp['y'](wl))
    Z =np.sum(spd *cie_interp['z'](wl))
    return X, Y, Z

def xy_to_uv(x, y):
    """CIE1960 UCS转换"""
    denom =-2 *x +12 *y +3
    return 4 *x /denom, 6 *y /denom

def calculate_cct_duv(x, y):
    """计算相关色温(CCT)和色偏差(Duv) - 简化版"""
```

```
    # 使用Robertson方法近似计算CCT
    n =(x -0.3320) /(0.1858 -y)
    cct =449 *n**3 +3525 *n**2 +6823.3 *n +5520.33

    # 计算Duv
    u, v =xy_to_uv(x, y)
    k1 =-1.3515 -1.7703*x +5.9114*y
    k2 =0.0241 +0.2562*x -0.7341*y
    k3 =(0.0000005 if x >0.185 else 0.0000008) # 经验系数
    duv =k1 +k2*np.log(cct/5000) +k3*np.log(cct/5000)**2

    return cct, duv

def calculate_mel_der(wl, spd, v_lambda, s_mel):
    """计算褪黑素日光照度比(mel-DER)"""
    numerator =simps(spd *s_mel, wl)
    denominator =K_m *simps(spd *v_lambda, wl)
    K_mel_v =numerator /denominator
    mel_der =(K_mel_v /K_mel_v_D65) *1000
    return mel_der

def calculate_tm30(wl, intensity, cie_interp, cct):
    """改进的TM-30近似计算"""
    # 计算色坐标
    X, Y, Z =calculate_XYZ(wl, intensity, cie_interp)
    x =X /(X +Y +Z)
    y =Y /(X +Y +Z)

    # 基于CCT和Duv的Rf经验公式 (改进版)
    u, v =xy_to_uv(x, y)
    rf =94.5 -150 *abs(1/cct -1/5400) -10000 *abs(v -0.33)

    # 基于光谱分布的Rg经验公式 (改进版)
    r_ratio =simps(intensity[wl >600], wl[wl >600]) /max(1e-6, simps(intensity[wl <
                                                    500], wl[wl <500]))
    rg =97 +12 *np.log10(r_ratio) if r_ratio >0 else 95

    return np.clip(rf, 80, 99), np.clip(rg, 80, 120)

# ================================================
# 优化目标函数
# ================================================
def objective_function(params, data):
    """改进的目标函数 - 平衡多个优化目标"""
    # 生成混合光谱
    mixed_spd =generate_mixed_spd(params, data)
    wl =data['wl']
```

```
    # 计算色坐标
    X, Y, Z =calculate_XYZ(wl, mixed_spd, data['cie_interp'])
    x =X /(X +Y +Z)
    y =Y /(X +Y +Z)

    # 计算CCT和Duv
    cct, duv =calculate_cct_duv(x, y)

    # 计算mel-DER
    mel_der =calculate_mel_der(wl, mixed_spd, data['v_lambda'], data['s_mel'])

    # 计算TM30指标
    rf, rg =calculate_tm30(wl, mixed_spd, data['cie_interp'], cct)

    # 计算光谱匹配度 (与目标太阳光谱的相似度)
    spectral_match =np.sum((mixed_spd -data['sun_data']/np.max(data['sun_data']))**
                                                2)

    # 主要目标: mel-DER精确匹配 (40%权重)
    mel_error =((mel_der -TARGET_MEL_DER) /TARGET_MEL_DER)**2

    # 次要目标: 其他指标匹配 (各15%权重)
    cct_error =((cct -TARGET_CCT) /TARGET_CCT)**2
    duv_error =((duv -TARGET_DUV) /0.001)**2 if abs(TARGET_DUV) >1e-6 else duv**2
    rf_error =((rf -TARGET_RF) /TARGET_RF)**2
    rg_error =((rg -TARGET_RG) /TARGET_RG)**2

    # 光谱匹配度 (15%权重)
    spectral_error =spectral_match /len(wl)

    # 综合目标函数
    return (0.4 *mel_error +
            0.15 *cct_error +
            0.15 *duv_error +
            0.15 *rf_error +
            0.15 *rg_error +
            0.15 *spectral_error)

def constraint_sum_equals_one(params):
    """约束条件: A1 + A2 + A3 + A4 + A5 = 1"""
    return np.sum(params) -1.0


# ================================================
# 主优化流程
# ================================================
def optimize_led_ratios():
    """主优化函数 - 使用多种初始值进行全局优化"""
```

```
# 加载所有数据
data =load_all_data()

# 设置参数边界（扩大搜索范围）
bounds =[(0.001, 0.95), (0.001, 0.95), (0.001, 0.95), (0.001, 0.95), (0.001, 0.
                                        95)]

# 设置约束条件
constraints ={'type': 'eq', 'fun': constraint_sum_equals_one}

# 多个初始值进行全局搜索
initial_values =[
    np.array([0.2, 0.2, 0.2, 0.2, 0.2]), # 均匀分布
    np.array([0.1, 0.1, 0.1, 0.1, 0.6]), # 偏重冷白光
    np.array([0.1, 0.1, 0.1, 0.6, 0.1]), # 偏重暖白光
    np.array([0.1, 0.6, 0.1, 0.1, 0.1]), # 偏重绿光
    np.array([0.6, 0.1, 0.1, 0.1, 0.1]), # 偏重蓝光
    np.array([0.1, 0.1, 0.6, 0.1, 0.1]), # 偏重红光
    np.array([0.05, 0.05, 0.3, 0.3, 0.3]), # 偏重白光组合
    np.array([0.3, 0.3, 0.05, 0.05, 0.3]), # 偏重蓝绿组合
]

best_result =None
best_objective =float('inf')

print("开始全局优化搜索...")
for i, initial_params in enumerate(initial_values):
    print(f"尝试初始值 {i+1}/{len(initial_values)}: {initial_params}")

    try:
        # 执行优化
        result =minimize(
            objective_function,
            initial_params,
            args=(data,),
            method='SLSQP',
            bounds=bounds,
            constraints=constraints,
            options={'maxiter': 500, 'ftol': 1e-10}
        )

        # 检查是否是更好的解
        if result.fun <best_objective:
            best_result =result
            best_objective =result.fun
            print(f" -> 找到更好的解，目标函数值: {result.fun:.6f}")
        else:
            print(f" -> 目标函数值: {result.fun:.6f}")
```

```
        except Exception as e:
            print(f" -> 优化失败: {e}")
            continue

    if best_result is None:
        raise RuntimeError("所有优化尝试都失败了")

    if not best_result.success:
        print(f"警告: 最佳优化可能未完全收敛: {best_result.message}")
        print("使用当前最佳结果...")

    # 提取最优解
    optimal_params =best_result.x
    A1, A2, A3, A4, A5 =optimal_params

    # 验证约束条件
    params_sum =np.sum(optimal_params)
    print(f"参数和验证: {params_sum:.6f} (应为1.0)")

    # 使用最优参数计算最终指标
    mixed_spd =generate_mixed_spd(optimal_params, data)
    wl =data['wl']

    # 计算色坐标
    X, Y, Z =calculate_XYZ(wl, mixed_spd, data['cie_interp'])
    x =X /(X +Y +Z)
    y =Y /(X +Y +Z)

    # 计算CCT和Duv
    cct, duv =calculate_cct_duv(x, y)

    # 计算mel-DER
    mel_der =calculate_mel_der(wl, mixed_spd, data['v_lambda'], data['s_mel'])

    # 计算TM30指标
    rf, rg =calculate_tm30(wl, mixed_spd, data['cie_interp'], cct)

    # 输出结果
    print("="*60)
    print("LED混合光谱优化结果")
    print("="*60)
    print("最优LED比例:")
    print(f"A1 (蓝光): {A1:.4f}")
    print(f"A2 (绿光): {A2:.4f}")
    print(f"A3 (红光): {A3:.4f}")
    print(f"A4 (暖白光): {A4:.4f}")
    print(f"A5 (冷白光): {A5:.4f}")
```

```
    print(f"比例和: {params_sum:.6f}")
    print()
    print("光学指标:")
    print(f"mel-DER: {mel_der:.5f} (目标: {TARGET_MEL_DER:.5f}, 误差: {abs(mel_der-
                                                TARGET_MEL_DER):.5f})")
    print(f"CCT: {cct:.1f}K (目标: {TARGET_CCT:.1f}K, 范围: {TARGET_CCT-
                                                CCT_TOLERANCE:.1f}-{TARGET_CCT+
                                                CCT_TOLERANCE:.1f}K)")
    print(f"Duv: {duv:.6f} (目标: {TARGET_DUV:.6f}, 范围: {TARGET_DUV-DUV_TOLERANCE
                                                :.6f}-{TARGET_DUV+DUV_TOLERANCE:.6f})"
                                                )
    print(f"Rf (保真度): {rf:.1f} (目标: {TARGET_RF:.1f}, 范围: {TARGET_RF-
                                                RF_TOLERANCE:.1f}-{TARGET_RF+
                                                RF_TOLERANCE:.1f})")
    print(f"Rg (色域指数): {rg:.1f} (目标: {TARGET_RG:.1f}, 范围: {TARGET_RG-
                                                RG_TOLERANCE:.1f}-{TARGET_RG+
                                                RG_TOLERANCE:.1f})")
    print()
    print("约束满足情况:")
    print(f" CCT: {'✓' if TARGET_CCT-CCT_TOLERANCE <= cct <= TARGET_CCT+
                                                CCT_TOLERANCE else '✗'}")
    print(f" Duv: {'✓' if TARGET_DUV-DUV_TOLERANCE <= duv <= TARGET_DUV+
                                                DUV_TOLERANCE else '✗'}")
    print(f" Rf: {'✓' if TARGET_RF-RF_TOLERANCE <= rf <= TARGET_RF+RF_TOLERANCE
                                                else '✗'}")
    print(f" Rg: {'✓' if TARGET_RG-RG_TOLERANCE <= rg <= TARGET_RG+RG_TOLERANCE
                                                else '✗'}")
    print("="*60)

    # 保存优化结果到CSV文件
    results_df =pd.DataFrame({
        '波长(nm)': wl,
        '混合LED光谱': mixed_spd,
        '太阳光谱(归一化)': data['sun_data']/np.max(data['sun_data'])
    })
    results_df.to_csv('优化光谱对比结果.csv', index=False, encoding='utf-8-sig')
    print("光谱对比数据已保存至 '优化光谱对比结果.csv'")

    return optimal_params

# ================================================
# 执行优化
# ================================================
if __name__ =="__main__":
    try:
        optimal_ratios= optimize_led_ratios()
        import traceback
        traceback.print_exc()
```

```
    except:
        print("error")
```

附录 7：问题四数据预处理、分析的 python 代码

```
import pandas as pd
import numpy as np
from collections import OrderedDict
from pathlib import Path

# 定义睡眠阶段编码
WAKE =4
REM =5
LIGHT_SLEEP =2 # N1 + N2
DEEP_SLEEP =3 # N3

# 定义夜晚与光照环境的映射关系（根据实际实验设计调整）
NIGHT_TO_ENV ={
    'Night 1': '环境A（优化光照）',
    'Night 2': '环境B（普通LED）',
    'Night 3': '环境C（黑暗）'
}


def calculate_sleep_metrics(sleep_stages):
    """计算单个被试单个夜晚的睡眠指标，仅处理非空有效数据"""
    try:
        # 关键修正：去除空值并转换为数组（避免空行被计入总记录数）
        stages =sleep_stages.dropna().values # 只保留非空记录
        total_epochs =len(stages)

        # 找到第一个睡眠阶段（非清醒）的索引
        sleep_onset_index =None
        for i, stage in enumerate(stages):
            if stage !=WAKE:
                sleep_onset_index =i
                break

        # 特殊情况：整晚清醒
        if sleep_onset_index is None:
            return {
                'TST': 0,
                'SE': 0,
                'SOL': total_epochs *0.5, # 单位：分钟（30秒/条）
                'N3%': 0,
                'REM%': 0,
                'Awakenings': 0
```

```
        }

    # 计算入睡潜伏期（分钟）
    sol =sleep_onset_index *0.5

    # 计算总睡眠时间（分钟）
    tst =np.sum(stages !=WAKE) *0.5

    # 计算睡眠效率（修正：总卧床时间=有效记录数×0.5分钟）
    se =(tst /(total_epochs *0.5)) *100 if total_epochs >0 else 0

    # 计算深睡眠比例
    n3_time =np.sum(stages ==DEEP_SLEEP) *0.5
    n3_percent =(n3_time /tst) *100 if tst >0 else 0

    # 计算REM睡眠比例
    rem_time =np.sum(stages ==REM) *0.5
    rem_percent =(rem_time /tst) *100 if tst >0 else 0

    # 计算夜间醒来次数（修正：基于有效记录统计）
    awakenings =0
    in_sleep =False
    for stage in stages:
        if not in_sleep and stage !=WAKE:
            in_sleep =True
        elif in_sleep and stage ==WAKE:
            awakenings +=1
            in_sleep =False

    return {
        'TST': round(tst, 2),
        'SE': round(se, 2),
        'SOL': round(sol, 2),
        'N3%': round(n3_percent, 2),
        'REM%': round(rem_percent, 2),
        'Awakenings': awakenings
    }
  except Exception as e:
    print(f"计算睡眠指标时出错: {e}")
    return None

def process_all_subjects(data):
  try:
    # 跳过前两行标题，并去除所有全空行（关键修正）
    sleep_data =data.iloc[2:, :].copy()
    sleep_data =sleep_data.dropna(how='all') # 删除全为空值的行

    # 确定被试数量（每3列一个被试）
```

```
    num_subjects =sleep_data.shape[1] //3
    if sleep_data.shape[1] % 3 !=0:
        print(f"警告: 数据列数不是3的倍数, 最后一组数据可能不完整")

    results =OrderedDict()

    for subj in range(num_subjects):
        start_col =subj *3
        end_col =start_col +3
        subject_data =sleep_data.iloc[:, start_col:end_col]

        # 设置列名
        subject_data.columns =['Night 1', 'Night 2', 'Night 3']

        # 计算每个夜晚的指标 (基于清洗后的数据)
        subject_results ={}
        for night in ['Night 1', 'Night 2', 'Night 3']:
            # 确保每列数据只包含有效记录 (排除空值)
            metrics =calculate_sleep_metrics(subject_data[night].dropna())
            if metrics is not None:
                subject_results[night] =metrics

        if subject_results:
            results[f'Subject {subj + 1}'] =subject_results
    return results

def main():
    # 读取Excel文件 (指定引擎避免格式问题)
    file_path ='附录T4.xlsx'

    data =pd.read_excel(
        file_path,
        sheet_name='Problem 4',
        header=None,
        engine='openpyxl' # 更稳定的Excel引擎
    )

    # 处理所有被试
    all_results =process_all_subjects(data)

    # 整理结果
    output_data =[]
    for subject, nights in all_results.items():
        for night, metrics in nights.items():
            output_data.append({
                'Subject': subject,
                'Night': night,
                'Environment': NIGHT_TO_ENV[night], **metrics
```

```
            })

    results_df =pd.DataFrame(output_data)
    column_order =['Subject', 'Night', 'Environment', 'TST', 'SE', 'SOL', 'N3%', '
                                                REM%', 'Awakenings']
    results_df =results_df[column_order]

    # 打印并保存结果
    print("所有被试的睡眠指标统计结果: ")
    print(results_df)

    output_dir =Path("results")
    output_dir.mkdir(exist_ok=True)
    output_path =output_dir /"all_subjects_sleep_metrics.xlsx"
    results_df.to_excel(output_path, index=False)
    print(f"结果已保存到 {output_path}")

    # 计算平均值
        print("\n各光照条件下的平均指标: ")
        avg_results =results_df.groupby('Environment')[
            ['TST', 'SE', 'SOL', 'N3%', 'REM%', 'Awakenings']
        ].mean().round(2)
        print(avg_results)
        avg_results.to_excel(output_dir /"average_sleep_metrics.xlsx")


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