所属类别	2025 年“华数杯”全国大学生数学建模竞赛	参赛编号
研究生		CM2503596

可调控生物节律的 LED 光源

摘要

发光二极管(LED)相比传统光源具有众多优势，近些年来在许多领域都得到了广泛的应用。发光二极管不仅在照明方面起到重大的作用，而且还能通过视网膜来影响人们的生理节律系统，调控褪黑激素的分泌，进而来调节我们的睡眠质量、认知功能与情绪状态。因此，拥有一种友好型的可调控的 LED 照明系统是非常重要的。本文以“可调控生物节律的 LED 光源”为核心，困绕四个问题 LED 光谱参数计算、光源合成、太阳光谱模拟、睡眠效果验证开展系统建模与量化分析。本文主要完成以下几方面工作：

问题一首先从 LED 光源的光谱功率分布(SPD)出发，建立了一套标准的 CIE 评价模型，将 SPD 转换为 XYZ 三刺激值，计算了相关色温(CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)等关健指标，计算结果显示，目标 SPD 的相关色温(CCT)：3913K，距离普朗克轨迹的距离(Duv)：-0.0009，保真度指数(Rf)：88.1，色域指数(Rg)：104.0，以及褪黑素日光照度比(mel-DER)：0.0894.

问题二在多个独立的发光通道(LED 芯片)组合的基础上，构建了通道强度与目标光谱之间关系，设计了一个多目标光谱优化求解方案。针对日间与夜间两种场景，成功得到了在不同场景下的人体生理节律光谱合成的方案。通过对 5 个通道红光、绿光、蓝光、暖白光和冷白光进行权重约束，可得日间场景最优权重组合为 Blue: 0.325，Green: 0.155，Red: 0.051，Warm White: 0.129，Cold White: 0.340，夜间场景最优权重组合为 Blue: 0.000，Green: 0.207，Red: 0.000，Warm White: 0.793，Cold White: 0.000。

问题三旨在结合附录数据文件给出的一个时间序列数据集，设计一个动态控制策略，使五通道 LED 模拟全天太阳光谱的节律变化。从早晨 8:30 至傍晚 19:30 的太阳光谱数据，提取各时间点的相关色温(CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)参数。通过求解各个时间点的通道权重，结果表明，在三个代表性时间点的合成光谱在关键参数上均与太阳光谱是高度吻合的。

问题四是对实际睡眠实验进行记录，通过人体睡眠实验验证优化光的实际效果。本文提取了包括总睡眠时间、睡眠效率、入睡潜伏期、深睡眠比例 N3%、REM 睡眠比例、夜间醒来次数等六项指标，结果显示仅 N3%在不同环境上存在显著差异，而其余指标在不同的环境上不存在显著差异。通过统计检验与显著性分析，发现优化光照 A 与普通光照 B 组在 N3%上无显著差异，而优化光照 A 组与黑暗环境 C 组在 N3%上存在显著差异，其差异幅度 Cohen's d 值为：1.206，差异幅度非常大。结果验证了优化光照对提高睡眠质量具有积极的意义。

综上，本文通过对 LED 光谱参数计算、光源合成、太阳光谱模拟、睡眠效果验证构建了一种友好型的可调控的 LED 照明系统，提出的一个科学又实用性的模型。

关键词：发光二极管(LED) 多通道优化 睡眠质量评估

一、问题背景与重述

1.1 问题背景

发光二极管(LED)作为一种高效、节能、环保的新型光源，近年来在许多领域得到了广泛应用。在照明领域，白光 LED 的效率已远超传统的白炽灯和荧光灯，成为最主要的照明光源，其优势在于可调节色温和光谱特性，从而来适应不同的照明需求。科学研究表明，光照不仅限于为人们提供视觉照明，特定波长的光照还会对人们的睡眠、认知与情绪产生影响。因此，如何设计和优化 LED 光源的光谱，使其在照明的同时，能够更好的调节人们的生理节律，避免生物钟的紊乱，这是需要更加关注的地方。

1.2 问题重述

问题 1：光源的光谱功率分布(SPD)是描述其物理特性的最基础数据，记录了光源在各个波长上的能量分布。为了综合评估 LED 光源在颜色质量与健康节律效应方面的性能，现给定一组 SPD 数据，每组数据为波长(nm)与对应的光谱功率(W/nm)，对五个参数建立标准化计算方法和数学模型建，求解下面三类共五个核心参数。颜色特性参数：相关色温(CCT)和距离普朗克轨迹的距离(Duv)，颜色还原参数：保真度指数(Rf)和色域指数(Rg)，生理节律效应参数：褪黑素日光照度比 (mel-DER)。

问题 2：需要利用这五个独立的 LED 通道(红光、绿光、蓝光、暖白光 WW、冷白光 CW)的光谱数据，通过调节各通道权重比例，合成满足两种场景需求的光谱。场景一：日间照明模式 ：模拟正午日光(CCT=6500K)，使得合成光谱的保真度指数(Rf)尽可能高(接近 100)。合成光谱的 CCT 在正午日光范围，6000±500K 以内。色域指数 Rg 在 95~105 之间，Rf>88 时可以保证颜色自然。计算并报告此模式下的视黑素日光效率比 (mel-DER)。场景二：夜间助眠模式，为了实现对人体生理节律的干扰最小，在低色温环境(合成光谱的 CCT=3000±500K)下，需要使合成光谱的视黑素日光效率比 (mel- DER) 尽可能低。同时，即使在助眠模式下，也应保证基本的颜色分辨能力，要求一般保真度指数(Rf)不低于 80。请针对以上两个场景，分别求出最优的通道权重组合，并展示合成光谱的关键参数。

问题 3：结合问题二中给出的五通道 LED，设计一个控制策略，使其合成的光谱能够在全天范围模拟给定的太阳光谱数据，使其具有相似的节律效果。并选取三个代表性时间点(早晨、正午、傍晚)，绘制合成光谱与目标太阳光谱的对比图，进行案例分析。

问题 4：该问题提供了一组临床睡眠实验数据，对实际睡眠实验进行记录，来评估我们设计的“优化光照”是否真正对改善人类睡眠质量有显著效果。提取了包括总睡眠时间、睡眠效率、入睡潜伏期、深睡眠比例、REM 睡眠比例、夜间醒来次数等多个指标，通过方差分析等统计方法，分析优化光照、普通光照、黑暗环境三种不同的光照对各项睡眠指标的影响是否存在显著性差异。最后来分析设计的“优化光照”相比于“普通光照”和“黑暗环境”，是否对睡眠质量产生了有益的改善。

二、模型假设

- 假设附录提供的光谱数据精度充分；
- 假设提供的太阳光谱数据视为真实户外自然光，忽略天气与地理差异；
- 假设问题四提供的实验交叉设计无遗留效应。

三、符号说明

数学符号	解释说明
λ	光谱的波长
CIE	三刺激值
A_{rest}	待测光源下围成的色域面积
A_{ref}	参考光源下围成的色域面积
$S_{mel}(\lambda)$	视网膜对蓝光的敏感度函数
SLSQP	序列最小二乘二次规划原理
w_i	对应光的权重

四、问题一的分析与模型建立

4.1 问题一分析

在本问题中，我们的目的是为了综合评估 LED 光源在颜色质量与健康节律效应方面的性能。这需要计算出 SPD 数据对应的五个特征参数值：相关色温 (CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)。这些参数分别描述了光源的颜色外观、还原物体真实色彩的能力，以及用于量化光照对人体生理节律的影响强度。我们的任务是基于给定的一组 LED 光源的 SPD 数据，构建数学模型来计算出 SPD 数据对应的五个特征参数值。

4.2 相关定义及计算流程

4.2.1 相关定义

1. CIE 三刺激值(CIE XYZ Tristimulus Values)

1931 年 CIE 根据前人实验，选择 700nm、546.1nm、435.8nm 做为 RGB 原色光，制定了 CIE-RGB 色彩体系，定义了 RGB 为匹配等能光谱色的三原色数量。根据 CIE 规定，对于 380nm 到 780nm 可见光波范围内，三刺激值可以通过以下公式计算：

$$X = K\int_{380}^{780} I(\lambda)\overline{x}(\lambda)d\lambda$$

$$Y = K\int_{380}^{780} I(\lambda)\overline{y}(\lambda)d\lambda$$

$$Z = K\int_{380}^{780} I(\lambda)\overline{z}(\lambda)d\lambda$$

其中λ为波长，在本题中积分范围为 380-780nm， $I(\lambda)$为是光源辐射的相对光谱功率分

布，$\bar{x}(\lambda)$、$\bar{y}(\lambda)$、$\bar{z}(\lambda)$是 CIE1931 标准色度观察者的光谱三刺激值通过 CIE XYZ 查表得到。

2.色品坐标

三原色各自的刺激值在三刺激值总量中所占的比例直接决定了一个单位颜色的色品，该三个比例值被定义为色品坐标，色品坐标的三个量之和为 1，计算公式如下：

$$x=\frac{X}{X+Y+Z}$$

$$y=\frac{Y}{X+Y+Z}$$

$$z=\frac{Z}{X+Y+Z}$$

3.相关色温(CCT)

因为色品坐标的三个量之和为 1，所以本质上只有两个独立量x和y， McCamy 于 1992 年提出了由色品坐标(x,y)，直接求相关色温 T 的简便算法[1]，计算公式如下：

$$CCT=-449n^3+3525n^2-6823.3n+5520.33$$

其中

$$n=\frac{x-0.3320}{0.1858-y}$$

4. 距离普朗克轨迹的距离(Duv)

距离普朗克轨迹的距离用来描述光源点的相对黑体轨迹位置[5]。颜色偏差的绝对值是与待测光源的黑体轨迹的距离，但颜色偏差为正当光源高于黑体轨迹时为正，当光源低于黑体轨迹时为负。计算公式如下：

$$Duv=\sqrt{(u-u_p)^2+(v-v_p)^2}\cdot sign(v-v_p)$$

其中

$$u=\frac{4x}{-2x+12y+3}$$

$$v=\frac{9y}{-2x+12y+3}$$

且(u_p,v_p)是普朗克线上最近的点，即黑体参考点，(u,v)为待测光源的 CIE 色品坐标。

5.色彩保真度指数(Rf)

色彩保真度代表色彩保真度指数衡量还原物体真实色彩的能力[2]，取值范围是 0-100，数值越高，还原的效果越好。计算公式如下：

$$R_f=100-6.73\times\overline{\triangle E}$$

其中

$$\triangle E_i = \sqrt{(J_i^{test} - J_i^{ref})^2 + (a_i^{test} - a_i^{ref})^2 + (b_i^{test} - b_i^{ref})^2}$$

$$\overline{\triangle E} = \frac{1}{N}\sum_{i=1}^{N} \triangle E_i$$

6. 色域指数(Rg)

色域指数反映光源对颜色的饱和能力，范围 60-140[2]，计算公式如下：

$$R_g = \frac{A_{test}}{A_{ref}} \times 100$$

其中 A_{rest}、A_{ref} 是 CIE 1976 UCS 色度图上，16 个色样色坐标点围成的多边形面积。A_{rest} 为待测光源下围成的色域面积，A_{ref} 为参考光源下的面积。

7. 褪黑素日光照度比(mel-DER)

mel-DER 是用来衡量光源对黑视素刺激效率的指标，与人体昼夜节律调节密切相关，理论取值范围：约 0.1 到 0.8[3]。mel-DER 是一个无量纲比值，表示相对于标准日光的黑视素刺激效率，计算公式如下：

$$mel - DER = \frac{\sum_{\lambda} I(\lambda) S_{mel}(\lambda) \Delta\lambda}{\sum_{\lambda} I_{D65}(\lambda) S_{mel}(\lambda) \Delta\lambda}$$

其中 $S_{mel}(\lambda)$ 表示视网膜对蓝光的敏感度函数。

4. 2. 2 计算流程

图 4-1 问题一指标计算流程图

4. 3 计算结果

4.3.1 光谱参数计算结果为：

1. 相关色温(CCT):3913K
2. 空间中与普朗克轨迹最近点的距离(Duv)：-0.0009
3. 保真度指数(Rf):88.1
4. 色域指数(Rg):104.0
5. 褪黑素日光照度比(mel-DER):0.0894

4.3.2 CIE 色度图

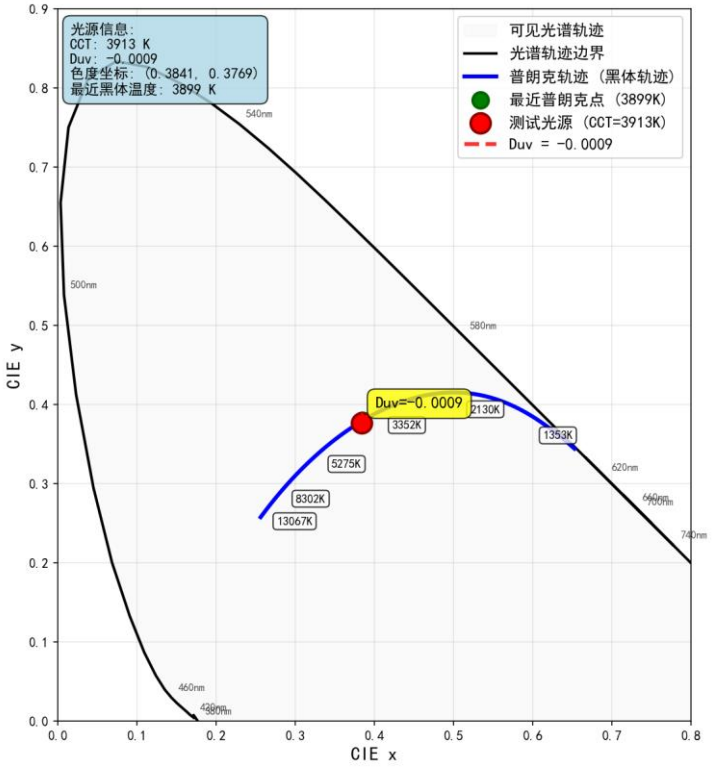


图 4-2 CIE 1931 色度图

五、问题二的分析与模型建立

5.1 问题二的分析

问题二是要求我们在已知的五个独立的 LED 通道(红光、绿光、蓝光、暖白光 WW、冷白光 CW)的光谱数据之间，添加线性加权，通过调节各通道权重比例，合成满足不同场景需求的光谱[6]。其中日间照明模式要保证颜色自然，使得合成光谱的保真度指数(Rf)尽可能高、合成光谱的 CCT 在正午日光范围、色域指数 Rg 在 95-105 之间、Rf>88。而夜间助眠模式需要使合成光谱的视黑素日光效率比(mel-DER)尽可能低、要满足低色温环境、应保证基本的颜色分辨能力。多通道 LED 光源的光谱合成是通过调节各通道的权重比例的动态调节[7]，是一个多目标，多约束的优化问题。

5.2 建立场景优化模型

5.2.1 模型建立

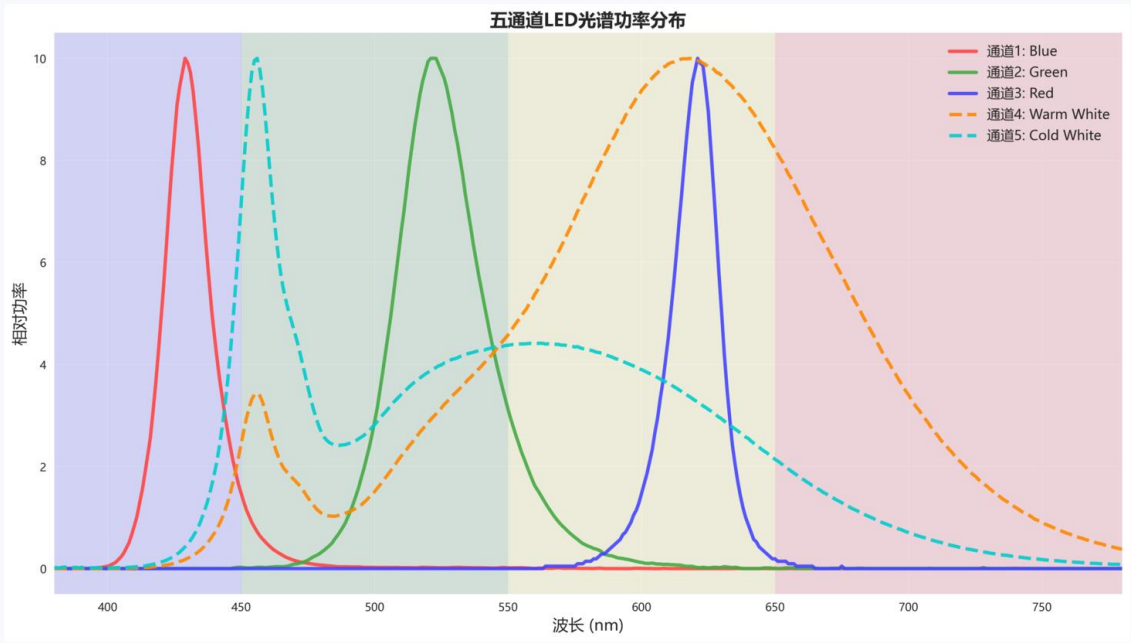


图 5-1 五通道 LED 光谱功率分布图

利用题目给出的五个独立 LED 通道的 SPD 数据，将数据可视化如图 5-1，可以看出五个通道在不同波长的光谱变化趋势，结合题目对两种照明模式的指数要求，建立函数模型，通过寻找最佳的权重组合来合成满足特定需求的光谱。我们设五个独立 LED 通道的光谱为S_i，$i=1,2,\cdots,5$，对应的权重系数为w_i，由此算出两个场景下的合成光谱功率，并计算两个场景下对应的核心指标值，合成光谱的功率分布公式如下：

$$SPD=\sum_{i=1}^{5}w_iS_i$$

场景一：日间照明模式

要求在模拟正午日光的条件下，使得合成光谱的保真度指数尽可能高，并计算此模式下的视黑素日光效率比(mel-DER)。结合题目对其余指标的数值要求，建立模型如下：

$$\max\ Rf$$

$$s.t.\begin{cases}5500\leqslant CCT\leqslant 6500\\ 95\leqslant Rg\leqslant 105\\ Rf>88\\ \sum_{i=1}^{5}w_i=1,\ \ w_i\geqslant 0\end{cases}$$

其中w_i为对应光的权重。

场景二：夜间助眠模式

要求在营造温馨的低色温环境下，需要使得合成光谱的视黑色素日光效率比尽可能低，同时，即使在助眠模式下，也要保证基本的颜色分辨能力，要求一般保真度指数不低于 80，建立模型如下：

$$\min \ \text{mel\_DER}$$

$$\begin{cases} 2500 \leqslant CCT \leqslant 3500 \\ Rf \geqslant 80 \\ \sum_{i=1}^{5} w_i = 1, w_i \geqslant 0 \end{cases}$$

5.2.2 算法逻辑以及原理

1.通过读取五通道 LED 的光谱数据，利用加权合成得到目标光谱。

2.问题二的数学模型是一个带线性等式约束且有着简单上下界约束的连续变量优化，符合 SLSQP 的应用场景，且相比遗传算法和粒子群算法更加高效，因此我们采用 SLSQP 优化算法作为求解器，分别针对日间和夜间场景，调整各通道权重，使合成光谱满足色温、色域、保真度和生物效应等性能指标。

3.计算核心参数包括：CCT(相关色温)、Duv(色度偏移)、Rf(保真度指数)、Rg(色域指数)、mel-DER(褪黑激素日光效率比)。其中: mel-DER(褪黑激素日光效率比)：表示光源对人体生物节律的影响，数值越高刺激越强，夜间应尽量降低[10]。

5.2.3 SLSQP(序列最小二乘二次规划)原理介绍

SLSQP 是一种用于求解非线性约束问题的迭代优化算法。它通过一系列二次规划子问题来近似原始非线性问题。在每次迭代中，它会对约束进行线性化，并使用目标函数的二次模型，然后求解得到的二次规划子问题来更新解。

一、目标

最小化受等式和不等式约束的非线性函数。

二、方法

1. 围绕当前点线性化约束。
2. 使用二次函数(使用梯度/Hessian 矩阵)近似目标函数。
3. 求解二次规划子问题以获得搜索方向。
4. 更新解并重复直至收敛。

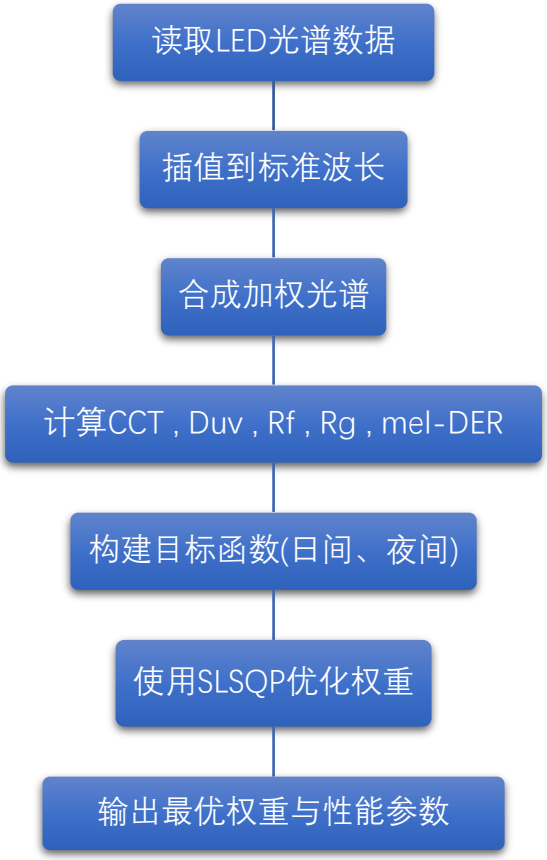
三、应用场景

- 同时受等式和不等式约束的非线性优化。
- 目标函数和约束条件平滑(可微分)的问题。
- 工程设计、参数拟合、资源配置、投资组合优化以及 LED 光谱权重优化。

四、优点

- 可处理等式和不等式约束。
- 适用于中小型问题。
- 广泛应用于科学和工程领域。

5.2.4 模型计算流程



5.3 计算结果

经过计算得到各通道最权重比例，合成满足不同场景需求的光谱，且得到在该光谱下的五个关键指标，结果对比图及合成效果图、色度图如图 5-2—图 5-6 所示。

- 日间最优的通道权重组合:

Blue: 0.325
Green: 0.155
Red: 0.051
Warm White: 0.129
Cold White: 0.340

- 关键指标值: CCT=5699K， Rf=94.3， Rg=95.0，Duv=-0.0069，mel-DER=0.004

- 夜间最优的通道权重组合:

Blue: 0.000
Green: 0.207
Red: 0.000
Warm White: 0.793
Cold White: 0.000

- 关键指标值: CCT=2711K， Rf=91.7， Rg=93.2， Duv=0.0143，mel-DER=0.002

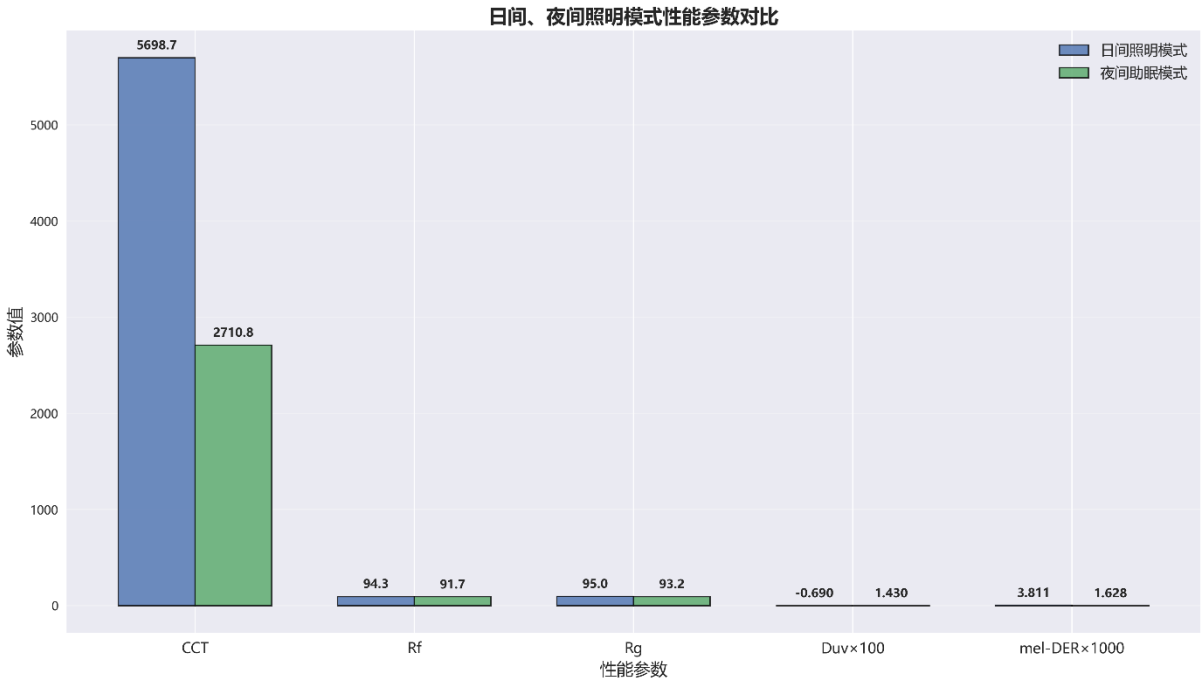


图 5-2 日间、夜间照明模式性能参数对比图

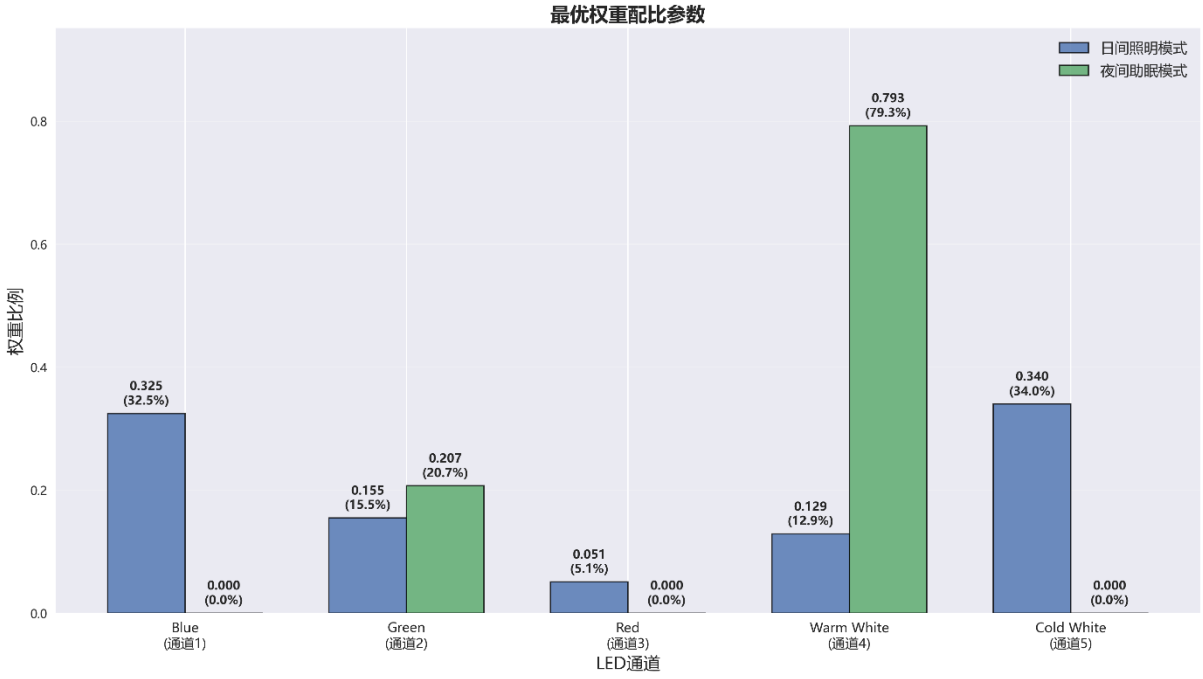


图 5-3 最优权重配比参数对比图

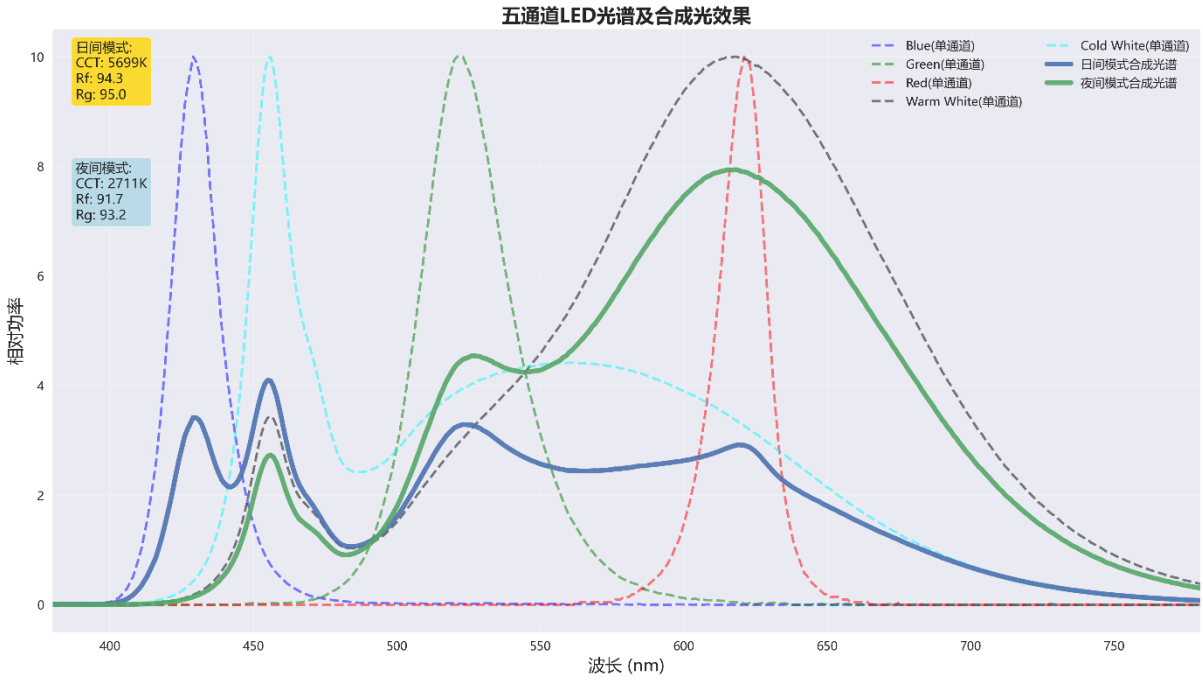


图 5-4 五通道 LED 光谱及合成效果图

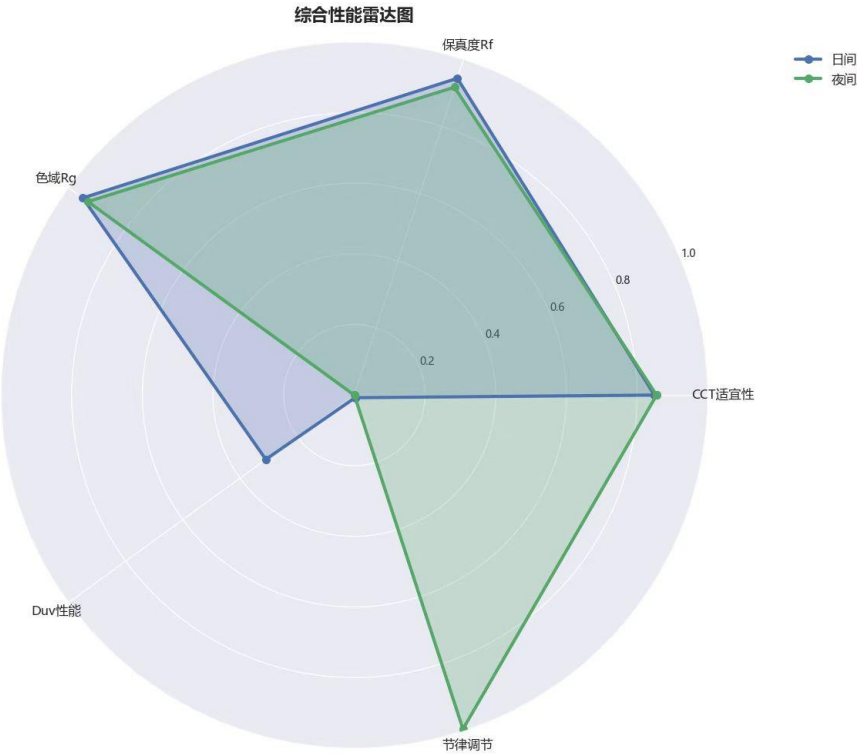


图 5-5 综合性能雷达图

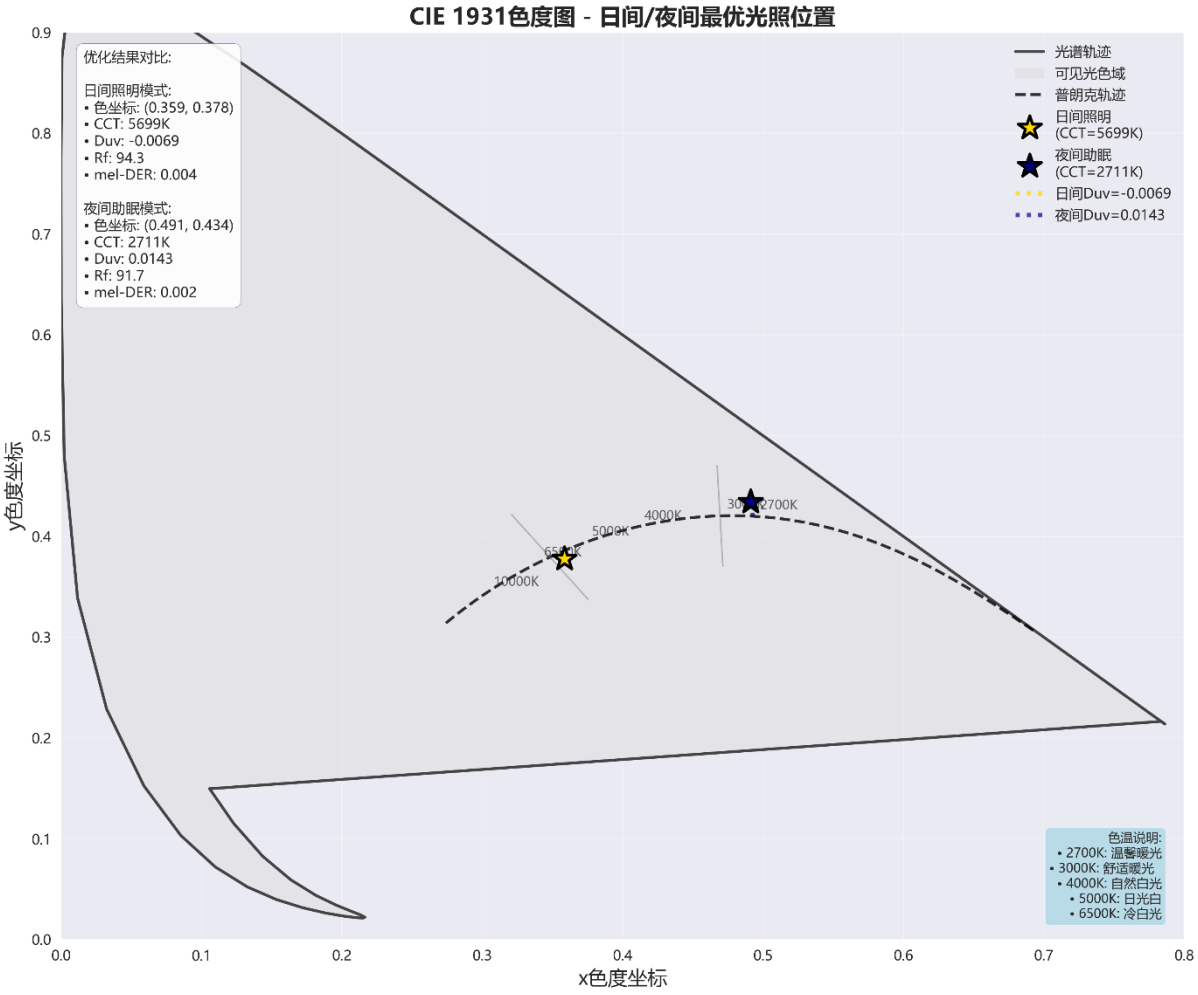


图 5-6 CIE 1931 色度图

六、问题三的分析与模型建立

6.1 问题三分析

问题三需要结合附录数据文件给出的一个时间序列数据集，基于问题二中给出的五通道 LED，设计一个控制策略，使得合成后的灯光也可满足自然光(太阳光)的对应参数[11]。附件给出早晨 5:30 至傍晚 19:30 的太阳光谱数据，每半小时测量一次日光 SPD，共 15 个时间节点，我们提取各时间点的相关色温(CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)参数，结合第二问的过程将单时刻的灯光合成策略拓展到全天连续照明控制策略，以确保其合成的光谱能够在全天范围模拟给定的太阳光谱数据，具有相似的节律效果[8]。

6.2 数据处理与模型建立

首先结合第一问给出的公式计算出在不同的时刻下问题三 SPD 各时间点数据对应的五个特征参数值，结果如下表 6-1 所示：

表 6-1 各时间点 SPD 特征参数值

	CCT	Duv	Rf	Rg	Mel-DER
05:30:00	5345	0.0026	93.8	101.5	0.0117
06:30:00	5321	0.0027	93.6	101.6	0.0115
07:30:00	5297	0.0027	93.5	101.7	0.0113
08:30:00	5272	0.0028	93.3	101.7	0.0112
09:30:00	5382	0.0024	94.1	101.3	0.012
10:30:00	5484	0.0021	94.7	101	0.0128
11:30:00	5579	0.0018	95.2	100.7	0.0136
12:30:00	5668	0.0015	95.7	100.4	0.0144
13:30:00	5585	0.0017	95.3	100.6	0.0136
14:30:00	5496	0.0019	94.7	100.9	0.0128
15:30:00	5400	0.0021	94.2	101.1	0.0119
16:30:00	5296	0.0023	93.5	101.4	0.0111
17:30:00	5155	0.002	92.8	101.6	0.0078
18:30:00	4836	0.0012	97.8	101.6	0.0044
19:30:00	3472	0.0011	95.3	103.4	0.001

上表给出了各时刻太阳光谱的五个指标具体数值，现设计一个控制策略，仿照第一问求出每一时刻的五参数(CCT,Duv,Rf,Rg,Mel-DER)，然后利用第二问中五通道 LED 灯光合成，使得合成后的灯光也可满足自然光(太阳光)的对应参数，考虑到要分析 15 个时刻(5:30-19:30)，故将每个参数的误差定为 20%，其中 CCT、Rg、Rf 三个参数的误差一直在 10%以下，见图 5-6，相当于对第二问的一个时刻改为同时控制 15 个时刻，即等价于如下多目标问题，算法仍使用 SLSQP，模型如下：

$$\min L(w) = \sum_{i=1}^{15} L_i(w)$$

其中$L_i(w)$为每个时刻的误差，

$$\begin{cases} L_i(w) = \alpha_1|\Delta CCT| + \alpha_2|\Delta Duv| + \alpha_3|\Delta Rf| + \alpha_4|\Delta Rg| + \alpha_5|\Delta mel - DER| \\ \Delta CCT = |CCT_{LED}(z) - CCT_{target}| \\ \Delta Duv = |Duw_{LED}(z) - Duv_{target}| \\ \Delta Rf = |Rf_{LED}(z) - Rf_{target}| \\ \Delta Rg = |Rg_{LED}(z) - Rg_{target}| \\ \Delta mel - DER = |DER_{LED}(z) - w_5 \cdot DER_{target}| \end{cases}$$

为保证五参数量级相近，权重设置如下：相关色温(CCT)：$\alpha_1 = 0.01$、距离普朗克轨迹的距离(Duv)：$\alpha_2 = 10$、保真度指数(Rf)：$\alpha_3 = 0.1$、色域指数(Rg)：$\alpha_4 = 0.1$、褪黑素日光照度比(mel-DER)：$\alpha_5 = 1$。

6.3 模型结果及对比

我们将合成光谱与目标太阳光谱节律效果进行对比首先五个特征指标在不同时间点对应的权重结果如下图 6-1 所示：

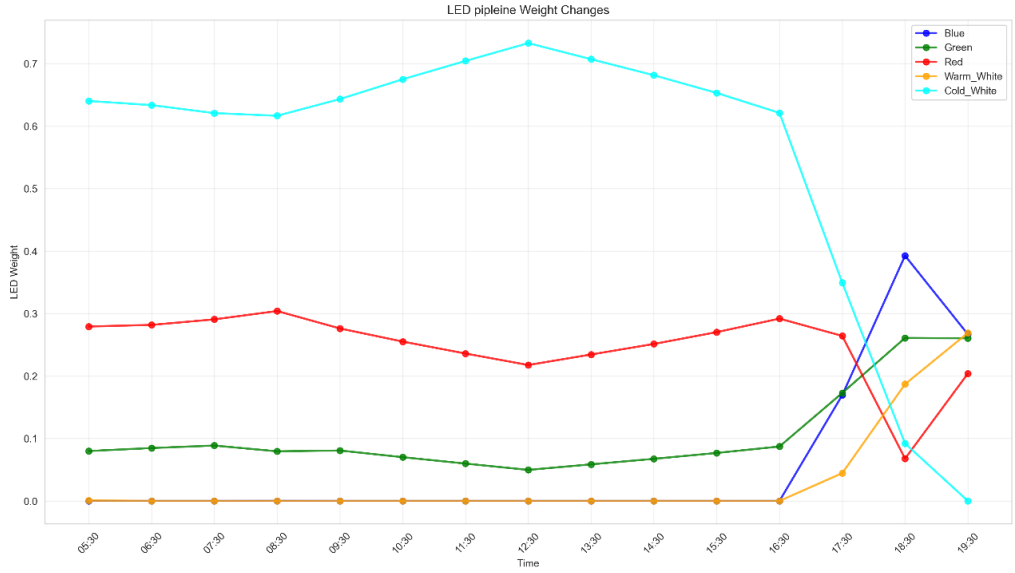


图 6-1 不同时间点对应权重折线图

选取 15 各时间点，制成了合成光谱与太阳光的对比图，结果如图 6-2 所示:

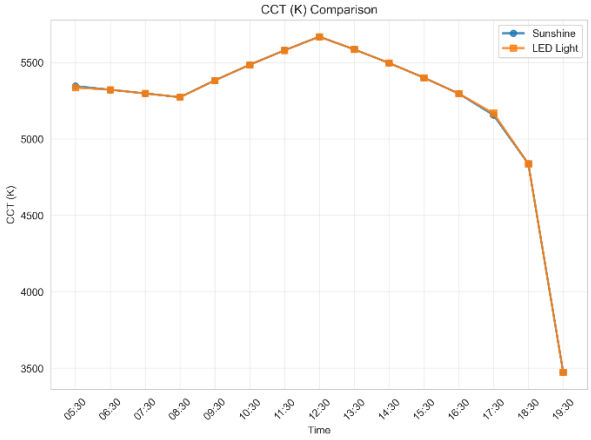


图 6-2(a) CCT 参数对比图

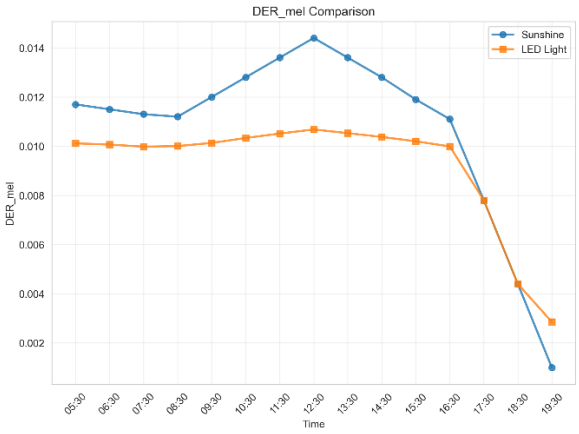


图 6-2(b) DER 参数对比图

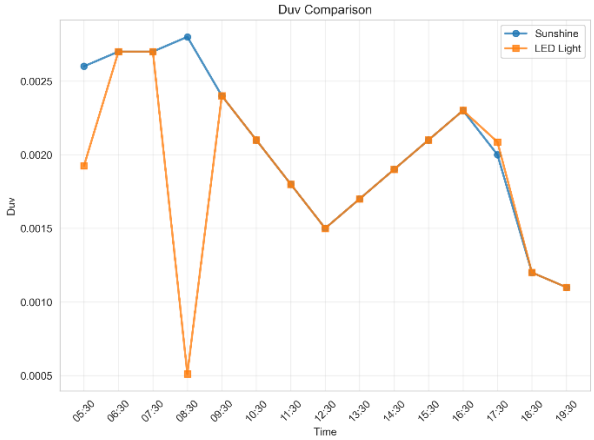


图 6-2(c) Duv 参数对比图

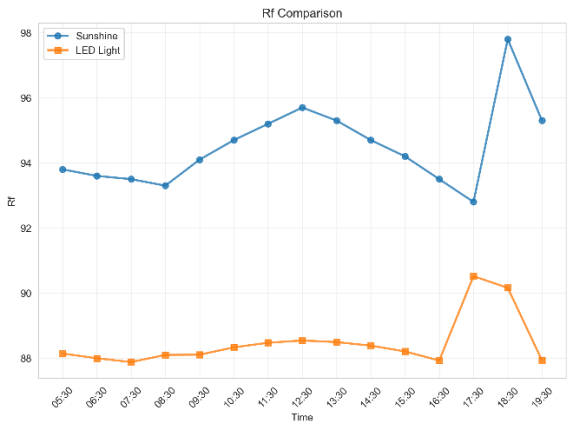


图 6-2(d) Rf 参数对比图

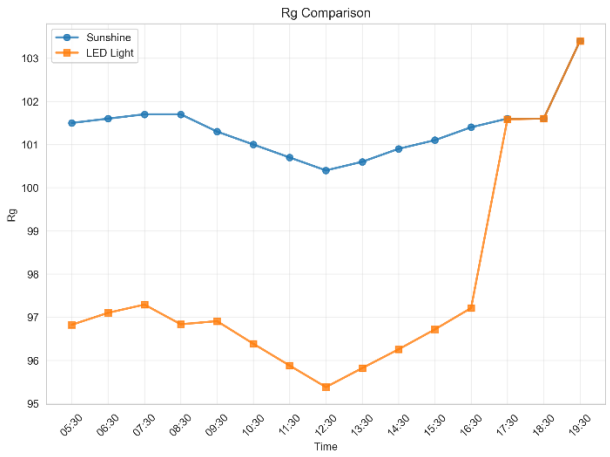


图 6-2(e) Rg 参数对比图

5 个参数随时间变化，结果如图 6-3 所示:

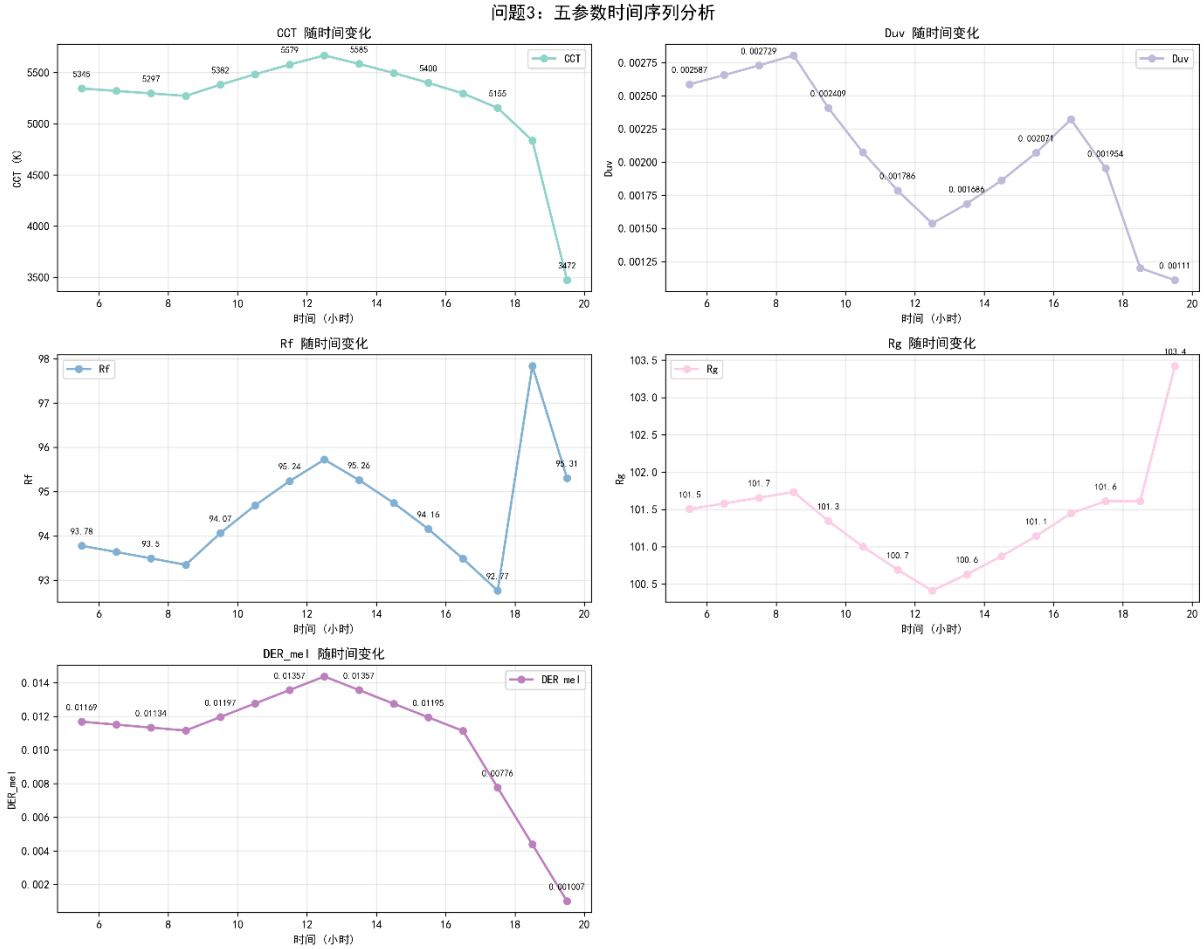


图 6-3 参数随时间变化结果图

由图 6-3 可以得到相关色温(CCT) 在 12:30 之前缓慢上升，在 12:30 达到顶峰，之后缓慢下降，在 17:30 之后迅速下降; 而距离普朗克轨迹的距离(Duv)在 5:30-12:30 缓慢下降，说明 LED 模拟越来越接近日光，在 12:30-16:30 上升，说明下午 LED 的模拟距离日光差距较大，但最大值也小于 0.003，说明模拟效果极好，Rf 虽有波动，但都在 93 以上，说明 LED 光的保真度很高，Rg 随时间变化波动，并在 12:30 达到最小值，但仍然 100 以上，色域指数覆盖超过标准. Mel-DER 变化趋势与 CCT 一致，符合模拟日光

的调律要求。

下面给出 5 个参数相关性分析热力图，结果如图 6-4 所示，发现 CCT 和 Mel-DER 相关性达到 0.93 同样符合上面的推论。

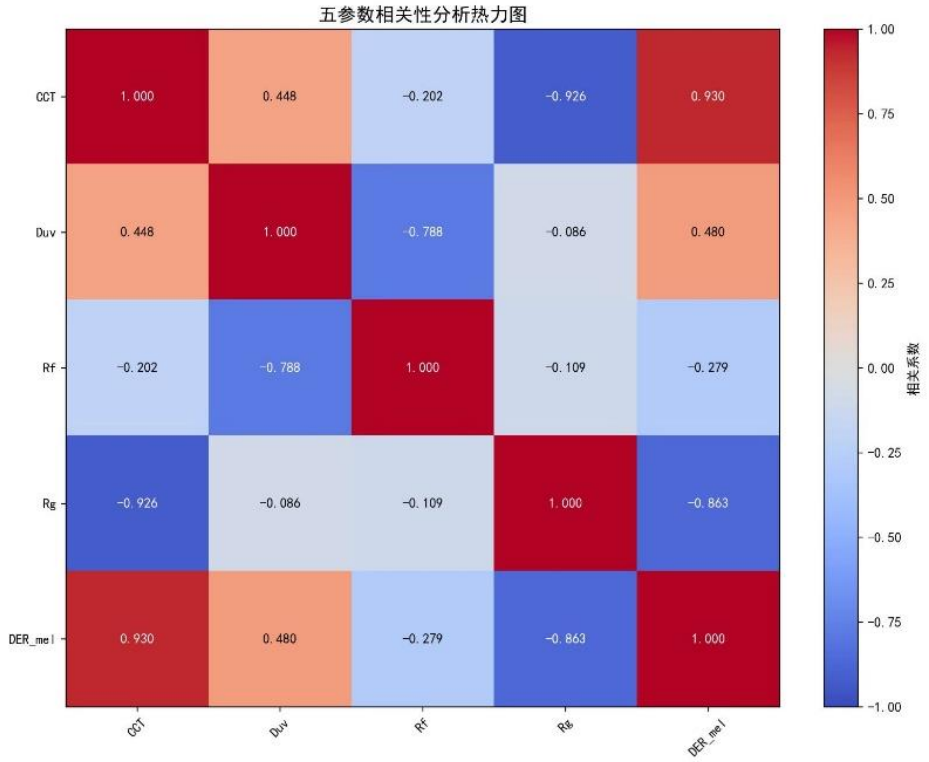


图 6-4 参数相关性分析热力图

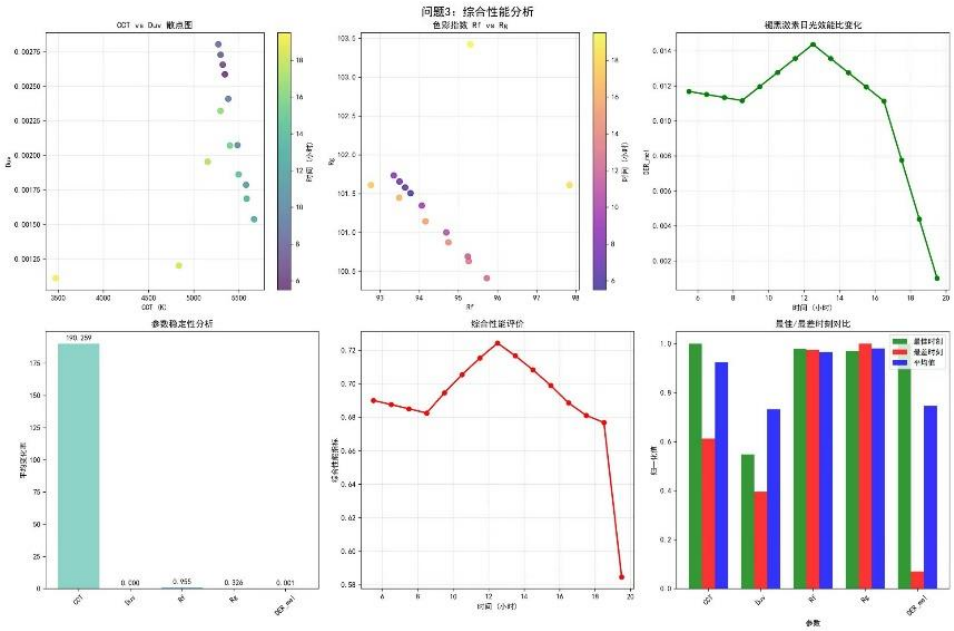


图 6-5 5 个参数的综合性能分析

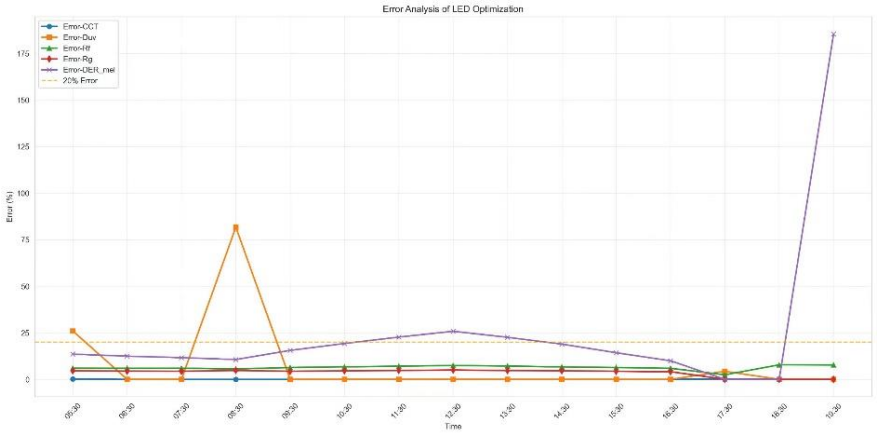


图 6-6 误差分析结果图

可以发现 CCT，Rg，Rg 误差始终保持在 10%以下，Duv 在 5:30，8:30 时误差超过 20%，Mel-DER 在 12:30 和 19:30 超过 20%，与图 5.4 综合考虑可得这是因为对应时刻日光的 CCT 变化过快，LED 模拟效果不好，最终可以得到的结果除 Mel-DER 外模拟都很好，Mel-DER 在 19:30 时误差最大，达到 175%。

6.4 模型的结果分析(优缺点)

一、模型优点

1. 物理与光学基础：模型基于 CIE 颜色匹配函数、普朗克黑体辐射理论、标准日光光谱等物理光学原理，确保参数计算的科学性和准确性。

2. 参数全面性：五参数（CCT、Duv、Rf、Rg、mel-DER）能够多维度评价光源性能，适应多种应用场景。

3. 优化算法：LED 权重优化采用约束优化（SLSQP），结合多初始猜测与目标函数加权，提升全局寻优能力，保证 LED 合成光谱与目标太阳光参数的接近性。

4. 数据处理鲁棒性：模型对输入数据进行多重清洗、异常值处理和插值，提升了对实际测量数据的适应能力。

5. 科学性强：参数计算与优化均有物理和生理基础，结果具备理论解释力。

6. 扩展性强：可灵活调整参数权重，适应不同应用需求；可扩展更多光源或参数。

二、 模型缺点与局限

1. 部分生理参数简化：如褪黑激素效应函数采用经验公式，未考虑个体差异和环境因素，存在一定简化。

2. 光谱插值误差：不同光源或测量设备的波长分辨率不一致时，插值可能引入误差。

3. 对于参数剧烈变化的拟合不好， 如 mel-DER 差较大。

七、问题四的分析与模型建立

7.1 问题四分析

通过前面三问的理论模型，设计并优化出了特定模式的光谱，现在要将理论光谱设计落到真实人体验证上。题目中提供了一组交叉实验，其中包含了 11 位健康被试，在睡前 2 小时，每位被试先后经历三种睡前光照环境，分别是：

环境 A：暴露于问题二中设计的“夜间助眠模式”的光谱下；

环境 B：暴露于一种普通的市售 LED 灯光下；

环境 C：处于严格的黑暗环境中。

根据提供的 11 位被试的整夜睡眠数据，数据由便携式睡眠监测仪采集，睡眠编码遵循美国睡眠医学会标准，以这些数据进行显著性差异分析来判断我们设计的优化光照是否对睡眠质量产生了有益的改善.

7.2 睡眠指标的定义与计算

根据美国睡眠医学会(AASM)的标准将睡眠分为了 5 个阶段：清醒(Wake)、REM 睡眠(Rapid Eye Movement)、N1 期睡眠(Stage N1)、N2 期睡眠(Stage N2)、N3 期睡眠(Stage N3， 即深睡眠/慢波睡眠)，其中 N1 期和 N2 期被合称为“浅睡眠”(Light Sleep)。

根据睡眠医学常用标准计算每个被试的以下关键指标：

1.总睡眠时间(Total Sleep Time，TST)：所有非清醒阶段的总时长，记：

$$TST = t_{N1} + t_{N2} + t_{N3} + t_{REM}$$

2.睡眠效率(Sleep Efficiency，SE)：衡量睡眠连续性的重要指标，计算公式为：

$$SE = \frac{TST}{t_{总卧床}} \times 100\%$$

其中$t_{总卧床}$指总卧床时间，是指从关灯入睡到最终醒来的总时长。

3.入睡潜伏期 (Sleep Onset Latency， SOL)：从关灯到首次进入任何睡眠阶段所需要的时间。

4. 深睡眠比例 (N3%)：深睡眠对于身体恢复至关重要，计算公式为：

$$N3\% = \frac{N3\text{期总时长}}{TST} \times 100\%$$

5. REM 睡眠比例 (REM%)：REM 睡眠与学习、记忆和情绪调节有关，计算公式为：

$$REM\% = \frac{REM\text{期总时长}}{TST} \times 100\%$$

6. 夜间醒来次数(Number of Awakenings)：睡眠开始后，清醒总次数。

7.2 指标计算与分析

根据计算公式，我们得出 11 位被试在三种环境下的各指标值如下表 7-1：

表 7-1 被试在三种环境下各指标统计

	TST	SE	SOL	N3%	REM%	NA
被试 1—A	310	86.95652	15	14.67742	20.48387	20
被试 1—B	312	90.04329	2	24.51923	23.23718	20
被试 1—C	288.5	82.07681	5	25.64991	25.64991	20
被试 2—A	322	65.78141	16	17.23602	32.14286	18
被试 2—B	348	75.24324	5	30.8908	29.74138	14
被试 2—C	347	91.67768	0	27.8098	23.63112	16
被试 3—A	360.5	91.84713	4	21.35922	26.21359	13
被试 3—B	260	71.33059	51	28.07692	20.19231	15

被试 3—C	259	68.24769	54.5	21.62162	24.13127	13
被试 4—A	334	83.70927	40.5	24.7006	33.98204	14
被试 4—B	386	94.37653	13	20.20725	42.35751	12
被试 4—C	347	87.40554	7	34.14986	21.61383	10
被试 5—A	397.5	93.30986	16	26.91824	22.51572	17
被试 5—B	430.5	93.99563	16	9.407666	36.12079	16
被试 5—C	384	84.3956	2	27.21354	18.09896	15
被试 6—A	382.5	89.68347	31	14.24837	15.29412	12
被试 6—B	358.5	74.60978	75	10.87866	21.75732	13
被试 6—C	365	91.70854	2.5	26.0274	12.46575	12
被试 7—A	371	84.31818	31	18.73315	26.68464	15
被试 7—B	401.5	88.33883	47.5	20.17435	28.01993	9
被试 7—C	379.5	94.28571	7	22.6614	7.509881	10
被试 8—A	381.5	89.449	31	12.71298	14.02359	14
被试 8—B	375.5	88.45701	2	22.63648	20.7723	18
被试 8—C	435	95.39474	2	25.97701	35.97701	13
被试 9—A	491	92.29323	6.5	19.7556	32.58656	23
被试 9—B	455.5	93.4359	13.5	20.19759	28.97914	26
被试 9—C	437	95.51913	1.5	15.90389	20.48055	23
被试 10—A	356	88.33747	7	17.41573	32.44382	15
被试 10—B	453.5	96.38682	5.5	10.80485	26.79162	12
被试 10—C	347	93.91069	9	18.15562	24.35159	9
被试 11—A	329.5	77.25674	23.5	21.39605	21.69954	12
被试 11—B	389.5	96.05425	8.5	15.53273	17.07317	12
被试 11—C	367.5	87.29216	24	26.12245	19.31973	21

将表格数据可视化得到如下六个指标详细图：

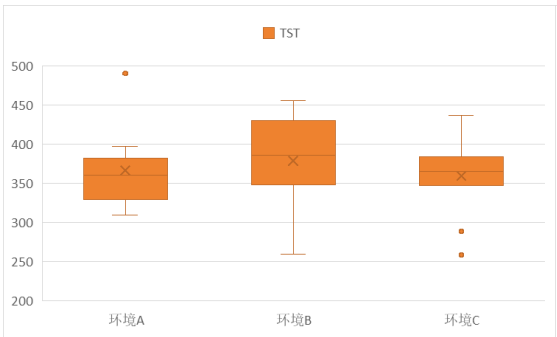


图 7-1 TST 箱型图

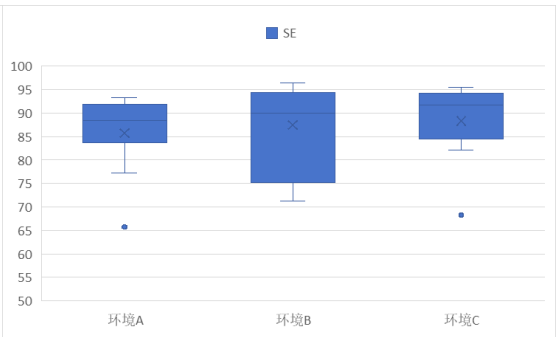


图 7-2 SE 箱型图

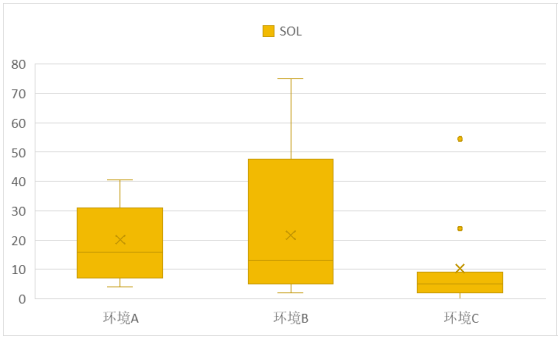


图 7-3 SOL 箱型图

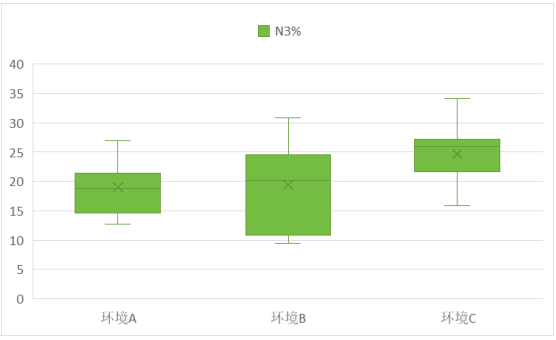


图 7-4 N3%箱型图

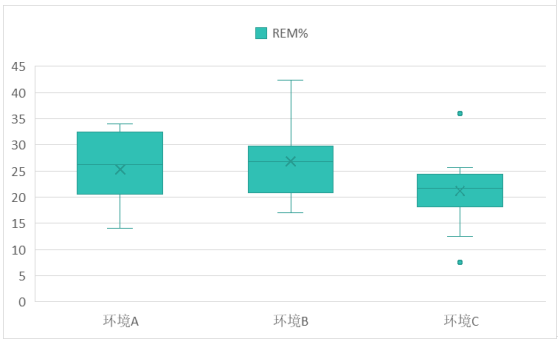


图 7-5 REM%箱型图

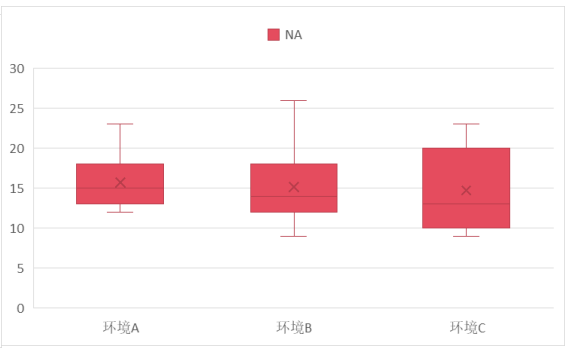


图 7-6 NA 箱型图

由箱型图可以初步观察出我们所优化的环境 A 在多个指标上相较于环境 B 和环境 C 表现出一定优势。

由总睡眠时间的箱型图可知，B 组平均值明显高于其他两组，A 组平均值高于 C 组，可以得出被试在普通环境下获得了更长的时间睡眠，但在环境 A 下箱型图的集中程度更好，离散程度较低，睡眠总时长指标更加稳定；由睡眠效率的箱型图可知，A 组的中位数和平均值均低于其他两组，但组间差距不大，说明在提高睡眠效率上优化环境的作用无法体现，受个体差异影响较大；由入睡潜伏期和深睡眠比例的箱型图可知，被试在全黑环境下进入睡眠的速度最快，A 组数据集中程度相对更好，减小了个体差异；由 REM 睡眠比例和夜间醒来次数的箱型图可知，A 组的夜间醒来次数的最大值相比之下最小，稳定性较好，REM 的睡眠比例较 B 组也有改善。整体来看我们设计的优化光照方案在一定程度和部分指标上改善了人们的睡眠质量，但可能受个体对光的敏感程度不同、个体产生褪黑素的能力不同，优化效果不够明显，直观的对比数据为后续的建模和检验奠定了一定的基础。

7.3 统计检验与显著性分析

根据数据的类型，我们通过单因素方差分析，对实现数据进行统计建模和显著性分析。单因素方差分析步骤如下：

1. 根据分类变量(X)对定量变量(Y)进行分组并检查其正态性，看数据的整体分布是否呈现正态分布;如果测试通过，则继续下一步。

2. 根据分类变量(X)对定量变量(Y)进行分组，进行方差齐性检验，看 P 值是否小于 0.05;如果 P 值大于 0.05，则使用方差分析来查看 P 值是否显著(P < 0.05)(理论上，数据必须通过正态性检验和方差齐性检验才能进行单因素方差分析;否则使用非参数检验，但在实际应用中没有必要那么严格)

3. 如果出现显著差异，我们可以根据平均值±标准差进行分析;否则表明不呈现差异。

4. 如果单因素方差分析呈现显著性，也可借助效应量化分析对差异性进行量化分析由此，我们利用 SPSS 得出结果如下表 7-2：

表 7-2 正态性检验和方差齐性检验

变量名	S-W 检验	K-S 检验	Levene 统计量	P
TST	0.978(0.737)	0.104(0.485)	0.203	0.818
SE	0.866(0.001***)	0.186(0.005***)	0.384	0.684
SOL	0.816(0.000***)	0.228(0.000***)	2.788	0.078*
N3%	0.984(0.891)	0.082(0.820)	1.475	0.245
REM%	0.99(0.987)	0.08(0.850)	0.09	0.914
NA	0.937(0.054*)	0.156(0.040**)	0.591	0.560

注：***、**、*分别代表 1%、5%、10%的显著性水平

上表展示了六个指标正态性检验和方差齐性的部分结果，利用 Shapiro-Wilk 检验进行正态分布检验，若呈现显著性(P<0.05)，则说明拒绝原假设(数据符合正态分布)，该数据不满足正态分布，反之则说明该数据满足正态分布，由表中 S-W 检验列的数据，SE、SOL 除外，显著性 P 值均大于 0.05，因此数据满足正态分布。对 SE、SOL 两个指标数据作正态性检验直方图，如下图：

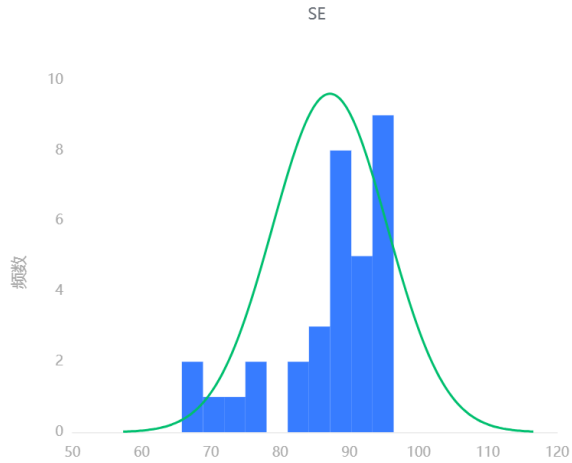


图 7-7 SE 正态检验直方图

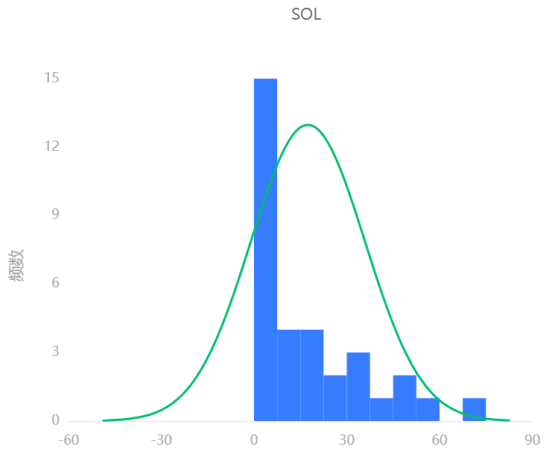


图 7-8 SOL 正态检验直方图

上面两图展示了定量变量 SE、SOL 数据正态性检验的结果，若正态图基本上呈现出钟形(中间高，两端低)， 则说明数据虽然不是绝对正态，但基本可接受为正态分布。

表中方差齐性的结果包含 Levene 检验结果、显著性 P 值。需分析每个分析项的 P 值是否显著(P<0.05)。若呈显著性，拒绝原假设(原假设：满足方差齐性)，则说明数据波动不一致，即说明方差不齐；反之则说明数据波动一致，说明数据满足方差齐性。所有 P 值均大于 0.05，所以数据满足方差齐性。

由上可得所用数据均符合正态分布且满足方差齐性，可以进行单因素方差分析。结果如表 7-3：

表 7-3 单因素方差分析结果

变量名	F 值	P 值	显著性结论
TST	0.363	0.698	不显著
SE	0.274	0.762	不显著
SOL	1.231	0.306	不显著
N3%	3.478	0.044**	显著
REM%	1.742	0.192	不显著
NA	0.145	0.866	不显著

注：***、**、*分别代表 1%、5%、10%的显著性水平

由于满足方差齐性，采用单样本方差检验，N3%的方差分析结果 P 值为 0.044**≤0.05，因此统计结果显著，说明不同的环境在 N3%上存在显著差异。其余指标在不同的环境上不存在显著差异。

同时，针对深睡眠比例的数据做了三组之间每两组的独立样本 T 检验，得出结论：由于满足方差齐性，采用独立样本 T 检验，显著性结果 P 值为 0.010**，因此统计结果显著，说明 A 组，C 组在 N3%上存在显著差异;其差异幅度 Cohen's d 值为：1.206，差异幅度非常大。结合箱型图，我们可以知道优化方案在提升深睡眠比例上有显著效果，让我们拥有更加连续、稳定的深度睡眠。

7.4 本章小结

通过本章的统计和分析，我们得到如下结论：我们设计的“优化光照”相比于“普通光照”和“黑暗环境”，对睡眠质量产生了一定的有益的改善，三组实验数据对比后可知，深睡眠比例在环 A 下有着显著的变化，特别是对比在严格处于黑暗环境的情况下，深睡眠比例有着显著提高。深睡眠对于身体恢复至关重要，如免疫系统巩固、记忆固化、生长激素分泌高峰、组织修复等，连续、稳定的深度睡眠比碎片化的深度睡眠更重要，不同的光照环境可能对褪黑素分泌的抑制程度不同，进而干扰我们的睡眠结构。所以，通过合理改变照明中的光谱成分，可以有效诱导更好更优质的深度睡眠。

然而，其余关键指标：总睡眠时间、睡眠效率、入睡潜伏期、REM 睡眠比例、夜间醒来次数，这些指标结构在不同光照环境下也有一定的改变，可能受限于被试个体差异较大、实验数据过少、被试者基础睡眠程度不同等客观因素，在统计结果中并未发现显著变化，光谱结构对于这几个指标的影响还需更深更远的研究。因此，未来还需更多的研究和设计来研究光谱对人们临床睡眠的影响，从而实现光照对人们睡眠质量的干预。

八、模型的优点与缺点

8.1 模型优点

(1) 模型很好的结合了实际中的问题，其中着重的考虑了生理节律、光谱的组成以及睡眠质量等比较关键的因素。比如在问题一中，从 LED 光源的光谱功率分布(SPD)出发，通过标准化对相关色温(CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)进行计算，成功量化了光照对生理节律的

影响强度。在问题三中，通过求解各个时间点的通道权重，设计一个动态控制策略，使五通道 LED 模拟全天太阳光谱的节律变化，合理的模拟了 LED 的合成效果。这些模拟贴合实际中的问题，具有很高的应用价值。

(2) 本文使用的算法方法具有计算效率高、模拟比较好等优点。例如，在问题二和问题三中，我们都应用了序列最小二乘二次规划原理，很好的寻找了最佳的权重组合来合成满足不同场景的需求，最优的通道组合能快速的得到。问题三用太阳光来模拟，很好的得到了不同时刻太阳光的动态运行。这些算法在我们建模中有很好的科学性与效率。

(3) 问题一建立了一套标准的 CIE 评价模型，通过标准化对相关色温(CCT)、距离普朗克轨迹的距离(Duv)、保真度指数(Rf)、色域指数(Rg)以及褪黑素日光照度比(mel-DER)五个参数进行计算，遵循了国际标准，确保了计算参数的科学性。问题四对实际睡眠实验是采用交叉设计来验证优化光的实际效果。本文提取了包括总睡眠时间、睡眠效率、入睡潜伏期等多个指标进行方差分析，对优化光照、普通光照、黑暗环境三种不同的光照下的实验结果进行了统计检验与显著性分析，具有统计学依据。

8.2 模型缺点

(1) 在实际的应用中，环境的温度与湿度、个体与个体之间的差异等因素可能也会成为影响生物节律与睡眠质量的重要因素。比如问题三的时间间隔为 1 小时，在很短时间内的光谱突然变换可能会被遗漏，没有考虑不同的气候对太阳光的影响，所以这些对结果都会有一定的影响。可能会一定程度上影响模型的全面性和预测的准确性。

(2) 本文提出的模型由于时间和数据限制，没有对不同的人群，比如青少年、老年人等，在不同的环境下进行深入的模拟验证。例如问题四实验的样本数比较少，又受限于被试个体差异较大、被试者基础睡眠程度不同等客观因素，在统计结果中并未发现显著变化，光谱结构对于这几个指标的影响还待研究。该模型的适用范围仍有待完善。

参考文献

[1]张浩，徐海松.光源相关色温算法的比较研究[J].光学仪器，2006，(01):54-58.
[2] Royer， Michael P. "Tutorial: Background and guidance for using the ANSI/IES TM-30 method for evaluating light source color rendition." Leukos 18.2 (2022): 191- 231.
[3] Schlangen， Luc JM， et al. "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”." Proceedings ofthe 29th Session of the CIE. CIE，2019. 114-118.
[4]常雪松.可调色温高显色 pc-LED 光源设计[D].大连工业大学，2023.
[5]李月.光源相关色温及色偏差计算方法研究[D].辽宁科技大学，2023.
[6]李宁. LED 混光的光视效能及显色性评估研究[D].北京大学，2014.
[7] Zhang， Jingjing， et al. "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 (2017): 193-198.
[8]郑峰,刘丽莹,刘小溪,石晓光,宦克为.多主色 LED 照明光源的相关色温调控[J].光学精密工程,2015,23(4):926-933.
[9]周锦荣,陈焕庭,周小方.白光 LED 色温的非线性动态预测模型[J].发光学报,2016,37(1):106-111.
[10] 谌江波,余建华,高亚飞,等. 超高显色指数和色温可调的 LED 白光照明光源研究[J].光学学报,2015,35(10):244-250.
[11] 高维惜,倪凯凯,林泽文,等. LED 模拟太阳光研究[J]. 照明工程学报,2015(1):80-83.

附录

由于代码数据过多，附录仅提供前两问代码，三四问具体代码见支撑材料。

问题一代码

```
import pandas as pd
import numpy as np
from scipy import interpolate
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import os
if not os.path.exists('output/Question1'):
    os.makedirs('output/Question1')

def load_data():
    """Load wavelength and intensity data from attachment-Q1.xlsx"""
    df = pd.read_excel('attachment-Q1.xlsx')
    wavelength = df.iloc[:, 0].values  # First column: wavelength (nm)
    intensity = df.iloc[:, 1].values   # Second column: intensity (mW/m²)
    return wavelength, intensity

def get_cie_color_matching_functions():
    """Get CIE 1931 2° standard observer color matching functions"""
    # Standard CIE 1931 2° observer data (380-780nm, 5nm intervals)
    wavelengths = np.arange(380, 785, 5)

    # CIE 1931 2° standard observer x̄(λ), ȳ(λ), z̄(λ)
    x_bar = np.array([
        0.0014, 0.0022, 0.0042, 0.0076, 0.0143, 0.0232, 0.0435, 0.0776,
        0.1344, 0.2148, 0.2839, 0.3285, 0.3483, 0.3481, 0.3362, 0.3187,
        0.2908, 0.2511, 0.1954, 0.1421, 0.0956, 0.0580, 0.0320, 0.0147,
        0.0049, 0.0024, 0.0093, 0.0291, 0.0633, 0.1096, 0.1655, 0.2257,
        0.2904, 0.3597, 0.4334, 0.5121, 0.5945, 0.6784, 0.7621, 0.8425,
        0.9163, 0.9786, 1.0263, 1.0567, 1.0622, 1.0456, 1.0026, 0.9384,
        0.8544, 0.7514, 0.6424, 0.5419, 0.4479, 0.3608, 0.2835, 0.2187,
        0.1649, 0.1212, 0.0874, 0.0636, 0.0468, 0.0329, 0.0227, 0.0158,
        0.0114, 0.0081, 0.0058, 0.0041, 0.0029, 0.0020, 0.0014, 0.0010,
        0.0007, 0.0005, 0.0003, 0.0002, 0.0002, 0.0001, 0.0001, 0.0001,
        0.0000
    ])

    y_bar = np.array([
        0.0000, 0.0001, 0.0001, 0.0002, 0.0004, 0.0006, 0.0012, 0.0022,
        0.0040, 0.0073, 0.0116, 0.0168, 0.0230, 0.0298, 0.0380, 0.0480,
```

```
        0.0600,  0.0739,  0.0910,  0.1126,  0.1390,  0.1693,  0.2080,  0.2586,
        0.3230,  0.4073,  0.5030,  0.6082,  0.7100,  0.7932,  0.8620,  0.9149,
        0.9540,  0.9803,  0.9950,  1.0000,  0.9950,  0.9786,  0.9520,  0.9154,
        0.8700,  0.8163,  0.7570,  0.6949,  0.6310,  0.5668,  0.5030,  0.4412,
        0.3810,  0.3210,  0.2650,  0.2170,  0.1750,  0.1382,  0.1070,  0.0816,
        0.0610,  0.0446,  0.0320,  0.0232,  0.0170,  0.0119,  0.0082,  0.0057,
        0.0041,  0.0029,  0.0021,  0.0015,  0.0010,  0.0007,  0.0005,  0.0004,
        0.0002,  0.0002,  0.0001,  0.0001,  0.0001,  0.0000,  0.0000,  0.0000,
        0.0000
    ])

    z_bar = np.array([
        0.0065,  0.0105,  0.0201,  0.0362,  0.0679,  0.1102,  0.2074,  0.3713,
        0.6456,  1.0391,  1.3856,  1.6230,  1.7471,  1.7826,  1.7721,  1.7441,
        1.6692,  1.5281,  1.2876,  1.0419,  0.8130,  0.6162,  0.4652,  0.3533,
        0.2720,  0.2123,  0.1582,  0.1117,  0.0782,  0.0573,  0.0422,  0.0298,
        0.0203,  0.0134,  0.0087,  0.0057,  0.0039,  0.0027,  0.0021,  0.0018,
        0.0017,  0.0014,  0.0011,  0.0010,  0.0008,  0.0006,  0.0003,  0.0002,
        0.0002,  0.0001,  0.0001,  0.0001,  0.0000,  0.0000,  0.0000,  0.0000,
        0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
        0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
        0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
        0.0000
    ])

    return wavelengths,  x_bar,  y_bar,  z_bar

def get_cri_test_color_samples():
    """Get CRI-14 test color sample reflectances (TM-30 color samples)"""
    # Simplified representation of TM-30 color samples
    # In practice,  you would load these from a standard database
    # Here we use a subset of typical color samples

    # Wavelengths for color sample data
    wavelengths = np.arange(380,  781,  5)

    # Sample reflectance data for 14 common color samples (simplified)
    # These are approximations - in practice you'd use official CIE data
    color_samples = []

    # Sample 1: Light greyish red
    sample1 = 0.2 + 0.3 * np.exp(-((wavelengths - 600) / 50)**2)
    color_samples.append(sample1)
```

```
    # Sample 2: Dark greyish yellow
    sample2 = 0.3 + 0.4 * np.exp(-((wavelengths - 570) / 40)**2)
    color_samples.append(sample2)

    # Sample 3: Strong yellow green
    sample3 = 0.2 + 0.5 * np.exp(-((wavelengths - 540) / 30)**2)
    color_samples.append(sample3)

    # Sample 4: Moderate yellowish green
    sample4 = 0.25 + 0.4 * np.exp(-((wavelengths - 520) / 35)**2)
    color_samples.append(sample4)

    # Sample 5: Light bluish green
    sample5 = 0.3 + 0.3 * np.exp(-((wavelengths - 500) / 40)**2)
    color_samples.append(sample5)

    # Sample 6: Light blue
    sample6 = 0.2 + 0.4 * np.exp(-((wavelengths - 480) / 30)**2)
    color_samples.append(sample6)

    # Sample 7: Light violet
    sample7 = 0.25 + 0.3 * np.exp(-((wavelengths - 450) / 25)**2)
    color_samples.append(sample7)

    # Sample 8: Light reddish purple
    sample8 = 0.2 + 0.25 * np.exp(-((wavelengths - 420) / 30)**2) + 0.2 *
np.exp(-((wavelengths - 650) / 40)**2)
    color_samples.append(sample8)

    # Additional samples 9-14
    for i in range(6):
        # Generate additional diverse color samples
        peak_wl = 400 + i * 60
        sample = 0.3 + 0.3 * np.exp(-((wavelengths - peak_wl) / 35)**2)
        color_samples.append(sample)

    return wavelengths,  np.array(color_samples)

def get_d65_illuminant():
    """Get D65 standard illuminant spectral power distribution"""
    # Extended wavelength range to match melanopic sensitivity
    wavelengths = np.arange(380,  781,  1)  # Changed from 5nm to 1nm intervals

    # Interpolate the original D65 data to 1nm intervals
```

```
    original_wavelengths = np.arange(380, 781, 5)
    original_d65_spd = np.array([
        49.975, 52.311, 54.648, 68.701, 82.754, 87.120, 91.486, 92.458,
        93.431, 90.057, 86.682, 95.773, 104.865, 110.936, 117.008,
117.410,
        117.812, 116.336, 114.861, 115.392, 115.923, 112.367, 108.811,
109.082,
        109.354, 108.578, 107.802, 106.296, 104.790, 106.239, 107.689,
106.047,
        104.405, 104.225, 104.046, 102.023, 100.000, 98.167, 96.334,
96.061,
        95.788, 92.237, 88.685, 89.996, 91.306, 90.231, 89.157, 88.475,
        87.793, 85.947, 84.101, 83.233, 82.365, 80.877, 79.388, 78.988,
        78.588, 77.630, 76.672, 76.639, 76.606, 76.851, 77.096, 75.984,
        74.872, 74.349, 73.816, 73.227, 72.638, 71.808, 70.978, 70.458,
        69.938, 69.498, 69.058, 68.269, 67.481, 66.953, 66.425, 65.942,
        65.459, 64.933, 64.408, 64.133
    ])

    # Check array lengths before interpolation
    print(f"Debug: original_wavelengths length: {len(original_wavelengths)}")
    print(f"Debug: original_d65_spd length: {len(original_d65_spd)}")
    print(f"Debug: target wavelengths length: {len(wavelengths)}")

    # Ensure arrays have same length
    min_len = min(len(original_wavelengths), len(original_d65_spd))
    original_wavelengths = original_wavelengths[:min_len]
    original_d65_spd = original_d65_spd[:min_len]

    # Interpolate to 1nm intervals
    f_d65 = interpolate.interp1d(original_wavelengths, original_d65_spd,
                                 kind='linear', bounds_error=False,
fill_value=0)
    d65_spd = f_d65(wavelengths)

    return wavelengths, d65_spd

def get_melanopic_sensitivity_lucas():
    """Get melanopic sensitivity function based on Lucas et al. 2014"""
    # CIE S 026 melanopic sensitivity function
    wavelengths = np.arange(380, 781, 1)

    # Melanopic sensitivity function values (Lucas et al. 2014)
    # Peak at ~490nm, values normalized to peak = 1
```

```
    s_mel = np.zeros_like(wavelengths, dtype=float)

    for i, wl in enumerate(wavelengths):
        if 380 <= wl <= 780:
            # More accurate melanopic sensitivity based on CIE S 026
            if wl < 490:
                # Blue side of the curve
                s_mel[i] = np.exp(-0.5 * ((wl - 490) / 38)**2)
            else:
                # Red side of the curve (steeper falloff)
                s_mel[i] = np.exp(-0.5 * ((wl - 490) / 42)**2)

    # Normalize to peak = 1
    if s_mel.max() > 0:
        s_mel = s_mel / s_mel.max()

    return wavelengths, s_mel

def xyz_to_cam02ucs(X, Y, Z, X_w, Y_w, Z_w, L_A=64):
    """Convert XYZ to CAM02-UCS color space"""
    # Simplified CAM02-UCS conversion
    # In practice, this would be a full CIECAM02 implementation

    # Adaptation
    Y_b = 20.0  # Background luminance factor
    F = 1.0     # Surround condition
    c = 0.69    # Impact of surround
    N_c = 1.0   # Chromatic induction factor

    # Normalize by white point
    X_rel = X / X_w
    Y_rel = Y / Y_w
    Z_rel = Z / Z_w

    # Simplified lightness calculation
    J = 100 * (Y_rel ** 0.5)

    # Simplified chroma calculations
    a = 200 * (X_rel - Y_rel)
    b = 100 * (Y_rel - Z_rel)

    return J, a, b
```

```
def calculate_color_samples_under_illuminant(illuminant_wl, illuminant_spd,
sample_wl, sample_reflectances):
    """Calculate XYZ values for color samples under given illuminant"""
    # Interpolate illuminant to sample wavelengths
    f_illuminant = interpolate.interp1d(illuminant_wl, illuminant_spd,
kind='linear',
                                        bounds_error=False, fill_value=0)
    illuminant_interp = f_illuminant(sample_wl)

    # Get CIE color matching functions
    cie_wl, x_bar, y_bar, z_bar = get_cie_color_matching_functions()

    # Interpolate color matching functions to sample wavelengths
    f_x = interpolate.interp1d(cie_wl, x_bar, kind='linear',
bounds_error=False, fill_value=0)
    f_y = interpolate.interp1d(cie_wl, y_bar, kind='linear',
bounds_error=False, fill_value=0)
    f_z = interpolate.interp1d(cie_wl, z_bar, kind='linear',
bounds_error=False, fill_value=0)

    x_bar_interp = f_x(sample_wl)
    y_bar_interp = f_y(sample_wl)
    z_bar_interp = f_z(sample_wl)

    # Calculate wavelength intervals
    d_lambda = np.diff(sample_wl)
    d_lambda = np.append(d_lambda, d_lambda[-1])

    # Normalization constant K for consistency
    K = 683  # lm/W for photopic vision

    # Calculate XYZ for each color sample under test and reference illuminants
    xyz_values = []
    for reflectance in sample_reflectances:
        # Multiply illuminant * reflectance * color matching function * K
        X = K * np.sum(illuminant_interp * reflectance * x_bar_interp *
d_lambda)
        Y = K * np.sum(illuminant_interp * reflectance * y_bar_interp *
d_lambda)
        Z = K * np.sum(illuminant_interp * reflectance * z_bar_interp *
d_lambda)
        xyz_values.append([X, Y, Z])

    return np.array(xyz_values)
```

```
def interpolate_color_matching_functions(wavelength_data):
    """Interpolate color matching functions to match the data wavelengths"""
    cie_wavelengths, x_bar, y_bar, z_bar =
get_cie_color_matching_functions()

    # Create interpolation functions
    f_x = interpolate.interp1d(cie_wavelengths, x_bar, kind='linear',
                               bounds_error=False, fill_value=0)
    f_y = interpolate.interp1d(cie_wavelengths, y_bar, kind='linear',
                               bounds_error=False, fill_value=0)
    f_z = interpolate.interp1d(cie_wavelengths, z_bar, kind='linear',
                               bounds_error=False, fill_value=0)

    # Interpolate to data wavelengths
    x_bar_interp = f_x(wavelength_data)
    y_bar_interp = f_y(wavelength_data)
    z_bar_interp = f_z(wavelength_data)

    return x_bar_interp, y_bar_interp, z_bar_interp

def calculate_tristimulus_values(wavelength, intensity):
    """Calculate CIE XYZ tristimulus values"""
    # Get interpolated color matching functions
    x_bar, y_bar, z_bar = interpolate_color_matching_functions(wavelength)

    # Calculate wavelength step
    d_lambda = np.diff(wavelength)
    d_lambda = np.append(d_lambda, d_lambda[-1])  # Use last interval for the
last point

    # Normalization constant K for photopic vision
    K = 683  # lm/W for photopic vision

    # Calculate tristimulus values using numerical integration with proper
normalization
    # Note: For relative calculations (CCT, chromaticity), K cancels out
    # But for absolute photometric calculations, K is essential
    X = K * np.sum(intensity * x_bar * d_lambda)
    Y = K * np.sum(intensity * y_bar * d_lambda)  # Y represents luminance when
multiplied by K
    Z = K * np.sum(intensity * z_bar * d_lambda)

    return X, Y, Z
```

```
def calculate_cct_duv(X, Y, Z):
    """Calculate Correlated Color Temperature (CCT) and Distance from
Planckian locus (Duv)"""
    # Convert to chromaticity coordinates
    x = X / (X + Y + Z)
    y = Y / (X + Y + Z)

    print(f"Debug: Chromaticity coordinates x={x:.6f}, y={y:.6f}")

    # McCamy's approximation for CCT
    n = (x - 0.3320) / (0.1858 - y)
    CCT = 449 * n**3 + 3525 * n**2 + 6823.3 * n + 5520.33

    # Calculate Duv using Robertson's method with correct Planckian locus
    # Use CIE 1960 UCS coordinates for more accurate calculation
    u = 4*x / (-2*x + 12*y + 3)
    v = 6*y / (-2*x + 12*y + 3)

    print(f"Debug: UCS coordinates u={u:.6f}, v={v:.6f}")

    # Find closest point on Planckian locus in UCS space
    T_range = np.linspace(max(1000, CCT-500), min(25000, CCT+500), 1000)
    min_dist = float('inf')
    closest_u_p = 0
    closest_v_p = 0

    for T in T_range:
        # Planckian locus in UCS coordinates (more accurate formula)
        u_p = (0.860117757 + 1.54118254e-4*T + 1.28641212e-7*T**2) / (1.0 +
8.42420235e-4*T + 7.08145163e-7*T**2)
        v_p = (0.317398726 + 4.22806245e-5*T + 4.20481691e-8*T**2) / (1.0 -
2.89741816e-5*T + 1.61456053e-7*T**2)

        dist = np.sqrt((u - u_p)**2 + (v - v_p)**2)
        if dist < min_dist:
            min_dist = dist
            closest_u_p = u_p
            closest_v_p = v_p

    # Calculate Duv with proper sign in UCS space
    # Vector from closest point on locus to test point
    du = u - closest_u_p
    dv = v - closest_v_p
```

```
    # Perpendicular direction to the locus (simplified)
    # Positive Duv means above the locus, negative means below
    duv_vector = dv  # Simplified: use v-direction as primary indicator

    # Final Duv calculation
    Duv = np.sqrt(du**2 + dv**2) * np.sign(duv_vector)

    print(f"Debug: min_dist={min_dist:.6f}, final Duv={Duv:.6f}")

    return CCT, Duv

def get_reference_illuminant_spd(cct):
    """Get reference illuminant spectral power distribution based on CCT"""
    wavelengths = np.arange(380, 781, 1)

    if cct < 5000:
        # Use Planckian radiator for CCT < 5000K
        h = 6.626e-34  # Planck constant
        c = 3.0e8      # Speed of light
        k = 1.381e-23  # Boltzmann constant

        spd = []
        for wl in wavelengths:
            wl_m = wl * 1e-9  # Convert nm to m
            # Planck's law
            B = (2 * h * c**2 / wl_m**5) / (np.exp(h * c / (wl_m * k * cct)) -
1)
            spd.append(B)
        spd = np.array(spd)
    else:
        # Use D-illuminant for CCT >= 5000K
        # Simplified D65 approximation scaled for different CCT
        d65_wl, d65_spd = get_d65_illuminant()
        f_d65 = interpolate.interp1d(d65_wl, d65_spd, kind='linear',
                                     bounds_error=False, fill_value=0)
        spd = f_d65(wavelengths)

        # Scale for different CCT (simplified)
        scale_factor = 6500 / cct
        spd = spd * scale_factor

    return wavelengths, spd
```

```
def calculate_rf_accurate(test_wavelength, test_intensity):
    """Calculate accurate Rf based on CAM02-UCS color differences"""
    # Get color samples
    sample_wl, sample_reflectances = get_cri_test_color_samples()

    # Get test source XYZ
    test_X, test_Y, test_Z = calculate_tristimulus_values(test_wavelength,
test_intensity)
    test_cct, _ = calculate_cct_duv(test_X, test_Y, test_Z)

    # Get reference illuminant
    ref_wl, ref_spd = get_reference_illuminant_spd(test_cct)

    # Interpolate test source to sample wavelengths
    f_test = interpolate.interp1d(test_wavelength, test_intensity,
kind='linear',
                                  bounds_error=False, fill_value=0)
    test_spd_interp = f_test(sample_wl)

    # Interpolate reference to sample wavelengths
    f_ref = interpolate.interp1d(ref_wl, ref_spd, kind='linear',
                                 bounds_error=False, fill_value=0)
    ref_spd_interp = f_ref(sample_wl)

    # Calculate XYZ for color samples under test and reference illuminants
    test_xyz = calculate_color_samples_under_illuminant(sample_wl,
test_spd_interp,
                                                        sample_wl,
sample_reflectances)
    ref_xyz = calculate_color_samples_under_illuminant(sample_wl,
ref_spd_interp,
                                                       sample_wl,
sample_reflectances)

    # Calculate white point for both illuminants
    white_sample = np.ones_like(sample_wl)  # Perfect reflector
    test_white_xyz = calculate_color_samples_under_illuminant(sample_wl,
test_spd_interp,
                                                              sample_wl,
[white_sample])[0]
    ref_white_xyz = calculate_color_samples_under_illuminant(sample_wl,
ref_spd_interp,
                                                             sample_wl,
[white_sample])[0]
```

```
    # Convert to CAM02-UCS
    delta_e_values = []

    for i in range(len(sample_reflectances)):
        # Test illuminant CAM02-UCS
        J_test, a_test, b_test = xyz_to_cam02ucs(test_xyz[i, 0],
test_xyz[i, 1], test_xyz[i, 2],
                                                 test_white_xyz[0],
test_white_xyz[1], test_white_xyz[2])

        # Reference illuminant CAM02-UCS
        J_ref, a_ref, b_ref = xyz_to_cam02ucs(ref_xyz[i, 0], ref_xyz[i,
1], ref_xyz[i, 2],
                                              ref_white_xyz[0], ref_white_xyz[1],
ref_white_xyz[2])

        # Calculate color difference in CAM02-UCS
        delta_e = np.sqrt((J_test - J_ref)**2 + (a_test - a_ref)**2 + (b_test -
b_ref)**2)
        delta_e_values.append(delta_e)

    # Calculate average color difference
    delta_e_avg = np.mean(delta_e_values)

    # Calculate Rf
    Rf = 100 - 6.73 * delta_e_avg

    return max(0, min(100, Rf))

def calculate_rg_accurate(test_wavelength, test_intensity):
    """Calculate accurate Rg based on color gamut area"""
    # Get color samples
    sample_wl, sample_reflectances = get_cri_test_color_samples()

    # Get test source XYZ
    test_X, test_Y, test_Z = calculate_tristimulus_values(test_wavelength,
test_intensity)
    test_cct, _ = calculate_cct_duv(test_X, test_Y, test_Z)

    # Get reference illuminant
    ref_wl, ref_spd = get_reference_illuminant_spd(test_cct)

    # Interpolate test source to sample wavelengths
```

```
    f_test = interpolate.interp1d(test_wavelength,  test_intensity,
kind='linear',
                                  bounds_error=False,  fill_value=0)
    test_spd_interp = f_test(sample_wl)

    # Interpolate reference to sample wavelengths
    f_ref = interpolate.interp1d(ref_wl,  ref_spd,  kind='linear',
                                 bounds_error=False,  fill_value=0)
    ref_spd_interp = f_ref(sample_wl)

    # Calculate XYZ for color samples under test and reference illuminants
    test_xyz = calculate_color_samples_under_illuminant(sample_wl,
test_spd_interp,
                                                        sample_wl,
sample_reflectances)
    ref_xyz = calculate_color_samples_under_illuminant(sample_wl,
ref_spd_interp,
                                                       sample_wl,
sample_reflectances)

    # Calculate white point for both illuminants
    white_sample = np.ones_like(sample_wl)  # Perfect reflector
    test_white_xyz = calculate_color_samples_under_illuminant(sample_wl,
test_spd_interp,
                                                              sample_wl,
[white_sample])[0]
    ref_white_xyz = calculate_color_samples_under_illuminant(sample_wl,
ref_spd_interp,
                                                             sample_wl,
[white_sample])[0]

    # Convert to CAM02-UCS and extract a*,  b* coordinates for gamut
calculation
    test_points = []
    ref_points = []

    for i in range(len(sample_reflectances)):
        # Test illuminant CAM02-UCS
        J_test,  a_test,  b_test = xyz_to_cam02ucs(test_xyz[i,  0],
test_xyz[i,  1],  test_xyz[i,  2],
                                                  test_white_xyz[0],
test_white_xyz[1],  test_white_xyz[2])
        test_points.append([a_test,  b_test])
```

```
        # Reference illuminant CAM02-UCS
        J_ref, a_ref, b_ref = xyz_to_cam02ucs(ref_xyz[i, 0], ref_xyz[i,
1], ref_xyz[i, 2],
                                              ref_white_xyz[0], ref_white_xyz[1],
ref_white_xyz[2])
        ref_points.append([a_ref, b_ref])

    test_points = np.array(test_points)
    ref_points = np.array(ref_points)

    # Calculate convex hull areas
    try:
        test_hull = ConvexHull(test_points)
        ref_hull = ConvexHull(ref_points)

        A_test = test_hull.volume  # In 2D, volume is area
        A_ref = ref_hull.volume

        # Calculate Rg
        if A_ref > 0:
            Rg = (A_test / A_ref) * 100
        else:
            Rg = 100

    except Exception:
        # Fallback if ConvexHull fails
        Rg = 100

    return max(50, min(150, Rg))

def calculate_mel-DER_accurate(test_wavelength, test_intensity):
    """Calculate accurate Mel-DER based on melanopic illuminance ratio"""
    # Get melanopic sensitivity function
    mel_wl, s_mel = get_melanopic_sensitivity_lucas()

    # Get D65 illuminant - now both have same wavelength range (380-780nm, 1nm
intervals)
    d65_wl, d65_spd = get_d65_illuminant()

    # Interpolate test source to melanopic wavelengths
    f_test = interpolate.interp1d(test_wavelength, test_intensity,
kind='linear',
                                  bounds_error=False, fill_value=0)
    test_spd_interp = f_test(mel_wl)
```

```
    # No need to interpolate D65 since both mel_wl and d65_wl are identical
    # But keep for safety in case wavelength ranges differ
    if len(d65_wl) == len(mel_wl) and np.allclose(d65_wl, mel_wl):
        d65_spd_interp = d65_spd
    else:
        f_d65 = interpolate.interp1d(d65_wl, d65_spd, kind='linear',
                                     bounds_error=False, fill_value=0)
        d65_spd_interp = f_d65(mel_wl)

    # Calculate wavelength intervals
    d_lambda = np.diff(mel_wl)
    d_lambda = np.append(d_lambda, d_lambda[-1])

    # Calculate melanopic illuminance for test source
    E_mel_test = np.sum(test_spd_interp * s_mel * d_lambda)

    # Calculate melanopic illuminance for D65
    E_mel_d65 = np.sum(d65_spd_interp * s_mel * d_lambda)

    # Calculate Mel-DER
    if E_mel_d65 > 0:
        Mel-DER = E_mel_test / E_mel_d65
    else:
        Mel-DER = 0

    return Mel-DER

def get_cie_1931_spectral_locus():
    """Get CIE 1931 spectral locus boundary for chromaticity diagram"""
    wavelengths = np.arange(380, 781, 5)
    cie_wl, x_bar, y_bar, z_bar = get_cie_color_matching_functions()

    # Calculate chromaticity coordinates for spectral locus
    x_coords = []
    y_coords = []

    for i, wl in enumerate(wavelengths):
        if i < len(x_bar):
            X = x_bar[i]
            Y = y_bar[i]
            Z = z_bar[i]

            if X + Y + Z > 0:
```

```
            x = X / (X + Y + Z)
            y = Y / (X + Y + Z)
            x_coords.append(x)
            y_coords.append(y)

    # Close the spectral locus by connecting to purple line
    x_coords.append(x_coords[0])
    y_coords.append(y_coords[0])

    return np.array(x_coords), np.array(y_coords), wavelengths

def get_planckian_locus():
    """Get Planckian locus (blackbody locus) for chromaticity diagram"""
    # Temperature range from 1000K to 20000K
    temperatures = np.logspace(3, 4.3, 100)  # 1000K to 20000K

    x_coords = []
    y_coords = []

    for T in temperatures:
        # Planck's law for relative spectral power distribution
        wavelengths = np.arange(380, 781, 5)
        h = 6.626e-34  # Planck constant
        c = 3.0e8      # Speed of light
        k = 1.381e-23  # Boltzmann constant

        spd = []
        for wl in wavelengths:
            wl_m = wl * 1e-9  # Convert nm to m
            B = (2 * h * c**2 / wl_m**5) / (np.exp(h * c / (wl_m * k * T)) - 1)
            spd.append(B)

        spd = np.array(spd)

        # Get color matching functions
        cie_wl, x_bar, y_bar, z_bar = get_cie_color_matching_functions()

        # Calculate XYZ
        X = np.sum(spd * x_bar)
        Y = np.sum(spd * y_bar)
        Z = np.sum(spd * z_bar)

        if X + Y + Z > 0:
            x = X / (X + Y + Z)
```

```
            y = Y / (X + Y + Z)
            x_coords.append(x)
            y_coords.append(y)

    return np.array(x_coords), np.array(y_coords), temperatures

def plot_cie_chromaticity_diagram(test_x, test_y, test_cct, duv_value):
    """Plot CIE 1931 chromaticity diagram with test source and Planckian
locus"""

    # Set up the plot with Chinese font support
    plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS', 'DejaVu
Sans']
    plt.rcParams['axes.unicode_minus'] = False

    fig, ax = plt.subplots(figsize=(12, 10))

    # Get spectral locus
    spec_x, spec_y, spec_wavelengths = get_cie_1931_spectral_locus()

    # Get Planckian locus
    planck_x, planck_y, temperatures = get_planckian_locus()

    # Plot spectral locus (y1 equivalent - boundary of visible colors)
    ax.fill(spec_x, spec_y, alpha=0.1, color='lightgray', label='可见光谱轨迹
')
    ax.plot(spec_x[:-1], spec_y[:-1], 'k-', linewidth=2, label='光谱轨迹边界
')

    # Add wavelength labels on spectral locus
    for i in range(0, len(spec_x)-1, 8):  # Every 8th point to avoid crowding
        if i < len(spec_wavelengths):
            ax.annotate(f'{spec_wavelengths[i]:.0f}nm',
                        (spec_x[i], spec_y[i]),
                        xytext=(5, 5), textcoords='offset points',
                        fontsize=8, alpha=0.7)

    # Plot Planckian locus (y1 - blackbody locus)
    ax.plot(planck_x, planck_y, 'b-', linewidth=3, label='普朗克轨迹 (黑体轨
迹)')

    # Add temperature labels on Planckian locus
    temp_indices = [10, 25, 40, 55, 70, 85]  # Selected indices for
labeling
```

```
    for i in temp_indices:
        if i < len(temperatures):
            ax.annotate(f'{temperatures[i]:.0f}K',
                        (planck_x[i], planck_y[i]),
                        xytext=(5, -15), textcoords='offset points',
                        fontsize=9, bbox=dict(boxstyle="round, pad=0.3",
facecolor="white", alpha=0.8))

    # Find closest point on Planckian locus for the test source (y3)
    min_dist = float('inf')
    closest_idx = 0
    for i, (px, py) in enumerate(zip(planck_x, planck_y)):
        dist = np.sqrt((test_x - px)**2 + (test_y - py)**2)
        if dist < min_dist:
            min_dist = dist
            closest_idx = i

    closest_planck_x = planck_x[closest_idx]
    closest_planck_y = planck_y[closest_idx]
    closest_temp = temperatures[closest_idx]

    # Plot closest Planckian point (y3)
    ax.plot(closest_planck_x, closest_planck_y, 'go', markersize=12,
            label=f'最近普朗克点 ({closest_temp:.0f}K)',
markeredgecolor='darkgreen', markeredgewidth=2)

    # Plot test source CCT point (y2)
    ax.plot(test_x, test_y, 'ro', markersize=15,
            label=f'测试光源 (CCT={test_cct:.0f}K)',
markeredgecolor='darkred', markeredgewidth=2)

    # Plot Duv line (y4 - distance from Planckian locus)
    ax.plot([test_x, closest_planck_x], [test_y, closest_planck_y],
            'r--', linewidth=3, alpha=0.8,
            label=f'Duv = {duv_value:.4f}')

    # Add arrow to show Duv direction
    ax.annotate('', xy=(test_x, test_y), xytext=(closest_planck_x,
closest_planck_y),
                arrowprops=dict(arrowstyle='<->', color='red', lw=2))

    # Add text annotation for Duv
    mid_x = (test_x + closest_planck_x) / 2
    mid_y = (test_y + closest_planck_y) / 2
```

```
    ax.annotate(f'Duv={duv_value:.4f}',
                (mid_x, mid_y),
                xytext=(10, 10), textcoords='offset points',
                fontsize=12, fontweight='bold',
                bbox=dict(boxstyle="round, pad=0.5", facecolor="yellow",
alpha=0.8))

    # Set labels and title
    ax.set_xlabel('CIE x', fontsize=14, fontweight='bold')
    ax.set_ylabel('CIE y', fontsize=14, fontweight='bold')
    ax.set_title('CIE 1931 色度图', fontsize=18, fontweight='bold', pad=20)

    # Set equal aspect ratio and limits
    ax.set_aspect('equal')
    ax.set_xlim(0, 0.8)
    ax.set_ylim(0, 0.9)

    # Add grid
    ax.grid(True, alpha=0.3)

    # Add legend
    ax.legend(loc='upper right', fontsize=12, framealpha=0.9)

    # Add coordinate info box
    info_text = f"""光源信息:
CCT: {test_cct:.0f} K
Duv: {duv_value:.4f}
色度坐标: ({test_x:.4f}, {test_y:.4f})
最近黑体温度: {closest_temp:.0f} K"""

    ax.text(0.02, 0.98, info_text, transform=ax.transAxes,
            fontsize=11, verticalalignment='top',
            bbox=dict(boxstyle="round, pad=0.5", facecolor="lightblue",
alpha=0.8))

    plt.tight_layout()

    # Save the plot
    plt.savefig('output/Question1/CIE_chromaticity_diagram.png', dpi=300,
bbox_inches='tight')
    print("CIE 色度图已保存为 'CIE_chromaticity_diagram.png'")

    # Show the plot
    plt.show()
```

```
    return fig, ax

def main():
    """Main function to calculate all parameters"""
    print("Loading data from attachment-Q1.xlsx...")
    wavelength, intensity = load_data()

    print(f"Data loaded: {len(wavelength)} data points")
    print(f"Wavelength range: {wavelength.min():.1f} - {wavelength.max():.1f}
nm")
    print(f"Intensity range: {intensity.min():.3f} - {intensity.max():.3f}
mW/m²")
    print()

    # 1. Calculate color matching functions (x̄, ȳ, z̄)
    x_bar, y_bar, z_bar = interpolate_color_matching_functions(wavelength)
    print("1. Color matching functions calculated")
    print(f"   x̄ range: {x_bar.min():.6f} - {x_bar.max():.6f}")
    print(f"   ȳ range: {y_bar.min():.6f} - {y_bar.max():.6f}")
    print(f"   z̄ range: {z_bar.min():.6f} - {z_bar.max():.6f}")
    print()

    # 2. Calculate CIE XYZ tristimulus values
    X, Y, Z = calculate_tristimulus_values(wavelength, intensity)
    print("2. CIE XYZ tristimulus values:")
    print(f"   X = {X:.3f}")
    print(f"   Y = {Y:.3f}")
    print(f"   Z = {Z:.3f}")
    print()

    # 3. Calculate CCT
    CCT, Duv = calculate_cct_duv(X, Y, Z)
    print("3. Correlated Color Temperature (CCT):")
    print(f"   CCT = {CCT:.0f} K")
    print()

    # 4. Calculate Duv
    print("4. Distance from Planckian locus (Duv):")
    print(f"   Duv = {Duv:.4f}")
    print()

    # 5. Calculate Rf (Color Fidelity Index) - Accurate method
    print("Calculating accurate Rf using CAM02-UCS color differences...")
```

```
    Rf = calculate_rf_accurate(wavelength, intensity)
    print("5. Color Fidelity Index (Rf):")
    print(f"   Rf = {Rf:.1f}")
    print()

    # 6. Calculate Rg (Color Gamut Index) - Accurate method
    print("Calculating accurate Rg using color gamut area...")
    Rg = calculate_rg_accurate(wavelength, intensity)
    print("6. Color Gamut Index (Rg):")
    print(f"   Rg = {Rg:.1f}")
    print()

    # 7. Calculate Mel-DER - Accurate method
    print("Calculating accurate Mel-DER using melanopic sensitivity...")
    Mel-DER = calculate_mel-DER_accurate(wavelength, intensity)
    print("7. Daylight Efficacy Ratio for melanopic response (Mel-DER):")
    print(f"   Mel-DER = {Mel-DER:.4f}")
    print()

    # Summary
    print("="*50)
    print("SUMMARY OF RESULTS")
    print("="*50)
    print(f"CIE X: {X:.3f}")
    print(f"CIE Y: {Y:.3f}")
    print(f"CIE Z: {Z:.3f}")
    print(f"CCT: {CCT:.0f} K")
    print(f"Duv: {Duv:.4f}")
    print(f"Rf: {Rf:.1f}")
    print(f"Rg: {Rg:.1f}")
    print(f"Mel-DER: {Mel-DER:.4f}")

    # Calculate chromaticity coordinates for plotting
    x_chromaticity = X / (X + Y + Z)
    y_chromaticity = Y / (X + Y + Z)

    # Plot chromaticity diagram
    print("\n生成 CIE 色度图...")
    plot_cie_chromaticity_diagram(x_chromaticity, y_chromaticity, CCT, Duv)

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

问题二代码

```python
# -*- coding: utf-8 -*-
"""
2025 华数杯全国大学生数学建模竞赛 C 题
问题 2：多通道 LED 最优权重设计
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
from scipy import optimize,  interpolate
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# 设置中文字体，避免乱码
import platform

def setup_chinese_font():
    """设置中文字体"""
    system = platform.system()

    if system == "Windows":
        font_candidates = [
            'Microsoft YaHei',   # 微软雅黑
            'SimHei',           # 黑体
            'SimSun',           # 宋体
            'KaiTi',            # 楷体
            'FangSong'          # 仿宋
        ]
    elif system == "Darwin":  # macOS
        font_candidates = [
            'PingFang SC',       # 苹方
            'Arial Unicode MS',  # Arial Unicode MS
            'Helvetica',         # Helvetica
            'STHeiti'           # 华文黑体
        ]
    else:  # Linux
        font_candidates = [
            'DejaVu Sans',       # DejaVu Sans
            'WenQuanYi Micro Hei',  # 文泉驿微米黑
            'Noto Sans CJK SC',  # Noto Sans CJK SC
            'SimHei'            # 黑体
        ]
```

```
    # 获取系统可用字体
    available_fonts = [f.name for f in fm.fontManager.ttflist]

    # 找到第一个可用的中文字体
    chinese_font = None
    for font in font_candidates:
        if font in available_fonts:
            chinese_font = font
            break

    if chinese_font:
        print(f"使用字体: {chinese_font}")
        plt.rcParams['font.sans-serif'] = [chinese_font, 'DejaVu Sans']
    else:
        print("未找到合适的中文字体，尝试使用系统默认字体")
        plt.rcParams['font.sans-serif'] = ['sans-serif']

    plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题
    plt.rcParams['figure.dpi'] = 100
    plt.rcParams['savefig.dpi'] = 300

# 初始化字体设置
setup_chinese_font()

def clear_font_cache():
    """清理 matplotlib 字体缓存"""
    try:
        import matplotlib
        matplotlib.font_manager._rebuild()
        print("字体缓存已清理")
    except:
        pass

def ensure_chinese_display():
    """确保中文正常显示"""
    setup_chinese_font()
    plt.clf()
    plt.cla()

# 清理字体缓存(首次运行时)
clear_font_cache()

# 设置绘图风格
```

```
sns.set_style("whitegrid")
try:
    plt.style.use('seaborn-v0_8')
except:
    plt.style.use('seaborn')

print("="*60)
print("2025 华数杯 C 题 - 问题 2：多通道 LED 最优权重设计")
print("="*60)
print()

class LEDLightAnalysis:
    """LED 光源分析主类"""

    def __init__(self):
        """初始化"""
        self.wavelengths = np.arange(380, 781, 1)  # 380-780nm, 1nm 间隔

        # CIE 标准观察者函数和颜色匹配函数
        self.setup_cie_functions()

        # 普朗克轨迹数据
        self.setup_planck_locus()

        # 褪黑激素效应函数
        self.setup_melanopic_function()

    def setup_cie_functions(self):
        """设置 CIE 标准函数"""
        wl = self.wavelengths

        # X 颜色匹配函数(改进的多峰拟合)
        x_main = 1.056 * np.exp(-0.5 * ((wl - 599.8) / 37.9)**2)
        x_secondary = 0.362 * np.exp(-0.5 * ((wl - 442.0) / 16.4)**2)
        x_negative = 0.065 * np.exp(-0.5 * ((wl - 501.1) / 20.9)**2)
        self.x_bar = x_main + x_secondary - x_negative

        # Y 颜色匹配函数(视觉亮度函数，峰值在 555nm)
        self.y_bar = np.exp(-0.5 * ((wl - 556.1) / 46.9)**2)

        # Z 颜色匹配函数(峰值在 449nm)
        self.z_bar = 1.669 * np.exp(-0.5 * ((wl - 449.8) / 19.4)**2)

        # 确保非负并归一化
```

```
        self.x_bar = np.maximum(self.x_bar, 0)
        self.y_bar = np.maximum(self.y_bar, 0)
        self.z_bar = np.maximum(self.z_bar, 0)

        # 标准化 Y 函数到 683 lm/W(光度学常数)
        self.y_bar = self.y_bar / np.max(self.y_bar)

    def setup_planck_locus(self):
        """设置普朗克轨迹"""
        # 色温范围
        self.cct_range = np.arange(1000, 25001, 100)

        # 计算普朗克轨迹上的 xy 色坐标
        self.planck_x = []
        self.planck_y = []

        for cct in self.cct_range:
            spd_planck = self.planck_spd(cct)
            x, y, z = self.calculate_xyz(spd_planck)
            if x + y + z > 0:
                self.planck_x.append(x / (x + y + z))
                self.planck_y.append(y / (x + y + z))
            else:
                self.planck_x.append(0)
                self.planck_y.append(0)

        self.planck_x = np.array(self.planck_x)
        self.planck_y = np.array(self.planck_y)

    def setup_melanopic_function(self):
        """设置褪黑激素效应函数"""
        wl = self.wavelengths

        # 更准确的 melanopic 效应光谱函数
        mel_function = np.zeros_like(wl)

        # 短波段(380-500nm): 主要响应区域
        short_wave_mask = (wl >= 380) & (wl <= 500)
        wl_short = wl[short_wave_mask]
        alpha = 0.5 * ((np.log(wl_short) - np.log(480)) / 0.15)**2
        mel_function[short_wave_mask] = np.exp(-alpha)

        # 长波段(500-780nm): 较弱的响应
        long_wave_mask = (wl > 500) & (wl <= 780)
```

```
        wl_long = wl[long_wave_mask]
        mel_function[long_wave_mask] = 0.1 * np.exp(-(wl_long - 500) / 100)

        # 归一化到峰值为 1
        self.mel_function = mel_function / np.max(mel_function)
        self.mel_function = np.maximum(self.mel_function,  0)

    def planck_spd(self,  temperature):
        """计算普朗克黑体辐射 SPD"""
        h = 6.626e-34  # 普朗克常数
        c = 3e8        # 光速
        k = 1.381e-23  # 玻尔兹曼常数

        wl_m = self.wavelengths * 1e-9  # 转换为米

        # 普朗克公式
        spd = (2 * h * c**2 / wl_m**5) / (np.exp(h * c / (wl_m * k *
temperature)) - 1)

        # 归一化
        spd = spd / np.max(spd) * 100

        return spd

    def calculate_xyz(self,  spd):
        """计算 XYZ 三刺激值"""
        if len(spd) != len(self.wavelengths):
            # 插值到标准波长
            f = interpolate.interp1d(np.linspace(380,  780,  len(spd)),  spd,
                                     bounds_error=False,  fill_value=0)
            spd = f(self.wavelengths)

        # 计算 XYZ
        X = np.trapz(spd * self.x_bar,  self.wavelengths)
        Y = np.trapz(spd * self.y_bar,  self.wavelengths)
        Z = np.trapz(spd * self.z_bar,  self.wavelengths)

        return X,  Y,  Z

    def calculate_cct_duv(self,  spd):
        """计算相关色温 CCT 和 Duv"""
        X,  Y,  Z = self.calculate_xyz(spd)

        if X + Y + Z == 0:
```

```
            return 0,  0

        # 计算 xy 色坐标
        x = X / (X + Y + Z)
        y = Y / (X + Y + Z)

        # 使用 Robertson 方法计算 CCT
        distances = np.sqrt((self.planck_x - x)**2 + (self.planck_y - y)**2)
        min_idx = np.argmin(distances)

        # 在最近点附近进行线性插值以提高精度
        if min_idx > 0 and min_idx < len(self.cct_range) - 1:
            idx_range = [min_idx-1,  min_idx,  min_idx+1]
            x_pts = self.planck_x[idx_range]
            y_pts = self.planck_y[idx_range]
            cct_pts = self.cct_range[idx_range]

            dists = [(x - x_pts[i])**2 + (y - y_pts[i])**2 for i in range(3)]
            weights = [1/max(d,  1e-10) for d in dists]
            total_weight = sum(weights)
            weights = [w/total_weight for w in weights]

            cct = sum(cct_pts[i] * weights[i] for i in range(3))
            closest_x = sum(x_pts[i] * weights[i] for i in range(3))
            closest_y = sum(y_pts[i] * weights[i] for i in range(3))
        else:
            cct = self.cct_range[min_idx]
            closest_x = self.planck_x[min_idx]
            closest_y = self.planck_y[min_idx]

        # Duv 计算(到普朗克轨迹的距离)
        duv = np.sqrt((x - closest_x)**2 + (y - closest_y)**2)

        # 判断 Duv 符号(绿色偏移为正，洋红偏移为负)
        if y > closest_y:
            duv = duv  # 绿色偏移
        else:
            duv = -duv  # 洋红偏移

        return cct,  duv

    def calculate_rf_rg(self,  spd):
        """计算保真度指数 Rf 和色域指数 Rg"""
        cct,  _ = self.calculate_cct_duv(spd)
```

```
        # 选择参考光源
        if cct < 5000:
            ref_spd = self.planck_spd(cct)
        else:
            ref_spd = self.daylight_spd(cct)

        # 定义代表性测试颜色的反射光谱
        test_colors = self.get_test_color_reflectances()

        # 计算测试光源和参考光源下的颜色
        test_colors_under_test = []
        test_colors_under_ref = []

        for reflectance in test_colors:
            # 测试光源下的颜色
            test_spectrum = spd * reflectance
            X_t, Y_t, Z_t = self.calculate_xyz(test_spectrum)
            if X_t + Y_t + Z_t > 0:
                test_colors_under_test.append([X_t/(X_t+Y_t+Z_t),
Y_t/(X_t+Y_t+Z_t)])
            else:
                test_colors_under_test.append([0.33, 0.33])

            # 参考光源下的颜色
            ref_spectrum = ref_spd * reflectance
            X_r, Y_r, Z_r = self.calculate_xyz(ref_spectrum)
            if X_r + Y_r + Z_r > 0:
                test_colors_under_ref.append([X_r/(X_r+Y_r+Z_r),
Y_r/(X_r+Y_r+Z_r)])
            else:
                test_colors_under_ref.append([0.33, 0.33])

        # 计算色差
        color_differences = []
        for i in range(len(test_colors)):
            x_t, y_t = test_colors_under_test[i]
            x_r, y_r = test_colors_under_ref[i]
            delta_e = np.sqrt((x_t - x_r)**2 + (y_t - y_r)**2) * 100
            color_differences.append(delta_e)

        # Rf 计算(保真度指数)
        avg_color_diff = np.mean(color_differences)
        rf = max(0, 100 - avg_color_diff * 4.6)
```

```
        # Rg 计算(色域指数)
        test_area = self.calculate_color_gamut_area(test_colors_under_test)
        ref_area = self.calculate_color_gamut_area(test_colors_under_ref)

        if ref_area > 0:
            rg = (test_area / ref_area) * 100
        else:
            rg = 100

        rg = max(50,  min(150,  rg))

        return rf,  rg

    def get_test_color_reflectances(self):
        """获取测试颜色的反射光谱"""
        colors = []
        wl = self.wavelengths

        # 红色(峰值在 700nm)
        red = 0.1 + 0.8 * np.exp(-((wl - 700) / 50)**2)
        colors.append(red)

        # 橙色(峰值在 600nm)
        orange = 0.1 + 0.7 * np.exp(-((wl - 600) / 40)**2)
        colors.append(orange)

        # 黄色(峰值在 580nm)
        yellow = 0.2 + 0.6 * np.exp(-((wl - 580) / 35)**2)
        colors.append(yellow)

        # 绿色(峰值在 530nm)
        green = 0.1 + 0.7 * np.exp(-((wl - 530) / 30)**2)
        colors.append(green)

        # 青色(峰值在 480nm)
        cyan = 0.1 + 0.6 * np.exp(-((wl - 480) / 25)**2)
        colors.append(cyan)

        # 蓝色(峰值在 450nm)
        blue = 0.1 + 0.8 * np.exp(-((wl - 450) / 30)**2)
        colors.append(blue)

        # 紫色(峰值在 420nm)
```

```
        purple = 0.1 + 0.6 * np.exp(-((wl - 420) / 25)**2)
        colors.append(purple)

        # 皮肤色(宽峰)
        skin = 0.3 + 0.4 * np.exp(-((wl - 600) / 80)**2)
        colors.append(skin)

        return colors

    def calculate_color_gamut_area(self, color_points):
        """计算色域面积"""
        if len(color_points) < 3:
            return 0

        from scipy.spatial import ConvexHull
        try:
            hull = ConvexHull(color_points)
            return hull.volume  # 2D 情况下 volume 就是面积
        except:
            return 0

    def daylight_spd(self, cct):
        """标准日光 D 系列 SPD"""
        wl = self.wavelengths

        if cct <= 4000:
            spd = 100 * (1 + 0.3 * np.exp(-((wl - 650) / 100)**2))
        elif cct >= 8000:
            spd = 100 * (1 + 0.5 * np.exp(-((wl - 450) / 80)**2))
        else:
            spd = 100 * np.ones_like(wl)

        return spd

    def calculate_mel_der(self, spd):
        """计算褪黑激素日光效率比 mel-DER"""
        mel_irradiance = np.trapz(spd * self.mel_function, self.wavelengths)
        photopic_irradiance = np.trapz(spd * self.y_bar, self.wavelengths)

        if photopic_irradiance == 0:
            return 0

        mel_der = mel_irradiance / photopic_irradiance
```

```
        return mel_der

# 创建分析实例
led_analyzer = LEDLightAnalysis()

def Question2_led_optimization():
    """问题 2：多通道 LED 最优权重设计"""
    print("\n" + "="*50)
    print("问题 2：多通道 LED 最优权重设计")
    print("="*50)

    try:
        # 读取五通道 LED 的 SPD 数据
        print("正在读取 Problem 2 LED SPD 数据...")

        excel_file = pd.ExcelFile('D:\\华数杯 C-Code(1-3)\\华数杯 C-Code(1-3)\\附
录.xlsx')

        print(f"Excel 文件包含的工作表: {excel_file.sheet_names}")

        # 查找 Problem 2 相关的工作表
        Question2_sheet = None
        for sheet in excel_file.sheet_names:
            if 'Problem' in sheet and '2' in sheet:
                Question2_sheet = sheet
                break

        if Question2_sheet is None:
            # 如果没找到，尝试第二个工作表
            if len(excel_file.sheet_names) > 1:
                Question2_sheet = excel_file.sheet_names[1]
            else:
                Question2_sheet = excel_file.sheet_names[0]

        print(f"使用工作表: {Question2_sheet}")

        # 读取数据
        led_data = pd.read_excel('D:\\华数杯 C-Code(1-3)\\华数杯 C-Code(1-
3)\\attachment-Q2.xlsx')
        print(f"数据形状: {led_data.shape}")
        print("数据前 5 行:")
        print(led_data.head())
        print(f"列名: {list(led_data.columns)}")
```

```
        # 假设第一列是波长，后面 5 列是 5 个通道的 SPD
        if len(led_data.columns) >= 6:
            wavelength_col = led_data.columns[0]
            led_channels = led_data.columns[1:6]  # 取前 5 个 LED 通道

            # 处理波长列
            wavelengths_raw = led_data[wavelength_col].values
            led_spds_raw = led_data[led_channels].values

            # 提取数值部分
            wavelengths = []
            led_spds_clean = []

            for i, wl in enumerate(wavelengths_raw):
                try:
                    # 处理波长
                    if isinstance(wl, str):
                        wl_num = float(wl.split('(')[0]) if '(' in wl else
float(wl)
                    else:
                        wl_num = float(wl)

                    # 处理 LED 数据
                    led_row = []
                    valid_row = True
                    for j in range(min(5, len(led_channels))):
                        try:
                            val = float(led_spds_raw[i, j])
                            if np.isnan(val):
                                valid_row = False
                                break
                            led_row.append(val)
                        except:
                            valid_row = False
                            break

                    if valid_row and not np.isnan(wl_num) and len(led_row) == 5:
                        wavelengths.append(wl_num)
                        led_spds_clean.append(led_row)
                except:
                    continue

            wavelengths = np.array(wavelengths)
            led_spds = np.array(led_spds_clean)
```

```
            print(f"有效数据点数: {len(wavelengths)}")
            print(f"波长范围: {wavelengths.min():.1f} - {wavelengths.max():.1f}
nm")
            print(f"LED 通道名称: {list(led_channels)}")

            # 插值到标准波长网格
            led_spds_standard = np.zeros((len(led_analyzer.wavelengths), 5))
            for i in range(5):
                f = interpolate.interp1d(wavelengths, led_spds[:, i],
                                         bounds_error=False, fill_value=0)
                led_spds_standard[:, i] = f(led_analyzer.wavelengths)

            # 优化权重
            optimize_led_weights(led_spds_standard, led_channels)

        else:
            print("数据格式不正确，需要至少 6 列数据(波长+5 个 LED 通道)")
            raise ValueError("数据格式错误")

    except Exception as e:
        print(f"读取数据时出错: {e}")
        print("使用模拟数据进行演示...")

        # 生成五通道 LED 的模拟 SPD 数据
        led_spds_standard = np.zeros((len(led_analyzer.wavelengths), 5))

        # 通道 1: 深红光 (Deep Red) - 峰值在 660nm
        deep_red = 100 * np.exp(-0.5 * ((led_analyzer.wavelengths - 660) /
20)**2)
        led_spds_standard[:, 0] = deep_red

        # 通道 2: 绿光 (Green) - 峰值在 530nm
        green = 100 * np.exp(-0.5 * ((led_analyzer.wavelengths - 530) / 25)**2)
        led_spds_standard[:, 1] = green

        # 通道 3: 蓝光 (Blue) - 峰值在 450nm
        blue = 100 * np.exp(-0.5 * ((led_analyzer.wavelengths - 450) / 20)**2)
        led_spds_standard[:, 2] = blue

        # 通道 4: 暖白光 WW (~3000K)
        ww_spd = led_analyzer.planck_spd(3000)
        led_spds_standard[:, 3] = ww_spd
```

```
        # 通道 5: 冷白光 CW (~6500K)
        cw_spd = led_analyzer.planck_spd(6500)
        led_spds_standard[:, 4] = cw_spd

        led_channels = ['深红光', '绿光', '蓝光', '暖白光 WW', '冷白光 CW']

        # 优化权重
        optimize_led_weights(led_spds_standard, led_channels)

def optimize_led_weights(led_spds, channel_names):
    """优化 LED 权重"""

    def objective_daylight(weights):
        """日间照明目标函数"""
        if np.sum(weights) == 0:
            return 1e10
        weights = weights / np.sum(weights)  # 归一化

        # 合成 SPD
        combined_spd = np.dot(led_spds, weights)

        # 计算参数
        cct, duv = led_analyzer.calculate_cct_duv(combined_spd)
        rf, rg = led_analyzer.calculate_rf_rg(combined_spd)

        # 目标: CCT 6000±500K, Rf 最大化(>88), Rg 95-105
        cct_penalty = 0
        if not (5500 <= cct <= 6500):
            cct_penalty = abs(cct - 6000) / 100

        rf_reward = rf if rf > 88 else rf - 100

        rg_penalty = 0
        if not (95 <= rg <= 105):
            if rg < 95:
                rg_penalty = (95 - rg) / 10
            else:
                rg_penalty = (rg - 105) / 10

        # 目标函数: 最大化 Rf, 最小化 CCT 和 Rg 偏差
        return -(rf_reward - cct_penalty * 10 - rg_penalty * 5)

    def objective_night(weights):
        """夜间助眠目标函数"""
```

```
        if np.sum(weights) == 0:
            return 1e10
        weights = weights / np.sum(weights)

        combined_spd = np.dot(led_spds, weights)

        cct, duv = led_analyzer.calculate_cct_duv(combined_spd)
        rf, rg = led_analyzer.calculate_rf_rg(combined_spd)
        mel_der = led_analyzer.calculate_mel_der(combined_spd)

        # 目标：CCT 3000±500K, Rf≥80, mel-DER 最小化
        cct_penalty = 0
        if not (2500 <= cct <= 3500):
            cct_penalty = abs(cct - 3000) / 100

        rf_penalty = 0
        if rf < 80:
            rf_penalty = (80 - rf) / 10

        # 目标函数：最小化 mel-DER，满足 CCT 和 Rf 约束
        return mel_der + cct_penalty * 5 + rf_penalty * 3

    # 约束条件
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},  # 权重和为 1
    ]
    bounds = [(0, 1) for _ in range(5)]  # 权重非负且小于等于 1

    # 初始猜测
    x0 = np.ones(5) / 5

    # 日间照明优化
    print("\n 优化日间照明模式...")
    result_day = optimize.minimize(objective_daylight, x0, method='SLSQP',
                                   bounds=bounds, constraints=constraints)

    if result_day.success:
        weights_day = result_day.x
        weights_day = weights_day / np.sum(weights_day)  # 归一化
        combined_spd_day = np.dot(led_spds, weights_day)

        cct_day, duv_day = led_analyzer.calculate_cct_duv(combined_spd_day)
        rf_day, rg_day = led_analyzer.calculate_rf_rg(combined_spd_day)
        mel_der_day = led_analyzer.calculate_mel_der(combined_spd_day)
```

```
        print("日间照明最优权重:")
        for i, (name, weight) in enumerate(zip(channel_names, weights_day)):
            print(f"  {name}: {weight:.3f}")
        print(f"性能指标: CCT={cct_day:.0f}K, Rf={rf_day:.1f},
Rg={rg_day:.1f}, mel-DER={mel_der_day:.3f}")
    else:
        print("日间照明优化失败")
        weights_day = np.ones(5) / 5
        combined_spd_day = np.dot(led_spds, weights_day)
        cct_day, duv_day = led_analyzer.calculate_cct_duv(combined_spd_day)
        rf_day, rg_day = led_analyzer.calculate_rf_rg(combined_spd_day)
        mel_der_day = led_analyzer.calculate_mel_der(combined_spd_day)

    # 夜间助眠优化
    print("\n优化夜间助眠模式...")
    result_night = optimize.minimize(objective_night, x0, method='SLSQP',
                                     bounds=bounds, constraints=constraints)

    if result_night.success:
        weights_night = result_night.x
        weights_night = weights_night / np.sum(weights_night)
        combined_spd_night = np.dot(led_spds, weights_night)

        cct_night, duv_night =
led_analyzer.calculate_cct_duv(combined_spd_night)
        rf_night, rg_night = led_analyzer.calculate_rf_rg(combined_spd_night)
        mel_der_night = led_analyzer.calculate_mel_der(combined_spd_night)

        print("夜间助眠最优权重:")
        for i, (name, weight) in enumerate(zip(channel_names,
weights_night)):
            print(f"  {name}: {weight:.3f}")
        print(f"性能指标: CCT={cct_night:.0f}K, Rf={rf_night:.1f},
Rg={rg_night:.1f}, mel-DER={mel_der_night:.3f}")
    else:
        print("夜间助眠优化失败")
        weights_night = np.array([0.3, 0.1, 0.0, 0.6, 0.0]) # 偏向暖色
        combined_spd_night = np.dot(led_spds, weights_night)
        cct_night, duv_night =
led_analyzer.calculate_cct_duv(combined_spd_night)
        rf_night, rg_night = led_analyzer.calculate_rf_rg(combined_spd_night)
        mel_der_night = led_analyzer.calculate_mel_der(combined_spd_night)
```

```
    # 创建图表 - 四个独立的图
    create_Question2_figures(led_spds, channel_names, weights_day,
weights_night,
                         combined_spd_day, combined_spd_night,
                         cct_day, rf_day, rg_day, mel_der_day,
                         cct_night, rf_night, rg_night, mel_der_night)

    # 输出答案
    print("\n" + "="*30)
    print("问题 2 答案:")
    print("="*30)
    print("日间照明模式最优权重:")
    for i, (name, weight) in enumerate(zip(channel_names, weights_day)):
        print(f"  {name}: {weight:.3f}")
    print(f"  性能: CCT={cct_day:.0f}K, Rf={rf_day:.1f}, Rg={rg_day:.1f},
mel-DER={mel_der_day:.3f}")

    print("\n夜间助眠模式最优权重:")
    for i, (name, weight) in enumerate(zip(channel_names, weights_night)):
        print(f"  {name}: {weight:.3f}")
    print(f"  性能: CCT={cct_night:.0f}K, Rf={rf_night:.1f},
Rg={rg_night:.1f}, mel-DER={mel_der_night:.3f}")
    print("="*30)

def create_Question2_figures(led_spds, channel_names, weights_day,
weights_night,
                         spd_day, spd_night, cct_day, rf_day, rg_day,
mel_der_day,
                         cct_night, rf_night, rg_night, mel_der_night):
    """创建问题 2 的六个独立图表"""

    # 确保中文字体正常显示
    ensure_chinese_display()

    # 创建输出文件夹
    import os
    if not os.path.exists('output/Question2'):
        os.makedirs('output/Question2')

    # 计算 Duv 值
    _, duv_day = led_analyzer.calculate_cct_duv(spd_day)
    _, duv_night = led_analyzer.calculate_cct_duv(spd_night)

    # 图 1: 日间、夜间参数柱状图
```

```
    plt.figure(figsize=(14, 8))

    # 参数名称和值
    params = ['CCT', 'Rf', 'Rg', 'Duv×100', 'mel-DER×1000']
    day_values = [cct_day, rf_day, rg_day, duv_day*100, mel_der_day*1000]
    night_values = [cct_night, rf_night, rg_night, duv_night*100,
mel_der_night*1000]

    x = np.arange(len(params))
    width = 0.35

    bars1 = plt.bar(x - width/2, day_values, width, label='日间照明模式',
                    alpha=0.8, edgecolor='black', linewidth=1)
    bars2 = plt.bar(x + width/2, night_values, width, label='夜间助眠模式',
                    alpha=0.8, edgecolor='black', linewidth=1)

    plt.xlabel('性能参数', fontsize=14)
    plt.ylabel('参数值', fontsize=14)
    plt.title('日间、夜间照明模式性能参数对比', fontsize=16, fontweight='bold')
    plt.xticks(x, params, fontsize=12)
    plt.legend(fontsize=12)
    plt.grid(True, alpha=0.3, axis='y')

    # 添加数值标注
    for i, (bar1, bar2, d_val, n_val) in enumerate(zip(bars1, bars2,
day_values, night_values)):
        # 日间模式标注
        plt.text(bar1.get_x() + bar1.get_width()/2., bar1.get_height() +
max(max(day_values), max(night_values))*0.01,
                 f'{d_val:.1f}' if i < 3 else f'{d_val:.3f}',
                 ha='center', va='bottom', fontsize=10, fontweight='bold')

        # 夜间模式标注
        plt.text(bar2.get_x() + bar2.get_width()/2., bar2.get_height() +
max(max(day_values), max(night_values))*0.01,
                 f'{n_val:.1f}' if i < 3 else f'{n_val:.3f}',
                 ha='center', va='bottom', fontsize=10, fontweight='bold')

    plt.tight_layout()
    plt.savefig('output/Question2/图 1_日间夜间参数对比.png', dpi=300,
bbox_inches='tight')
    plt.show()

    # 图 2: 五通道 LED 光谱图
```

```
    plt.figure(figsize=(14, 8))
    colors = ['Blue', 'Green', 'Red', 'Black', 'Cyan']
    linestyles = ['-', '-', '-', '--', '--']

    for i, (spd, name, color, ls) in enumerate(zip(led_spds.T,
channel_names, colors, linestyles)):
        plt.plot(led_analyzer.wavelengths, spd, color=color, linewidth=3,
                 label=f'通道{i+1}: {name}', alpha=0.9, linestyle=ls)

    plt.xlabel('波长 (nm)', fontsize=14)
    plt.ylabel('相对功率', fontsize=14)
    plt.title('五通道 LED 光谱功率分布', fontsize=16, fontweight='bold')
    plt.legend(fontsize=12, loc='upper right')
    plt.grid(True, alpha=0.3)
    plt.xlim(380, 780)

    # 添加波长区域标注
    plt.axvspan(380, 450, alpha=0.1, color='blue', label='蓝光区域')
    plt.axvspan(450, 550, alpha=0.1, color='green', label='绿光区域')
    plt.axvspan(550, 650, alpha=0.1, color='yellow', label='黄光区域')
    plt.axvspan(650, 780, alpha=0.1, color='red', label='红光区域')

    plt.tight_layout()
    plt.savefig('output/Question2/图 2_五通道 LED 光谱分布.png', dpi=300,
bbox_inches='tight')
    plt.show()

    # 图 3: 最优权重参数图(柱状图)
    plt.figure(figsize=(14, 8))
    x = np.arange(len(channel_names))
    width = 0.35

    bars1 = plt.bar(x - width/2, weights_day, width, label='日间照明模式',
                    alpha=0.8, edgecolor='black', linewidth=1)
    bars2 = plt.bar(x + width/2, weights_night, width, label='夜间助眠模式',
                    alpha=0.8, edgecolor='black', linewidth=1)

    plt.xlabel('LED 通道', fontsize=14)
    plt.ylabel('权重比例', fontsize=14)
    plt.title('最优权重配比参数', fontsize=16, fontweight='bold')
    plt.xticks(x, [f'{name}\n(通道{i+1})' for i, name in
enumerate(channel_names)],
               fontsize=11, ha='center')
    plt.legend(fontsize=12)
```

```
    plt.grid(True, alpha=0.3, axis='y')

    # 添加权重数值标注
    for bar, weight in zip(bars1, weights_day):
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height + 0.01,
                f'{weight:.3f}\n({weight*100:.1f}%)',
                ha='center', va='bottom', fontsize=10, fontweight='bold')

    for bar, weight in zip(bars2, weights_night):
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height + 0.01,
                f'{weight:.3f}\n({weight*100:.1f}%)',
                ha='center', va='bottom', fontsize=10, fontweight='bold')

    plt.ylim(0, max(max(weights_day), max(weights_night)) * 1.2)
    plt.tight_layout()
    plt.savefig('output/Question2/图 3_最优权重配比数.png', dpi=300,
bbox_inches='tight')
    plt.show()

    # 图 4: 五通道 LED 光谱图 (与图 2 相同但强调合成效果)
    plt.figure(figsize=(14, 8))

    # 绘制单独通道光谱(较浅颜色)
    for i, (spd, name, color) in enumerate(zip(led_spds.T, channel_names,
colors)):
        plt.plot(led_analyzer.wavelengths, spd, color=color, linewidth=2,
                alpha=0.5, linestyle='--', label=f'{name}(单通道)')

    # 绘制加权合成光谱
    plt.plot(led_analyzer.wavelengths, spd_day, linewidth=4,
            alpha=0.9, label='日间模式合成光谱')
    plt.plot(led_analyzer.wavelengths, spd_night, linewidth=4,
            alpha=0.9, label='夜间模式合成光谱')

    plt.xlabel('波长 (nm)', fontsize=14)
    plt.ylabel('相对功率', fontsize=14)
    plt.title('五通道 LED 光谱及合成光效果', fontsize=16, fontweight='bold')
    plt.legend(fontsize=10, loc='upper right', ncol=2)
    plt.grid(True, alpha=0.3)
    plt.xlim(380, 780)

    # 添加性能指标文本框
```

```
    textstr_day = f'日间模式:\nCCT: {cct_day:.0f}K\nRf: {rf_day:.1f}\nRg:
{rg_day:.1f}'
    textstr_night = f'夜间模式:\nCCT: {cct_night:.0f}K\nRf: {rf_night:.1f}\nRg:
{rg_night:.1f}'

    plt.text(0.02, 0.98, textstr_day, transform=plt.gca().transAxes,
fontsize=11,
             verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='gold', alpha=0.8))
    plt.text(0.02, 0.78, textstr_night, transform=plt.gca().transAxes,
fontsize=11,
             verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='lightblue', alpha=0.8))

    plt.tight_layout()
    plt.savefig('output/Question2/图 4_LED 光谱合成效果.png', dpi=300,
bbox_inches='tight')
    plt.show()

    # 图 5: 综合性能雷达图
    from math import pi

    plt.figure(figsize=(12, 10))

    # 设置雷达图参数
    categories = ['CCT 适宜性', '保真度 Rf', '色域 Rg', 'Duv 性能', '节律调节']
    N = len(categories)

    # 标准化函数
    def normalize_cct(cct, is_day=True):
        target = 6000 if is_day else 3000
        return max(0, 1 - abs(cct - target) / 2000)

    def normalize_duv(duv):
        return max(0, 1 - abs(duv) / 0.01)  # Duv 理想值接近 0

    def normalize_mel_der(mel_der, is_day=True):
        if is_day:
            return min(1, mel_der / 0.5)  # 日间可以有一定 mel-DER
        else:
            return max(0, 1 - mel_der / 0.3)  # 夜间 mel-DER 越小越好

    # 日间模式标准化数据
    day_values = [
```

```
    normalize_cct(cct_day, True),
    rf_day / 100,
    min(rg_day / 100, 1.2),
    normalize_duv(duv_day),
    normalize_mel_der(mel_der_day, True)
]

# 夜间模式标准化数据
night_values = [
    normalize_cct(cct_night, False),
    rf_night / 100,
    min(rg_night / 100, 1.2),
    normalize_duv(duv_night),
    normalize_mel_der(mel_der_night, False)
]

# 计算角度
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]  # 闭合

day_values += day_values[:1]
night_values += night_values[:1]

# 创建雷达图
ax = plt.subplot(111, projection='polar')

# 绘制数据
ax.plot(angles, day_values, 'o-', linewidth=3, label='日间照明模式',
        markersize=8)
ax.fill(angles, day_values, alpha=0.25, )

ax.plot(angles, night_values, 'o-', linewidth=3, label='夜间助眠模式',
         markersize=8)
ax.fill(angles, night_values, alpha=0.25, )

# 设置标签
ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories, fontsize=12)
ax.set_ylim(0, 1)
ax.set_yticks([0.2, 0.4, 0.6, 0.8, 1.0])
ax.set_yticklabels(['0.2', '0.4', '0.6', '0.8', '1.0'], fontsize=10)
ax.grid(True)

plt.title('综合性能雷达图', fontsize=16, fontweight='bold', pad=20)
```

```
    plt.legend(loc='upper right', bbox_to_anchor=(1.3, 1.0), fontsize=12)

    # 添加性能说明
    performance_text = """
    性能评价标准:
    • CCT 适宜性: 日间 6000K, 夜间 3000K 为理想值
    • 保真度 Rf: 越接近 100 越好
    • 色域 Rg: 接近 100 为理想
    • Duv 性能: 越接近 0 越好
    • 节律调节: 日间适中, 夜间越低越好
    """

    plt.figtext(0.02, 0.02, performance_text, fontsize=10,
                bbox=dict(boxstyle='round', facecolor='lightgray', alpha=0.8))

    plt.tight_layout()
    plt.savefig('output/Question2/图 5_综合性能雷达图.png', dpi=300,
bbox_inches='tight')
    plt.show()

    # 图 6: 标准色盘上显示日间/夜间最优光照的位置
    plt.figure(figsize=(12, 10))

    # 绘制 CIE 1931 色度图轮廓
    # 创建光谱轨迹(单色光轨迹)
    wavelengths_mono = np.arange(380, 700, 5)
    spectrum_x = []
    spectrum_y = []

    for wl in wavelengths_mono:
        # 创建单色光谱(在特定波长处有峰值)
        mono_spd = np.exp(-0.5 * ((led_analyzer.wavelengths - wl) / 5)**2)
        X, Y, Z = led_analyzer.calculate_xyz(mono_spd)
        if X + Y + Z > 0:
            spectrum_x.append(X / (X + Y + Z))
            spectrum_y.append(Y / (X + Y + Z))

    # 闭合光谱轨迹(连接紫端)
    spectrum_x.append(spectrum_x[0])
    spectrum_y.append(spectrum_y[0])

    # 绘制光谱轨迹
    plt.plot(spectrum_x, spectrum_y, 'k-', linewidth=2, label='光谱轨迹',
alpha=0.7)
```

```
    plt.fill(spectrum_x, spectrum_y, color='lightgray', alpha=0.3, label='可
见光色域')

    # 绘制普朗克轨迹(黑体轨迹)
    plt.plot(led_analyzer.planck_x, led_analyzer.planck_y, 'k--',
             linewidth=2, label='普朗克轨迹', alpha=0.8)

    # 计算日间和夜间光照的 xy 色坐标
    X_day, Y_day, Z_day = led_analyzer.calculate_xyz(spd_day)
    X_night, Y_night, Z_night = led_analyzer.calculate_xyz(spd_night)

    if X_day + Y_day + Z_day > 0:
        x_day = X_day / (X_day + Y_day + Z_day)
        y_day = Y_day / (X_day + Y_day + Z_day)
    else:
        x_day, y_day = 0.33, 0.33

    if X_night + Y_night + Z_night > 0:
        x_night = X_night / (X_night + Y_night + Z_night)
        y_night = Y_night / (X_night + Y_night + Z_night)
    else:
        x_night, y_night = 0.33, 0.33

    # 绘制日间和夜间光照位置
    plt.scatter(x_day, y_day, s=300, c='gold', marker='*',
                edgecolor='black', linewidth=2, label=f'日间照明
\n(CCT={cct_day:.0f}K)', zorder=10)
    plt.scatter(x_night, y_night, s=300, c='navy', marker='*',
                edgecolor='black', linewidth=2, label=f'夜间助眠
\n(CCT={cct_night:.0f}K)', zorder=10)

    # 标注 Duv 线(到普朗克轨迹的距离)
    # 找到普朗克轨迹上最近的点
    distances_day = np.sqrt((led_analyzer.planck_x - x_day)**2 +
(led_analyzer.planck_y - y_day)**2)
    min_idx_day = np.argmin(distances_day)

    distances_night = np.sqrt((led_analyzer.planck_x - x_night)**2 +
(led_analyzer.planck_y - y_night)**2)
    min_idx_night = np.argmin(distances_night)

    # 绘制 Duv 线
    plt.plot([x_day, led_analyzer.planck_x[min_idx_day]],
             [y_day, led_analyzer.planck_y[min_idx_day]],
```

```
            'gold', linewidth=3, alpha=0.7, linestyle=':',
            label=f'日间 Duv={duv_day:.4f}')

    plt.plot([x_night, led_analyzer.planck_x[min_idx_night]],
            [y_night, led_analyzer.planck_y[min_idx_night]],
            'navy', linewidth=3, alpha=0.7, linestyle=':',
            label=f'夜间 Duv={duv_night:.4f}')

    # 标注一些关键色温点
    key_ccts = [2700, 3000, 4000, 5000, 6500, 10000]
    for cct in key_ccts:
        # 找到对应的 xy 坐标
        distances_cct = np.abs(led_analyzer.cct_range - cct)
        idx_cct = np.argmin(distances_cct)
        if idx_cct < len(led_analyzer.planck_x):
            plt.annotate(f'{cct}K',
                         (led_analyzer.planck_x[idx_cct],
led_analyzer.planck_y[idx_cct]),
                         xytext=(5, 5), textcoords='offset points',
                         fontsize=9, alpha=0.7)

    # 添加等色温线(示意)
    for cct in [3000, 6000]:
        distances_cct = np.abs(led_analyzer.cct_range - cct)
        idx_cct = np.argmin(distances_cct)
        if idx_cct < len(led_analyzer.planck_x):
            # 绘制等色温线(垂直于普朗克轨迹)
            center_x = led_analyzer.planck_x[idx_cct]
            center_y = led_analyzer.planck_y[idx_cct]

            # 简化的等色温线
            line_length = 0.05
            if idx_cct > 0 and idx_cct < len(led_analyzer.planck_x) - 1:
                # 计算切线方向
                dx = led_analyzer.planck_x[idx_cct+1] -
led_analyzer.planck_x[idx_cct-1]
                dy = led_analyzer.planck_y[idx_cct+1] -
led_analyzer.planck_y[idx_cct-1]
                norm = np.sqrt(dx**2 + dy**2)
                if norm > 0:
                    # 垂直方向
                    perp_x = -dy / norm * line_length
                    perp_y = dx / norm * line_length
```

```
                    plt.plot([center_x - perp_x, center_x + perp_x],
                             [center_y - perp_y, center_y + perp_y],
                             'gray', linewidth=1, alpha=0.5)

    # 设置图表属性
    plt.xlabel('x 色度坐标', fontsize=14)
    plt.ylabel('y 色度坐标', fontsize=14)
    plt.title('CIE 1931 色度图 - 日间/夜间最优光照位置', fontsize=16,
fontweight='bold')
    plt.legend(fontsize=10, loc='upper right')
    plt.grid(True, alpha=0.3)

    # 设置坐标轴范围
    plt.xlim(0, 0.8)
    plt.ylim(0, 0.9)

    # 添加性能信息文本框
    info_text = f"""优化结果对比:

日间照明模式:
• 色坐标: ({x_day:.3f}, {y_day:.3f})
• CCT: {cct_day:.0f}K
• Duv: {duv_day:.4f}
• Rf: {rf_day:.1f}
• mel-DER: {mel_der_day:.3f}

夜间助眠模式:
• 色坐标: ({x_night:.3f}, {y_night:.3f})
• CCT: {cct_night:.0f}K
• Duv: {duv_night:.4f}
• Rf: {rf_night:.1f}
• mel-DER: {mel_der_night:.3f}"""

    plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
fontsize=10,
             verticalalignment='top', horizontalalignment='left',
             bbox=dict(boxstyle='round, pad=0.5', facecolor='white',
alpha=0.9, edgecolor='gray'))

    # 添加色温说明
    temp_text = """色温说明:
• 2700K: 温馨暖光
• 3000K: 舒适暖光
• 4000K: 自然白光
```

```
• 5000K: 日光白
• 6500K: 冷白光"""

    plt.text(0.98, 0.02, temp_text, transform=plt.gca().transAxes,
fontsize=9,
            verticalalignment='bottom', horizontalalignment='right',
            bbox=dict(boxstyle='round,pad=0.3', facecolor='lightblue',
alpha=0.8))

    plt.tight_layout()
    plt.savefig('output/Question2/图 6_CIE 色度图最优光照位置.png', dpi=300,
bbox_inches='tight')
    plt.show()

    print("问题 2 的六个图表已保存到 output/Question2/ 文件夹:")
    print("  - 图 1: 日间、夜间参数对比柱状图")
    print("  - 图 2: 五通道 LED 光谱分布")
    print("  - 图 3: 最优权重配比参数")
    print("  - 图 4: LED 光谱合成效果")
    print("  - 图 5: 综合性能雷达图")
    print("  - 图 6: CIE 色度图最优光照位置")

if __name__ == "__main__":
    print("LED 光源分析系统初始化完成！")
    print("已设置 CIE 标准函数、普朗克轨迹和褪黑激素效应函数")
    print()

    # 运行问题 2
    Question2_led_optimization()
```