重庆大学

数学建模校内竞赛论文

论文题目：

组号：

成员：

选题：

姓名	学院	年级	专业	学号	联系电话	数学分析	高等代数	微积分	高等数学	线性代数	概率统计	数学实验	数学模型	CET4	CET6

日期

摘 要

待补全

关键词：少样本分类；关系建模；对比学习；语义信息表示

Key words:

目 录

图目录

表目录

1 绪 论

dawd[1]

1.1 研究背景与意义

随着当下显示器技术的发展，超高清技术、HDR 技术的出现，显示器设备对色彩表现力的要求越来越高。然而，由于图像采集设备与图像现实设备二者对色彩的感知和还原能力存在差异，导致视频源中的色彩信息往往无法在显示设备上完美复现。在当前超高清显示器的需求日益增长的背景下，如何在有限色域的显示器中还原视频源的色彩，已成为高性能显示设备设计的关键难题。

国际照明委员会建立了标准色度学系统，这位颜色表达和转换提供了统一的数学框架。比如 CIE1931 色度图，可以将不同设备的色欲覆盖可视化。BT2020 色彩空间是一种标准的高清视频源的三基色色空间。BT2020 具有更广的色域范围，通常用于高动态范围视频何超高清电视的显示。而现实中普通显示屏色彩空间，诸如 sRGB、NTSC 等通常色域更小。这通常会导致部分高色彩饱和度区域无法准确重建，从而形成色彩损失问题。

针对上述问题，工业界提出多通道拓展方案，将视频源引入第四个颜色通道 V（RGBV）拓宽了记录色域，而显示设备则拓展为五通道（RGBCX）以提升色彩重现能力。因而如何设计从高色域空间到显示器色域的映射函数，使得色彩失真最小，是提升色彩显示能力的关键任务。

此外被广泛使用的 LED 显示器，因其本身的制造差异、驱动电路的非线性影响等因素会导致整屏颜色显示不一致。这导致了显示效果的非一致性会严重影响视觉体验。因此基于颜色空间转换与匹配原理，合理构建映射函数和校正策略，对 LED 像素点颜色进行精细调控，从而实现整屏一致性的色彩校正，已成为提升 LED 显示品质的重要手段。

1.2 问题提出与研究内容

1.2.1 问题一

本问题的核心在于实现不同色域之间的映射。BT2020 色域更广，而 sRGB 色域相对较小。二者在色度坐标、亮度范围等方面存在较大差异，直接映射会导致显示器难以还原视频源的颜色，进而损失色彩，还会导致失真、亮度饱和度损失等问题。因此需要定义合适的转换损失函数，减小色彩损失。因此在映射过程中应当选择合适的损失函数，保证转换后的色彩贴合人眼视觉特性，提高感知效果。选择损失函数后还应当采用梯度下降法或基于样本的非线性最小二乘法进行求解。

2 问题分析

2.1 问题重述

2.2 模型假设

2.3 符号定义

2.4 理论基础

为了便于对后续视频源 BT.2020 色域与普通显示屏 RGB 色域之间映射关系的分析，我们首先引入标准色度系统的数学模型，对常见色彩空间进行建模表示。这些空间构成了本问题中色彩转换和损失评估的基础框架。

2.4.1 CIE1931 标准色度观察者与光谱三刺激值

CIE 1931 是是由国际照明委员会（CIE）于 1931 年定义的色彩模型，其核心在于基于实验测量建立的“标准色度观察者”响应曲线。这一模型通过三条匹配函数 $\overline{x}(\lambda), \overline{y}(\lambda), \overline{z}(\lambda)$ 将任意波长下的光谱功率分布（SPD）映射为三刺激值（Tristimulus Values）：

$$X = \int_{\lambda} S(\lambda)\overline{x}(\lambda)d\lambda, \quad Y = \int_{\lambda} S(\lambda)\overline{y}(\lambda)d\lambda, \quad Z = \int_{\lambda} S(\lambda)\overline{z}(\lambda)d\lambda \tag{2.1}$$

2.4.2 CIEXYZ 颜色空间

CIEXYZ 是一个以三刺激值为基础的线性色彩空间，被视为“设备无关”的色彩表示方式。其三个分量 (X, Y, Z) 分别对应红、绿、蓝三种感知通道。Y 分量也通常用作**亮度（Luminance）**的代表。该空间是许多其他色彩空间（如 Lab、sRGB、BT.2020）的中间标准基础。通常不同色域之间的转换以此为中介。

2.4.3 CIELab 颜色空间

CIELab 空间是基于 CIEXYZ 空间定义的感知均匀色彩空间，能够更好地符合人眼对颜色差异的敏感性。其由以下三个分量构成：

$$L^*\ ,\ a^*\ ,\ b^P* \tag{2.2}$$

其中，L^* 代表明度，a^* 代表红绿轴，b^* 代表黄蓝轴。具体变换公式如下（以 D65 白点为例）：

$$f(t) = \begin{cases} t^{\frac{1}{3}} & t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & t \le \delta^3 \end{cases}, \quad \delta = \frac{6}{29} \tag{2.3}$$

$$L^* = 116f(\frac{Y}{Y_n}) - 16, \quad a^* = 500[f(\frac{X}{X_n}) - f(\frac{Y}{Y_n})], \quad b^* = 200[f(\frac{Y}{Y_n}) - f(\frac{Z}{Z_n})] \tag{2.4}$$

其中 (X_n, Y_n, Z_n) 为参考白点（如 D65）的三刺激值。

2.4.4 CIE1931xy 色度图与色域表示

CIEXYZ 空间中颜色可以通过如下变换得到色度图中的坐标：

$$x = \frac{X}{X+Y+Z}, \ y = \frac{Y}{X+Y+Z} \tag{2.5}$$

该色度图表示了所有可见光的二维投影范围，设备的色域可以通过其三基色的 (x, y) 点连线形成三角形表示。色域越大，所能表示的颜色越丰富。该图是色彩匹配与色彩损失分析的重要工具。

2.4.5 CIEDE2000 色差公式

为了更精确的对问题进行建模并且便于后续损失函数以及差分进化算法的实现，我们将题目中的 BT.2020 颜色空间以及显示屏的颜色空间从 xy 色度坐标转换为 XYZ 颜色空间，再利用 Lab 颜色空间公式转换为 (L^*, a^*, b^*) 。最后计算 ΔE_{00} 损失值。

在将 BT.2020 高清视频源的色彩空间映射至普通显示屏 RGB 色域时，由于显示设备色域较小，无法完整覆盖原始色域，导致部分颜色无法被准确再现。因此，我们需要设计一个合理的**色彩转换映射矩阵** $M \in \mathbb{R}^{3\times3}$, 以最小化从 BT.2020 色域到显示屏色域的映射过程中所产生的**主观感知误差**。

为度量这一色彩差异，应选择符合人眼视觉感知的度量方式。传统的欧几里得差异（如 RGB 或 XYZ 空间中的 L2 距离）不能很好地反映颜色感知误差。我们引入国际照明委员会（CIE）推荐的 ΔE_{00} 作为感知误差的度量函数。

对任意两个颜色在 Lab 空间中的向量：

$$Lab_1 = (L_1^*, a_1^*, b_1^*), \quad Lab_2 = (L_2^*, a_2^*, b_2^*) \tag{2.6}$$

ΔE_{00} 的计算公式如下：

$$\Delta E_{00} = \sqrt{(\frac{\Delta L'}{k_L S_L})^2 + (\frac{\Delta C'}{k_C S_C})^2 + (\frac{\Delta H'}{k_H S_H})^2 + R_T \cdot (\frac{\Delta C'}{k_C S_C}) \cdot (\frac{\Delta H'}{k_H S_H})} \tag{2.7}$$

（1）明度差与平均明度

$$\Delta L' = L_2^* - L_1^*, \ \overline{L} = \frac{L_2^* - L_1^*}{2} \tag{2.8}$$

（2）色度差与平均色度

$$C_1 = \sqrt{a_1^{*2} + b_1^{*2}},\ C_2 = \sqrt{a_2^{*2} + b_2^{*2}},\ \Delta C' = C_2 - C_1,\ \overline{C} = \frac{C_1 + C_2}{2} \tag{2.9}$$

（3）色相角差与平均色相角

$$\begin{aligned} &h_1 = \arctan 2(b_1^*, a_1^*),\ h_2 = \arctan 2(b_2^*, a_2^*) \\ &\Delta h' = h_2 - h_1,\ \Delta H^1 = 2\sqrt{C_1 C_2}\sin(\frac{\Delta h'}{2}) \\ &\overline{h} = \begin{cases} \frac{h_1+h_2}{2},\ |h_1 - h_2| > 180^\circ \\ \frac{h_1+h_2+360^\circ}{2},\ |h_1 - h_2| \le 180^\circ \end{cases} \end{aligned} \tag{2.10}$$

（4）调整因子

$$\begin{aligned} &G = 0.5(1 - \sqrt{\frac{\overline{C^7}}{\overline{C^7} + 25^7}}) \\ &T = 1 - 0.17\cos(\overline{h} - 30^\circ) + 0.24\cos(2\overline{h}) + 0.32\cos(3\overline{h} + 6^\circ) - 0.20\cos(4\overline{h} - 63^\circ) \end{aligned} \tag{2.11}$$

（5）权重因子

$$S_L = 1 + \frac{0.015(L - 50)^2}{\sqrt{20 + (\overline{L} - 50)^2}},\ S_C = 1 + 0.045\overline{C},\ S_H = 1 + 0.015\overline{C}T \tag{2.12}$$

（6）旋转补偿因子

$$R_T = -\sin(2\Delta\theta) \cdot R_C,\ \Delta\theta = 30\exp\{-(\frac{\overline{h} - 275^\circ}{25})^2\},\ R_C = 2\sqrt{\frac{\overline{C^7}}{\overline{C^7} + 25^7}} \tag{2.13}$$

其中：$\Delta L'$：明度差，$\Delta C'$：色度差，$\Delta H'$：色相差，S_L, S_C, S_H：感知缩放因子，$k_L = k_C = k_H = 1$：常用单位权重。

由上述公式，可以计算出两个 CIELab 值的色差。该函数对人眼感知差异具有良好拟合性能，因此被广泛用于图像质量、颜色匹配等领域。

3 模型建立与求解

3.1 问题 1：颜色空间转换模型

3.1.1 模型建立与求解

为求解 BT.2020 空间到显示屏 RGB 空间的最优线性映射矩阵 $M \in \mathbf{R}^{3\times3}$，我们采样一组代表性 BT.2020 RGB 样本 $\{c_i\}_{i=1}^{N} \in [0,1]^3$，其色彩向量经过如下映射：

$$c_i^{'} = M \cdot C_i \tag{3.1}$$

然后分别映射至 CIELab 空间，并计算感知误差：

$$L(M) = \frac{1}{N}\sum_{i=1}^{N} \Delta E_{00}(Lab(M_{BT\to XYZ}\cdot c_i), Lab(M_{DP\to XYZ}\cdot(M\cdot c_i))) \tag{3.2}$$

优化目标为：

$$\underset{M\in\mathbb{R}^{3\times3}}{min}\, L(M) \tag{3.3}$$

为求解上述非线性、不可导且可能存在多个局部极小值的优化问题，我们引入差分进化（Differential Evolution, DE）算法。DE 是一种基于种群的全局优化方法，具有较强的鲁棒性与跳出局部最优的能力。

（1）参数编码与搜索空间将 $M \in (R^{3\times3})$ 展开为 9 维向量 $\mathbf{x} \in \mathbb{R}^{\nrightarrow}$，并定义搜索空间边界为：

$$x_j \in [-2,2]m \;\; j = 1,...,9 \tag{3.4}$$

（2）初始化种群生成 NP 个个体 $\mathbf{x}_i^{(0)} \in \mathbb{R}^{\nrightarrow}$：

$$x_{i,j}^{(0)} = l_j + r_{i,j}\cdot(u_j - l_j),\;\; r_{i,j} \sim \mathcal{U}(0,1) \tag{3.5}$$

（3）变异操作对第 i 个个体，在不同个体中随机选择 $\mathbf{x}_{r1}, \mathbf{x_{r2}}, \mathbf{x_{r3}}$，构造差分向量：

$$\mathbf{v_i} = \mathbf{x_{r1}} + F\cdot(\mathbf{x_{r2}} - \mathbf{x_{r3}}) \tag{3.6}$$

其中 $F \in (0,2)$ 是差分缩放因子，控制探索强度。

（4）交叉操作构造试验个体 $\mathbf{u_i}$：

$$u_{i,j} = \begin{cases} v_{i,j}, & rand_j < CR \; or \; j = j_{rand} \\ x_{i,j}^{t}, & otherwise \end{cases} \tag{3.7}$$

其中 $CR \in [0,1]$ 为交叉概率，j_{rand} 确保至少一维来自 $\mathbf{v_i}$。

（5）选择操作通过目标函数比较试验解与当前个体，选择保留更优者：

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{u}_i, & L(\mathbf{u_i}) < L(\mathbf{x_i^{(t)}}) \\ \mathbf{x}_i^{(t)}, & otherwise \end{cases} \tag{3.8}$$

综上所述，为优化色彩转换矩阵 M，本文选用差分进化算法（DE）。该方法将 M 参数化为 9 维向量，并在预设的搜索空间边界内进行优化。通过其经典的种群初始化、变异、交叉及选择等核心操作，DE 算法能够迭代地搜寻旨在最小化以 ΔE_{00} 度量的感知色彩差异的解。鉴于目标函数的非线性、不可导以及可能存在多个局部极小值的特性，DE 算法的全局优化能力和鲁棒性，使其成为获取高质量色彩映射的有效计算途径。

3.1.2 问题一结果分析

为实现 BT.2020 色域向目标显示屏 RGB 色域的最优映射，本文构建了感知误差最小化的优化模型，目标为在 CIELab 空间中最小化 ΔE_{00} 感知色差。我们采样了多个 BT.2020 RGB 颜色点，并通过线性映射矩阵 $M \in \mathbb{R}^{3\times3}$ 变换后, 转化至目标显示屏空间，再经过标准变换矩阵应设至 XYZ、CIELab 空间，并利用 ΔE_{00} 公式计算感知误差。

在模型求解过程中，本文采用了差分进化（Differential Evolution, DE）优化方法，对初始映射矩阵进行迭代寻优。为验证我们模型的稳定性，并提供更可靠的性能评价，我们执行了 50 轮随机优化，并统计其性能指标。主要分析结果如下：

（1）ΔE_{00} 感知误差分布

图 1 显示了在 50 次独立优化实验中，各次优化所达成的最终 ΔE_{00} 损失值分布情况。其中最大值为 1.0183 ，这表明映射结果在感知层面极为接近参考目标。均值为 0.0744 ，标准差为 0.2083 。这表明该基于 ΔE_{00} 损失函数的差分进化算法在不同采样条件下都能稳定收敛于较小的感知误差区域，并且具有良好的泛化性能以及良好的稳定性和鲁棒性。

（2）色度空间三角形面积变化

为评估映射后色域覆盖度变化，我们进一步对比了 sRGB 色度三角与模型输出映射后所得的色度三角面积。面积通过三角形在 CIE xy 色度图上的顶点（RGB 基色经映射后的 xy 坐标）计算而得。结果表明，所有 50 次优化中，面积差绝对值均低于 0.001，说明映射后色域几乎无压缩，色彩覆盖极小损失。显然我们可以得出，模型在保持色域范围完整性的同时，完成了精准的 RGB 空间映射，并且与 sRGB 的覆盖几乎一致，无明显压缩或扭曲现象。映射后的面积误差控制在 10^{-3} 量级，说明模型不仅保持了色彩准确性，也很好地保留了 BT.2020 色域映射后的

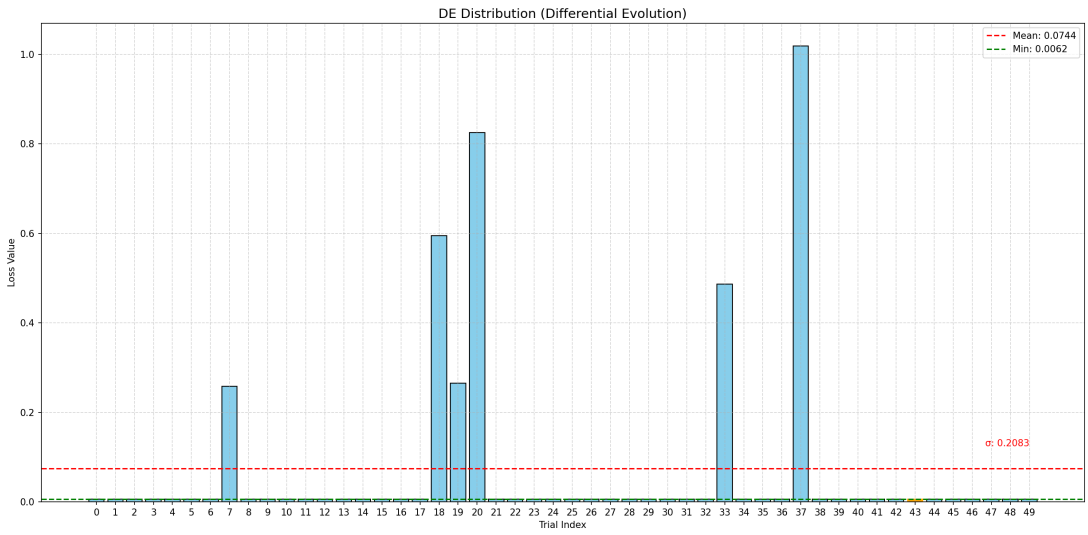

图 3.1

Fig. 3.1

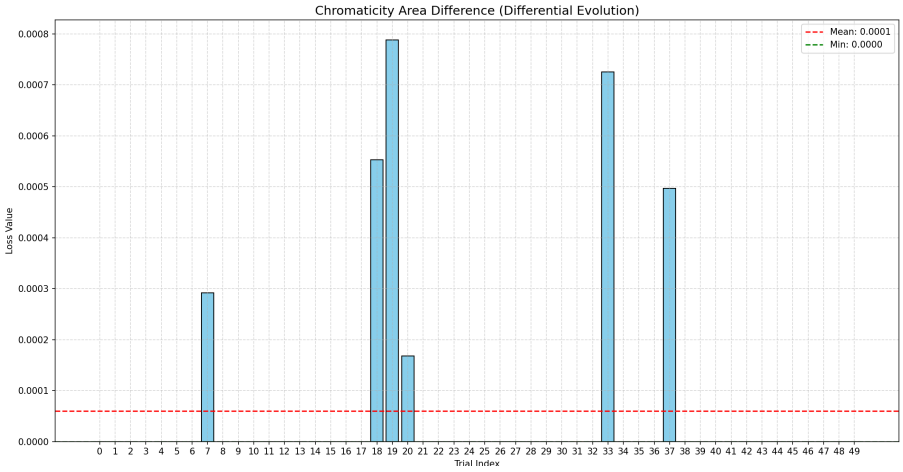

图 3.2

Fig. 3.2

覆盖特性。

（3）色度图可视化对比

为直观评估映射效果，我们将 BT.2020、sRGB 以及映射后所得色度三角同时绘制于 CIE 1931 xy 色度图中（见图 3）。可以观察到，模型优化后所得色度三角与标准 sRGB 色域几乎完全重合，进一步验证了在极低感知误差下，实现了对目标色域的高保真拟合。

3.2 问题 2：四通道到五通道颜色转换模型

3.3 问题 3：LED 显示器颜色校正模型

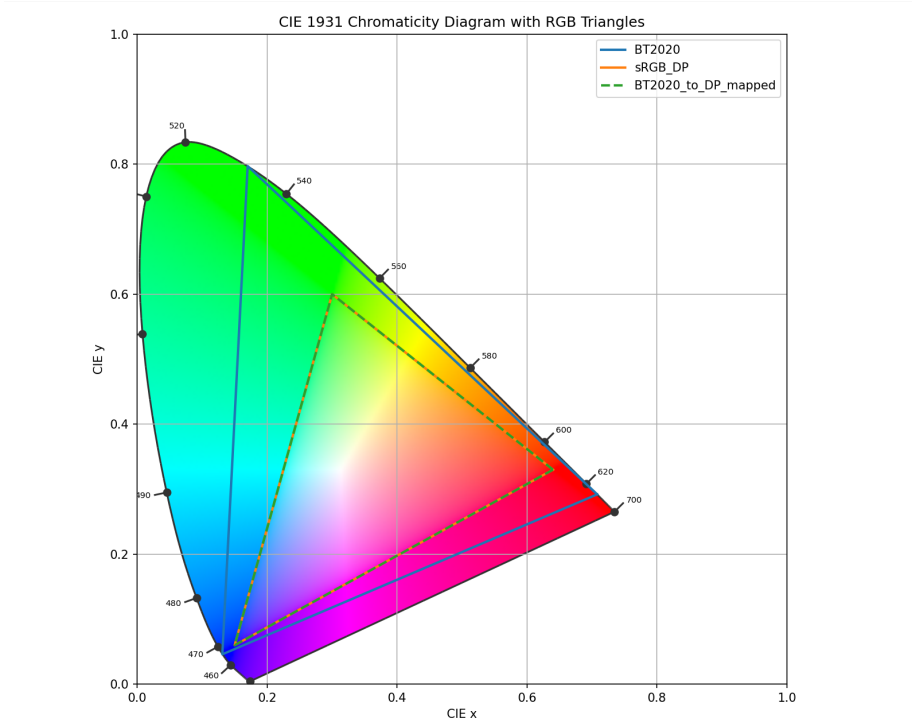
CIE 1931 Chromaticity Diagram with RGB Triangles
BT2020
sRGB_DP
BT2020_to_DP_mapped
520
540
550
580
600
620
700
490
480
470
460
CIE y
CIE x
0.0
0.2
0.4
0.6
0.8
1.0

图 3.3

Fig. 3.3

4 模型评价与推广

4.1 主要结论

4.2 模型优点

4.3 不足与改进方向

参考文献

[1] Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks [C]//International Conference on Machine Learning. 2017: 1126-1135.

附 录

A. 问题 1 使用代码

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from scipy.optimize import differential_evolution
import matplotlib.pyplot as plt
import colour
from colormath.color_objects import LabColor, XYZColor
from colormath.color_conversions import convert_color

BT2020 = [[0.708, 0.292], [0.170, 0.797], [0.131, 0.046]]
sRGB_DP = [[0.64, 0.33], [0.30, 0.60], [0.15, 0.06]]
NTSC = [[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]]

M_sRGB_to_XYZ = np.array([
    [0.4124564, 0.3575761, 0.1804375],
    [0.2126729, 0.7151522, 0.0721750],
    [0.0193339, 0.1191920, 0.9503041]
])

def lab_to_xyz_batch(lab_array):
    result = []
    for lab in lab_array:
        lab_color = LabColor(*lab)
        xyz_color = convert_color(lab_color, XYZColor)
        result.append([xyz_color.xyz_x, xyz_color.xyz_y, xyz_color.xyz_z])
    return np.array(result)

def rgb_to_xy(rgb, M_rgb_to_xyz):
    xyz = rgb @ M_rgb_to_xyz.T
    x = xyz[:, 0] / (xyz[:, 0] + xyz[:, 1] + xyz[:, 2])
    y = xyz[:, 1] / (xyz[:, 0] + xyz[:, 1] + xyz[:, 2])
    return np.stack([x, y], axis=1)

def xyz_to_xy_test(M_opt, RGB_basic, M_bt2020_to_xyz):
    # BT2020 to DP
    M_opt_inv = np.linalg.inv(M_opt)
    dp_rgb_mapped = (M_opt_inv @ RGB_basic.T).T  # shape (3, 3)
    BT2020_to_DP_mapped = rgb_to_xy(dp_rgb_mapped, M_bt2020_to_xyz)
    return BT2020_to_DP_mapped

def chromaticity_to_xyz_matrix(primaries, whitepoint):
    M = []
    for x, y in primaries:
```

```
44          z = 1 - x - y
45          M.append([x / y, 1.0, z / y])
46      M = np.array(M).T
47      Xw, Yw, Zw = whitepoint
48      S = np.linalg.inv(M) @ np.array([Xw, Yw, Zw])
49      return M * S
50
51  def delta_e_00_batch(lab1, lab2):
52      lab1 = np.array(lab1)
53      lab2 = np.array(lab2)
54
55      L1, a1, b1 = lab1[:, 0], lab1[:, 1], lab1[:, 2]
56      L2, a2, b2 = lab2[:, 0], lab2[:, 1], lab2[:, 2]
57
58      avg_L = 0.5 * (L1 + L2)
59      C1 = np.sqrt(a1**2 + b1**2)
60      C2 = np.sqrt(a2**2 + b2**2)
61      avg_C = 0.5 * (C1 + C2)
62
63      G = 0.5 * (1 - np.sqrt((avg_C**7) / (avg_C**7 + 25**7)))
64      a1p = (1 + G) * a1
65      a2p = (1 + G) * a2
66      C1p = np.sqrt(a1p**2 + b1**2)
67      C2p = np.sqrt(a2p**2 + b2**2)
68      avg_Cp = 0.5 * (C1p + C2p)
69
70      h1p = np.degrees(np.arctan2(b1, a1p)) % 360
71      h2p = np.degrees(np.arctan2(b2, a2p)) % 360
72
73      deltahp = h2p - h1p
74      deltahp = np.where(deltahp > 180, deltahp - 360, deltahp)
75      deltahp = np.where(deltahp < -180, deltahp + 360, deltahp)
76
77      delta_Hp = 2 * np.sqrt(C1p * C2p) * np.sin(np.radians(deltahp / 2))
78      delta_Lp = L2 - L1
79      delta_Cp = C2p - C1p
80
81      avg_hp = np.where(np.abs(h1p - h2p) > 180, (h1p + h2p + 360) / 2, (h1p +
            ↪ h2p) / 2)
82      T = 1 - 0.17 * np.cos(np.radians(avg_hp - 30)) + 0.24 * np.cos(np.radians
            ↪ (2 * avg_hp)) \
83          + 0.32 * np.cos(np.radians(3 * avg_hp + 6)) - 0.20 * np.cos(np.radians
                ↪ (4 * avg_hp - 63))
84
85      delta_theta = 30 * np.exp(-((avg_hp - 275) / 25)**2)
86      Rc = 2 * np.sqrt((avg_Cp**7) / (avg_Cp**7 + 25**7))
87      Sl = 1 + (0.015 * (avg_L - 50)**2) / np.sqrt(20 + (avg_L - 50)**2)
88      Sc = 1 + 0.045 * avg_Cp
```

```
    Sh = 1 + 0.015 * avg_Cp * T
    Rt = -np.sin(np.radians(2 * delta_theta)) * Rc

    delta_E = np.sqrt(
        (delta_Lp / Sl)**2 +
        (delta_Cp / Sc)**2 +
        (delta_Hp / Sh)**2 +
        Rt * (delta_Cp / Sc) * (delta_Hp / Sh)
    )

    return delta_E

def f(t):
    delta = 6/29
    return np.where(t > delta**3, np.cbrt(t), (t / (3 * delta**2)) + (4/29))

def xyz_to_lab_batch(xyz, white_point=(0.95047, 1.00000, 1.08883)):
    Xn, Yn, Zn = white_point
    X = xyz[:, 0] / Xn
    Y = xyz[:, 1] / Yn
    Z = xyz[:, 2] / Zn

    fx = f(X)
    fy = f(Y)
    fz = f(Z)

    L = 116 * fy - 16
    a = 500 * (fx - fy)
    b = 200 * (fy - fz)

    return np.stack([L, a, b], axis=1)

def combined_loss(M_flat, rgb_samples, M_bt2020_to_xyz, M_dp_to_xyz,
    ↪ xyz_to_lab_batch):

    # 变换矩阵
    M = M_flat.reshape(3, 3)

    # ═══ ΔE00 感知损失 ═══
    rgb_dp = rgb_samples @ M.T
    rgb_dp = np.clip(rgb_dp, 0, 1)

    xyz_pred = rgb_dp @ M_dp_to_xyz.T
    lab_pred = xyz_to_lab_batch(xyz_pred)

    xyz_true = rgb_samples @ M_bt2020_to_xyz.T
    lab_true = xyz_to_lab_batch(xyz_true)

```

```
    deltaE = delta_e_00_batch(lab_true, lab_pred)
    color_loss = np.mean(deltaE)

    return color_loss

def optimize_model_N_times(whitepoint, sRGB_DP, M_flat_init, M_bt2020_to_xyz,
    ↪ M_dp_to_xyz, xyz_to_lab_batch,
                           N=10, method='DE', random_seed_offset=31):
    """
    对比不同优化器在 N 轮随机样本下的表现

    参数:
    - M_flat_init: 初始 M(flatten)
    - M_bt2020_to_xyz: BT2020 → XYZ 变换矩阵
    - M_dp_to_xyz: DP → XYZ 变换矩阵
    - xyz_to_lab_batch: XYZ → Lab 转换函数(批量)
    - N: 循环次数
    - method: 优化方法选择, 'L-BFGS-B' 或 'DE'
    - random_seed_offset: 随机种子偏移量, 确保每轮样本不同但可复现

    返回:
    - losses: ndarray[N], 每轮优化得到的 loss
    """

    losses = []
    area_diffs = []
    RGB_basic = np.eye(3)
    ref_area = triangle_area(sRGB_DP)

    for i in range(N):
        seed = i + random_seed_offset
        np.random.seed(seed)
        test_samples = np.random.rand(100, 3)

        def loss_fn(M_flat):
            return combined_loss(M_flat, test_samples, M_bt2020_to_xyz,
                ↪ M_dp_to_xyz, xyz_to_lab_batch)

        if method == 'DE':
            bounds = [(-2, 2)] * 9
            res = differential_evolution(
                loss_fn,
                bounds,
                strategy='best1bin',
                maxiter=1000,
                polish=True,
                seed=seed
            )
```

```
182             else:
183                 raise ValueError(f"Unknown method: {method}. Supported: 'DE'")
184 
185             M_opt = res.x.reshape(3, 3)
186             # ==========================
187             BT2020_to_DP_mapped = xyz_to_xy_test(M_opt, RGB_basic, M_bt2020_to_xyz
                    ↪ )
188             BT_mapped_xyz = chromaticity_to_xyz_matrix(BT2020_to_DP_mapped,
                    ↪ whitepoint)
189             BT_mapped_lab = xyz_to_lab_batch(BT_mapped_xyz)
190             BT_lab = xyz_to_lab_batch(M_sRGB_to_XYZ)
191             loss = np.mean(delta_e_00_batch(BT_mapped_lab, BT_lab))
192             # ==========================
193             # final_loss = loss_fn(res.x)
194             losses.append(loss)
195 
196             triangle_xy = xyz_to_xy_test(M_opt, RGB_basic, M_bt2020_to_xyz)
197             area = triangle_area(triangle_xy)
198             area_diff = abs(area - ref_area)
199             area_diffs.append(area_diff)
200 
201     return np.array(losses), np.array(area_diffs)
202 
203 def triangle_area(pts):
204     """
205     计算三角形面积: pts 是 3x2 的 xy 坐标矩阵
206     使用 Shoelace formula（鞋带公式）
207     """
208     pts = np.array(pts)
209     x = pts[:, 0]
210     y = pts[:, 1]
211     return 0.5 * abs(x[0]*(y[1]-y[2]) + x[1]*(y[2]-y[0]) + x[2]*(y[0]-y[1]))
212 
213 
214 def plot_chromaticity_with_triangles(example_dict):
215     """
216     在 CIE 1931 xy 色度图上叠加多个 RGB 三角形。
217     前两个三角形为实线，后续为虚线，图例使用变量名。
218     """
219     figure, axes = colour.plotting.plot_chromaticity_diagram_CIE1931(
            ↪ standalone=False)
220 
221     colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
222     linestyle_solid = '-'
223     linestyle_dashed = '--'
224 
225     for i, (label, triangle) in enumerate(example_dict.items()):
226         triangle = np.array(triangle)
```

```
227         polygon = np.vstack([triangle, triangle[0]])
228         linestyle = linestyle_solid if i < 2 else linestyle_dashed
229 
230         axes.plot(polygon[:, 0], polygon[:, 1],
231                   color=colors[i % len(colors)],
232                   linewidth=2,
233                   linestyle=linestyle,
234                   label=label)
235 
236     axes.legend()
237     axes.set_title("CIE 1931 Chromaticity Diagram with RGB Triangles")
238     plt.grid(True)
239     plt.show()
240 
241 
242 def plot_loss_statistics(losses, title='Loss Distribution', method_name='L-
        ↪ BFGS-B'):
243     """
244     绘制柱状图并显示统计信息。
245 
246     参数:
247     - losses: 一维 ndarray, 优化 N 次的 loss 值
248     - title: 图表标题
249     - method_name: 优化方法名称, 用于图表显示
250     """
251 
252     # 计算统计量
253     mean_loss = np.mean(losses)
254     min_loss = np.min(losses)
255     std_loss = np.std(losses)
256 
257     # 创建柱状图
258     plt.figure(figsize=(10, 6))
259     bars = plt.bar(range(len(losses)), losses, color='skyblue', edgecolor='
            ↪ black')
260 
261     # 高亮最小值
262     min_index = np.argmin(losses)
263     bars[min_index].set_color('orange')
264 
265     # 标注统计量
266     plt.axhline(mean_loss, color='red', linestyle='--', label=f'Mean: {
            ↪ mean_loss:.4f}')
267     plt.axhline(min_loss, color='green', linestyle='--', label=f'Min: {
            ↪ min_loss:.4f}')
268     plt.text(len(losses) - 1, mean_loss + 0.05, f' : {std_loss:.4f}', color='
            ↪ red', fontsize=10, ha='right')
269 
```

```
    # 图形美化
    plt.title(f'{title} ({method_name})', fontsize=14)
    plt.xlabel('Trial Index')
    plt.ylabel('Loss Value')
    plt.xticks(range(len(losses)))
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

if __name__ ==”__main__”:
    # D65 whitepoint in XYZ
    whitepoint = (0.3127 / 0.3290, 1.0, (1 - 0.3127 - 0.3290) / 0.3290)
    # BT2020 → XYZ
    M_bt2020_to_xyz = chromaticity_to_xyz_matrix(BT2020, whitepoint)
    # DP/sRGB → XYZ
    M_dp_to_xyz = chromaticity_to_xyz_matrix(sRGB_DP, whitepoint)

    # ========== 训练部分 ==========
    # 采样一组 BT.2020 RGB 样本 {c_i}
    M0 = np.eye(3).flatten()
    M0_flat = np.eye(3).flatten()

    # L-BFGS-B 优化 50 次
    losses_lbfgs, area_diffs = optimize_model_N_times(
        whitepoint,
        sRGB_DP,
        M0_flat,
        M_bt2020_to_xyz,
        M_dp_to_xyz,
        xyz_to_lab_batch,
        N=50,
        method='DE'
    )
    print(”DE Losses:”, losses_lbfgs)

    plot_loss_statistics(losses_lbfgs, title='DE Distribution', method_name='
        ↪ Differential Evolution')
    plot_loss_statistics(area_diffs, title='Chromaticity Area Difference',
        ↪ method_name='Differential Evolution')
    # ========== 单独测试 ==========
    np.random.seed(35)
    test_samples = np.random.rand(100, 3)
    bounds = [(-2, 2)] * 9
    def loss_fn(M_flat):
        return combined_loss(M_flat, test_samples, M_bt2020_to_xyz,
            ↪ M_dp_to_xyz, xyz_to_lab_batch)
```

```
315
316     res1 = differential_evolution(
317         loss_fn,
318         bounds,
319         strategy='best1bin',
320         maxiter=1000,
321         polish=True,
322         seed=35,
323         )
324     M_opt = res1.x.reshape(3, 3)
325     # 映射到色度图上
326     # DB to BT2020
327     RGB_basic = np.eye(3)
328     # BT2020 to DP
329     BT2020_to_DP_mapped = xyz_to_xy_test(M_opt, RGB_basic, M_bt2020_to_xyz)
330
331     examples = {
332         "BT2020": BT2020,
333         "sRGB_DP": sRGB_DP,
334         "BT2020_to_DP_mapped": BT2020_to_DP_mapped
335     }
336
337     plot_chromaticity_with_triangles(examples)
```

B. 问题 2 使用代码

```
1 def greet(name):
2     print(f"Hello, {name}!")
3
4 greet("ChatGPT")
```

C. 问题 3 使用代码

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from scipy.optimize import minimize, differential_evolution
5
6  # 设置中文字体
7  plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
8  plt.rcParams['axes.unicode_minus'] = False
9
10 class LEDColorCorrection:
11     """
12     基于三基色原理和CIE Lab色彩空间的颜色校正
13     使用差分进化算法优化校正矩阵
14     """
```

```
15
16      def __init__(self):
17          self.correction_matrix = None
18          self.correction_bias = None
19          self.gamma_correction = None
20          self.measured_data = None
21          self.target_data = None
22
23      def load_excel_data(self, excel_path):
24          """从Excel文件加载数据"""
25          print(f"正在加载Excel文件: {excel_path}")
26
27          sheets = ['R', 'G', 'B', 'target_R', 'target_G', 'target_B']
28          data_dict = {}
29
30          for sheet_name in sheets:
31              df = pd.read_excel(excel_path, sheet_name=sheet_name, header=None)
                    ↪ .iloc[0:64,0:64]
32              data_dict[sheet_name] = df.values
33              print(f"已加载工作表 '{sheet_name}': {df.shape}")
34
35          # 组织数据
36          self.measured_data = np.stack([
37              data_dict['R'],
38              data_dict['G'],
39              data_dict['B']
40          ], axis=-1)
41
42          self.target_data = np.stack([
43              data_dict['target_R'],
44              data_dict['target_G'],
45              data_dict['target_B']
46          ], axis=-1)
47
48          print(f"测量数据形状: {self.measured_data.shape}")
49          print(f"目标数据形状: {self.target_data.shape}")
50
51      def rgb_to_xyz(self, rgb):
52          """RGB转XYZ色彩空间"""
53          rgb_norm = rgb / 255.0
54
55          # Gamma校正
56          rgb_linear = np.where(rgb_norm <= 0.04045,
57                                rgb_norm / 12.92,
58                                np.power((rgb_norm + 0.055) / 1.055, 2.4))
59
60          # sRGB到XYZ的转换矩阵
61          transform_matrix = np.array([
```

```
            [0.4124564, 0.3575761, 0.1804375],
            [0.2126729, 0.7151522, 0.0721750],
            [0.0193339, 0.1191920, 0.9503041]
        ])

        xyz = np.dot(rgb_linear, transform_matrix.T)
        return xyz

    def xyz_to_lab(self, xyz):
        """XYZ转CIE Lab色彩空间"""
        # D65白点
        Xn, Yn, Zn = 0.95047, 1.00000, 1.08883

        x = xyz[..., 0] / Xn
        y = xyz[..., 1] / Yn
        z = xyz[..., 2] / Zn

        # 立方根变换
        fx = np.where(x > 0.008856, np.power(x, 1/3), (7.787 * x + 16/116))
        fy = np.where(y > 0.008856, np.power(y, 1/3), (7.787 * y + 16/116))
        fz = np.where(z > 0.008856, np.power(z, 1/3), (7.787 * z + 16/116))

        L = 116 * fy - 16
        a = 500 * (fx - fy)
        b = 200 * (fy - fz)

        return np.stack([L, a, b], axis=-1)

    def calculate_color_difference(self, lab1, lab2):
        """计算CIE Delta E 2000色差"""
        L1, a1, b1 = lab1[..., 0], lab1[..., 1], lab1[..., 2]
        L2, a2, b2 = lab2[..., 0], lab2[..., 1], lab2[..., 2]

        C1 = np.sqrt(a1**2 + b1**2)
        C2 = np.sqrt(a2**2 + b2**2)
        C_bar = 0.5 * (C1 + C2)

        G = 0.5 * (1 - np.sqrt(C_bar**7 / (C_bar**7 + 25**7)))
        a1p = (1 + G) * a1
        a2p = (1 + G) * a2

        C1p = np.sqrt(a1p**2 + b1**2)
        C2p = np.sqrt(a2p**2 + b2**2)

        h1p = np.degrees(np.arctan2(b1, a1p)) % 360
        h2p = np.degrees(np.arctan2(b2, a2p)) % 360

        dLp = L2 - L1
```

```
        dCp = C2p - C1p

        dhp = h2p - h1p
        dhp = dhp - 360 * (dhp > 180) + 360 * (dhp < -180)
        dHp = 2 * np.sqrt(C1p * C2p) * np.sin(np.radians(dhp / 2))

        L_bar = 0.5 * (L1 + L2)
        C_bar_p = 0.5 * (C1p + C2p)

        h_bar_p = (h1p + h2p + 360 * (np.abs(h1p - h2p) > 180)) / 2
        h_bar_p %= 360

        T = (1
             - 0.17 * np.cos(np.radians(h_bar_p - 30))
             + 0.24 * np.cos(np.radians(2 * h_bar_p))
             + 0.32 * np.cos(np.radians(3 * h_bar_p + 6))
             - 0.20 * np.cos(np.radians(4 * h_bar_p - 63)))

        Sl = 1 + (0.015 * (L_bar - 50)**2) / np.sqrt(20 + (L_bar - 50)**2)
        Sc = 1 + 0.045 * C_bar_p
        Sh = 1 + 0.015 * C_bar_p * T

        delta_theta = 30 * np.exp(-((h_bar_p - 275)/25)**2)
        Rc = 2 * np.sqrt(C_bar_p**7 / (C_bar_p**7 + 25**7))
        Rt = -np.sin(np.radians(2 * delta_theta)) * Rc

        dE = np.sqrt(
             (dLp / Sl)**2 +
             (dCp / Sc)**2 +
             (dHp / Sh)**2 +
             Rt * (dCp / Sc) * (dHp / Sh)
        )

        return dE

    def estimate_gamma_parameters(self):
        """估计LED的Gamma参数（保留线性比例偏移）"""
        print("正在估计Gamma参数...")
        gamma_params = {}
        for i, channel in enumerate(['R', 'G', 'B']):
            meas = self.measured_data[..., i].flatten() / 255.0
            targ = self.target_data[..., i].flatten() / 255.0
            mask = (targ > 0.04) & (targ < 0.96) & (meas > 0)
            m = meas[mask]
            t = targ[mask]
            if len(m) > 0:
                # 拟合 log(m) = gamma * log(t) + offset
                A = np.vstack([np.log(t + 1e-8), np.ones_like(t)]).T
```

```
158                 gamma, offset = np.linalg.lstsq(A, np.log(m + 1e-8), rcond=
                        ↪ None)[0]
159                 gamma = float(np.clip(gamma, 0.1, 3.0))
160                 scale = float(np.exp(offset))
161             else:
162                 gamma, scale = 1.0, 1.0
163             gamma_params[channel] = {'gamma': gamma, 'scale': scale}
164             print(f"{channel}通道 Gamma: {gamma:.3f}, Scale: {scale:.3f}")
165         self.gamma_correction = gamma_params
166         return gamma_params
167
168     def apply_gamma_correction(self, rgb_data, inverse=False):
169         """应用Gamma校正：在归一化 [0,1] 空间先应用线性比例，再做幂运算"""
170         if self.gamma_correction is None:
171             return rgb_data
172         data = rgb_data.astype(np.float32) / 255.0
173         out = np.zeros_like(data)
174         for i, channel in enumerate(['R', 'G', 'B']):
175             gamma = self.gamma_correction[channel]['gamma']
176             scale = self.gamma_correction[channel]['scale']
177             ch = data[..., i]
178             if not inverse:
179                 # 前向：先比例，再幂
180                 tmp = ch * scale
181                 tmp = np.clip(tmp, 0.0, 1.0)
182                 out_ch = np.power(tmp, gamma)
183             else:
184                 # 反向：开幂，再去比例
185                 tmp = np.power(ch, 1.0 / gamma)
186                 out_ch = tmp / np.maximum(scale, 1e-8)
187             out[..., i] = np.clip(out_ch, 0.0, 1.0)
188         # 恢复到 [0,255]
189         return (out * 255.0).astype(rgb_data.dtype)
190
191     def correction_function(self, params, measured_lin, target_lin):
192         """
193         优化函数：线性校正矩阵 M 和偏置 b，params 长度 12。
194         corrected = clip(M @ measured + b, [0,1])
195         计算 ΔE  + 正则化。
196         """
197         M = params[:9].reshape(3,3)
198         b = params[9:].reshape(1,3)
199
200         # 应用矩阵和偏置
201         corr = np.dot(measured_lin, M.T) + b
202         corr = np.clip(corr, 0.0, 1.0)
203
204         # 转到 XYZ → Lab
```

```
205         transform = np.array([[0.4124564,0.3575761,0.1804375],
206                               [0.2126729,0.7151522,0.0721750],
207                               [0.0193339,0.1191920,0.9503041]])
208         tgt_xyz = np.dot(target_lin, transform.T)
209         corr_xyz = np.dot(corr, transform.T)
210         tgt_lab = self.xyz_to_lab(tgt_xyz.reshape(-1,3)).reshape(corr.shape)
211         corr_lab = self.xyz_to_lab(corr_xyz.reshape(-1,3)).reshape(corr.shape)
212
213         # 色差
214         deltaE = self.calculate_color_difference(tgt_lab, corr_lab)
215         loss = np.mean(deltaE)
216
217         # 矩阵正则 + 偏置正则
218         loss += 0.001 * (np.sum((M - np.eye(3))**2) + np.sum(b**2))
219         det = np.linalg.det(M)
220         if det <= 0 or abs(det) < 0.1:
221             loss += 1000.0
222         return loss
223
224
225     def calibrate_correction_matrix(self):
226         print("开始校正: 矩阵 + 偏置...")
227         self.estimate_gamma_parameters()
228         # 预处理: 线性化
229         meas = self.apply_gamma_correction(self.measured_data.astype(np.
                ↪ float32), inverse=True)/255.0
230         targ = self.apply_gamma_correction(self.target_data.astype(np.float32)
                ↪ , inverse=True)/255.0
231         meas_flat = meas.reshape(-1,3)
232         targ_flat = targ.reshape(-1,3)
233         # 差分进化优化 12 参数
234         bounds = [(-2,2)]*9 + [(-0.1,0.1)]*3
235         res = differential_evolution(
236             self.correction_function, bounds,
237             args=(meas_flat, targ_flat), maxiter=200, popsize=15, seed=42
238         )
239         x0 = res.x
240         # 局部 L-BFGS-B
241         local = minimize(
242             self.correction_function, x0, args=(meas_flat,targ_flat),
243             method='L-BFGS-B', options={'maxiter':500}
244         )
245         M_opt = local.x[:9].reshape(3,3)
246         b_opt = local.x[9:].reshape(3)
247         self.correction_matrix = M_opt
248         self.correction_bias = b_opt
249         print("校正完成; 矩阵行列式: ", np.linalg.det(M_opt))
250         print("偏置: ", b_opt)
```

```
251         return M_opt, b_opt
252
253
254     def apply_correction(self, input_rgb):
255         """应用带偏置的线性校正"""
256         lin = self.apply_gamma_correction(input_rgb.astype(np.float32),
                ↪ inverse=True)/255.0
257         flat = lin.reshape(-1,3)
258         corr = np.dot(flat, self.correction_matrix.T) + self.correction_bias
259         corr = np.clip(corr, 0.0, 1.0).reshape(input_rgb.shape)
260         out = (corr * 255.0).astype(np.float32)
261         final = self.apply_gamma_correction(out, inverse=False)
262         return final.astype(np.uint8)
263
264     def evaluate_correction(self):
265         """评估校正效果"""
266         corrected = self.apply_correction(self.measured_data.astype(np.float32
                ↪ ))
267
268         measured_xyz = self.rgb_to_xyz(self.measured_data.astype(np.float32))
269         corrected_xyz = self.rgb_to_xyz(corrected.astype(np.float32))
270         target_xyz = self.rgb_to_xyz(self.target_data.astype(np.float32))
271
272         measured_lab = self.xyz_to_lab(measured_xyz)
273         corrected_lab = self.xyz_to_lab(corrected_xyz)
274         target_lab = self.xyz_to_lab(target_xyz)
275
276         diff_before = self.calculate_color_difference(measured_lab, target_lab
                ↪ )
277         diff_after = self.calculate_color_difference(corrected_lab, target_lab
                ↪ )
278
279         print("="*50)
280         print("校正效果评估报告")
281         print("="*50)
282         print(f"校正前平均色差: {np.mean(diff_before):.3f}")
283         print(f"校正后平均色差: {np.mean(diff_after):.3f}")
284         print(f"色差改善: {np.mean(diff_before) - np.mean(diff_after):.3f}")
285         print(f"改善百分比: {((np.mean(diff_before) - np.mean(diff_after)) /
                ↪ np.mean(diff_before) * 100):.1f}%")
286         print(f"校正前最大色差: {np.max(diff_before):.3f}")
287         print(f"校正后最大色差: {np.max(diff_after):.3f}")
288         print(f"色差<1.0的像素比例: 校正前{np.mean(diff_before < 1.0)*100:.1f
                ↪ }%, 校正后{np.mean(diff_after < 1.0)*100:.1f}%")
289         print("="*50)
290
291         return corrected, diff_before, diff_after
292
```

```
293     def visualize_results(self):
294         """可视化校正结果"""
295         corrected_data = self.apply_correction(self.measured_data.astype(np.
                ↪ float32))
296 
297         fig, axes = plt.subplots(3, 4, figsize=(20, 15))
298 
299         # 第一行: 测量数据
300         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
301             im = axes[0, i].imshow(self.measured_data[:, :, i], cmap=color,
                    ↪ vmin=0, vmax=255)
302             axes[0, i].set_title(f'测量值 - {channel} 通道')
303             axes[0, i].axis('off')
304             plt.colorbar(im, ax=axes[0, i], fraction=0.046, pad=0.04)
305 
306         measured_rgb = np.clip(self.measured_data / 255.0, 0, 1)
307         axes[0, 3].imshow(measured_rgb)
308         axes[0, 3].set_title('测量值 - RGB合成')
309         axes[0, 3].axis('off')
310 
311         # 第二行: 目标数据
312         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
313             im = axes[1, i].imshow(self.target_data[:, :, i], cmap=color, vmin
                    ↪ =0, vmax=255)
314             axes[1, i].set_title(f'目标值 - {channel} 通道')
315             axes[1, i].axis('off')
316             plt.colorbar(im, ax=axes[1, i], fraction=0.046, pad=0.04)
317 
318         target_rgb = np.clip(self.target_data / 255.0, 0, 1)
319         axes[1, 3].imshow(target_rgb)
320         axes[1, 3].set_title('目标值 - RGB合成')
321         axes[1, 3].axis('off')
322 
323         # 第三行: 校正后数据
324         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
325             im = axes[2, i].imshow(corrected_data[:, :, i], cmap=color, vmin
                    ↪ =0, vmax=255)
326             axes[2, i].set_title(f'校正后 - {channel} 通道')
327             axes[2, i].axis('off')
328             plt.colorbar(im, ax=axes[2, i], fraction=0.046, pad=0.04)
329 
330         corrected_rgb = np.clip(corrected_data / 255.0, 0, 1)
331         axes[2, 3].imshow(corrected_rgb)
332         axes[2, 3].set_title('校正后 - RGB合成')
333         axes[2, 3].axis('off')
```

```
334
335         plt.tight_layout()
336         plt.show()
337
338
339 # 主函数
340 if __name__ == "__main__":
341     files = ["MathModel_Code\\data\\preprocess\\p3\\RedPicture.xlsx", "
        ↪ MathModel_Code\\data\\preprocess\\p3\\GreenPicture.xlsx", "
        ↪ MathModel_Code\\data\\preprocess\\p3\\BluePicture.xlsx"]
342
343     corrector = LEDColorCorrection()
344
345     for filepath in files:
346         corrector.load_excel_data(filepath)
347         correction_matrix = corrector.calibrate_correction_matrix()
348
349         print("\n评估校正效果:")
350         corrected_display, diff_before, diff_after = corrector.
            ↪ evaluate_correction()
351
352         corrector.visualize_results()
353
354         print("\n校正完成!")
```

D. 像素数据集