上海工程技术大学

第十 三届研究生数学建模竞赛

学　　院	电子电气工程学院
专　　业	电子信息 控制科学与工程
班　　级	控1　电3　电4
参赛队号	A202516029
队员姓名	1. 刘　钢
	2. 陈家涛
	3. 朱仁杰

上海工程技术大学

第十 三届研究生数学建模竞赛

题　目　　**LED 显示屏颜色转换设计与校正**

摘　　要：

针对 LED 显示设备与视频源颜色感知能力不一致导致的色彩失真问题，本文基于 CIE1931 色度学理论，构建了从色域映射到像素级校正的完整颜色转换模型。

针对问题一，建立 BT2020 广色域向 LED 显示屏 RGB 色域的最优映射模型。基于 CIE76 色差公式构建损失函数，采用差分进化算法求解最优转换矩阵。实验结果表明，优化后的 RGB 值与目标值偏差精度可达小数点后 8 位，色差接近零，成功实现了高保真色域映射。

针对问题二，解决四基色 RGBV 视频源向五基色 RGBCX 显示系统的多通道颜色空间映射问题。构建了低维到高维的线性映射优化模型，引入 F-范数正则化约束防止参数发散。通过差分进化算法求得 5×4 转换矩阵，实现了色彩重建误差低于 2%，同时将色域覆盖能力提升 41.06%。

针对问题三，设计 LED 显示屏像素级颜色校正方案。提出全局-区域分层颜色校正算法，通过联合优化均方误差与空间一致性约束，构建综合目标函数。在 64×64 显示模块上的实验表明，校正后 RGB 三通道的变异系数分别从 0.151、0.122、0.126 降低至 0.107、0.107、0.101，均匀性提升达 4.3%、1.5%、2.6%，准确度均超过 0.989，整体质量得分达到 0.942。

本研究提出的多层次颜色转换与校正方法，有效解决了 LED 显示系统中色域映射、多通道转换和像素一致性的关键技术问题，为高品质 LED 显示产品的工程应用提供了理论支撑和实践方案。

关键字：LED 显示屏；色域映射；多通道颜色空间；差分进化算法；颜色校正；CIE1931 色度学

目录

1 引言

1.1 背景

色彩是人类认知世界的重要信息载体。颜色的本质是光作用于人眼后，通过视网膜感光细胞和大脑神经信号处理形成的视觉感知现象。不同物体因材质和光照条件的不同，会反射或发射出不同波长的电磁波，从而呈现出丰富多彩的世界。随着数字化和信息化的发展，色彩的数字化采集与高保真显示成为现代显示技术的重要目标。

在色彩复现过程中，色彩采集设备（如光谱色差仪、摄像机）与显示设备（如 LED 显示屏）之间存在感知和还原能力的差异。显示设备如何尽可能真实地还原采集到的色彩，是高性能显示系统设计的核心难题。现代色度学采用 CIE（国际照明委员会）标准色度学系统，通过三基色原理（通常为红、绿、蓝）实现对自然界各种颜色的合成与表达。1931CIE-RGB 与 XYZ 颜色系统为色彩的数学表达和转换提供了理论基础。

1931CIE 色彩空间是国际上广泛采用的标准色度空间。其核心是 1931CIE 颜色匹配函数（CMFs），通过对人眼对不同波长光的响应进行实验测量，建立了颜色的三刺激值模型。XYZ 表色系统则通过线性变换，将 RGB 三刺激值转化为 XYZ 三刺激值，使所有颜色坐标均为正值，便于数学处理和工程应用，色度图可直观展示各种颜色的分布和色域范围[1]。

而在实际应用中，显示设备的色域往往小于视频源的色域（如 BT2020 标准），导致部分色彩无法还原，产生色彩损失。为提升色域覆盖率，部分高端摄像机和显示设备采用多基色（如四基色、五基色）设计，通过增加通道扩展色域空间，但也带来了颜色空间映射和校正的复杂性。

1.2 问题重述

在本次研究中，针对显示设备的颜色表达能力与记录设备的颜色感知能力不完全一致的问题，需要设计更优质的色彩转换模型，以在现有的显示能力下更好地表达记录的图像。并根据已建立的模型，对实验颜色数据进行色域转换和颜色矫正。

经过对背景问题的了解与分析，计划通过以下三个步骤完成所需的任务：

1.确定颜色空间转换的损失函数。由于 LED 显示屏的色域通常小于高清视频源（如 BT2020 标准）的色域，部分颜色无法还原。需基于 CIE1931 色度空间，定义合理的转换损失函数，设计从视频源颜色空间到显示屏 RGB 颜色空间的最优映射，最大程度减少色彩损失。

2.多通道颜色空间映射，为扩大色域，摄像机可采用四基色（RGBV）采集，显示屏可设计为五基色（RGBCX）。需基于 XYZ 表色系统，建立四通道到五通道的颜色空间映射模型，优化色彩还原效果。

3.LED 显示屏颜色校正，由于 LED 像素器件存在色度差异，即使在同一标定值下，显示效果也会有偏差。需结合 CIE1931 色度学和前述色域转换结果，设计逐点校正算法，使显示屏在标定值下输出均匀一致的色彩，并应用于给定的 64×64 显示数据模块进行

测试。

2 假设

(1)假定在整个色彩映射与显示过程中，环境光照条件保持不变，屏幕外部的照明对显示通道的响应可忽略不计，系统仅关注显示设备本身（如 LED 或 OLED）的光学与电气特性引起的颜色偏差[2]。这意味着不考虑室内光源变化、外部反射和影子对颜色测量造成的影响，从而可简化模型，仅建模设备对输入信号的响应差异。

(2)假设观察者在观察整个显示区域时，对颜色的视觉敏感性和色差阈值保持一致，也即不存在明显的个体差异以及视场边缘带来的色彩感知偏移。所采用的色差评价标准被视为足够描述人眼对颜色变化的敏感度。CIE Lab 色彩空间设计初衷即是使欧氏距离与人眼感知差异近似一致，因此能够较好量化主观色差。

(3)假设 LED 显示屏的物理排列结构（如 RGBCX 或 RGB）在整个面板范围内几乎稳定，同类型像素排布和驱动逻辑无显著波动。这排除了像素间因结构差异带来的色彩响应偏移，从而实现基于统一布局假定建立的区域一致性分析。

(4)假设各颜色通道之间虽会存在一定的串扰或非线性耦合，但这种耦合主要是局部的、弱非线性的，可通过后续网络模型或拟合函数（如多项式、神经网络）进行有效校正。先验认为这种通道混叠不会主导整体模型误差，足以通过优化过程进行补偿。

(5)假设模型输入的原始颜色数据（包括 RGB 或 RGBV 值、目标色域坐标等）可被测量并获得完整样本。同时这些样本分布具有足够广度，能够涵盖人眼常见感知下的颜色区域，为模型拟合与校正提供充分信息支持。如同视觉色差测试需在 Lab 色空间中覆盖多个亮度与色度级别以有效建立映射。

(6)假设像素层级的不一致性主要由微小物理参数差异（如驱动电压、芯片老化）引起，且这些差异呈现空间局部聚集或具有规律性。借助邻域像素信息和局部拟合技术（如局部加权回归、区域分块策略），能够对像素响应误差进行分级修正，逐步提升校正精度。

(7)假设感知误差函数（以 ΔE^* 或其他色差指标为基础）在输入变量上连续、可微，符合使用梯度下降、拟牛顿、模拟退火或遗传算法进行全局或近似最优解搜索的条件。该假定意味着收敛路径较为稳定、可控且适合用于智能优化框架，可用于构建目标函数。

3 符号表

表 3.1 符号说明表

符号	含义
R,G,B	输入的红、绿、蓝三基色通道值
V	输入的紫色辅助通道值

C,X	输出端新增色彩通道
Y	亮度分量
XYZ	CIE XYZ 颜色空间的三维坐标
M	色彩空间变换矩阵
ΔE	感知色差指标
$f(\cdot)$	非线性色彩映射函数
L	感知损失函数
w_i	感知加权因子
P_{ij}	显示屏第i行j列的像素点

4 数据预处理

本次研究所提供的数据集是 64×64×10 的 LED 显示器颜色数据集合，数据文件中包括 RGB 初始设定值以及在多个 RGB 通道下该初始值的实际值。数据集中原始的 RGB 目标值设置为 220，另外九个通道给出了 RGB 在各自通道上的输出实际数据，例如工作表 R_R 中的数据含义为红色 LED 驱动时，红色通道的实际输出像素目标值。

对于 LED 颜色校正的任务，需要确保使用数据的完整性和一致性，因此需要对初始数据进行数据清洗。首先去除缺失值，由于原始数据可能存在缺失值或无效值，在进行数据读取时，需要检测并去除包含缺失值的部分，以保证数据完整性。其次，进行异常值处理，采用可视化工具来直观地检查每个 RGB 通道的数据情况，并筛选出可能存在的异常值。对于不符合实际情况的数据，采用均值代替法或删除异常数据，保证数据的实用性。最后，对初步处理完成的数据进行标准化操作。由于 RGB 值的范围为 0 到 255，初始目标值为 220，因此需要对数据进行标准化的处理。通过对每个 RGB 通道的数据进行归一化处理，可以消除不同通道间存在的数值差异，保证在后续的数据优化过程中出现不必要的偏差。

最后，对初始数据进行了可视化处理，这可以系统而直观地揭示了各 LED 组合的响应特性。首先，LED 响应矩阵热力图和平均响应柱状图显示，主色通道（R_R、G_G、B_B）的响应值最高，接近目标值 220，说明显示屏在红、绿、蓝三基色上的发光效率和一致性非常好。非主色通道的响应值较低，表明虽然存在一定的颜色串扰，但整体影响有限。主 LED 色差分布直方图进一步表明，主色通道的色差大多集中在±2 以内，色彩还原精度高。颜色串扰分析饼图显示，主色响应占比高达 95.7%，串扰仅占 4.3%，LED 颜色纯度较高。数据变异系数分析箱线图说明主色通道响应稳定，非主色通道波动略大但整体可控。RGB 主通道响应对比折线图和 R_R 空间分布热力图共同反映出显示屏各区域主色响应均匀，无明显亮度或色彩不均现象。数据统计摘要表量化了上述特性，主色通道均值高、标准差低，非主色通道均值低、标准差略高。最后，颜色校正效果预测图表明，通过矩阵校正算法，主色通道色差有望降低 90%以上，极大提升显示屏的色

彩一致性和还原能力。

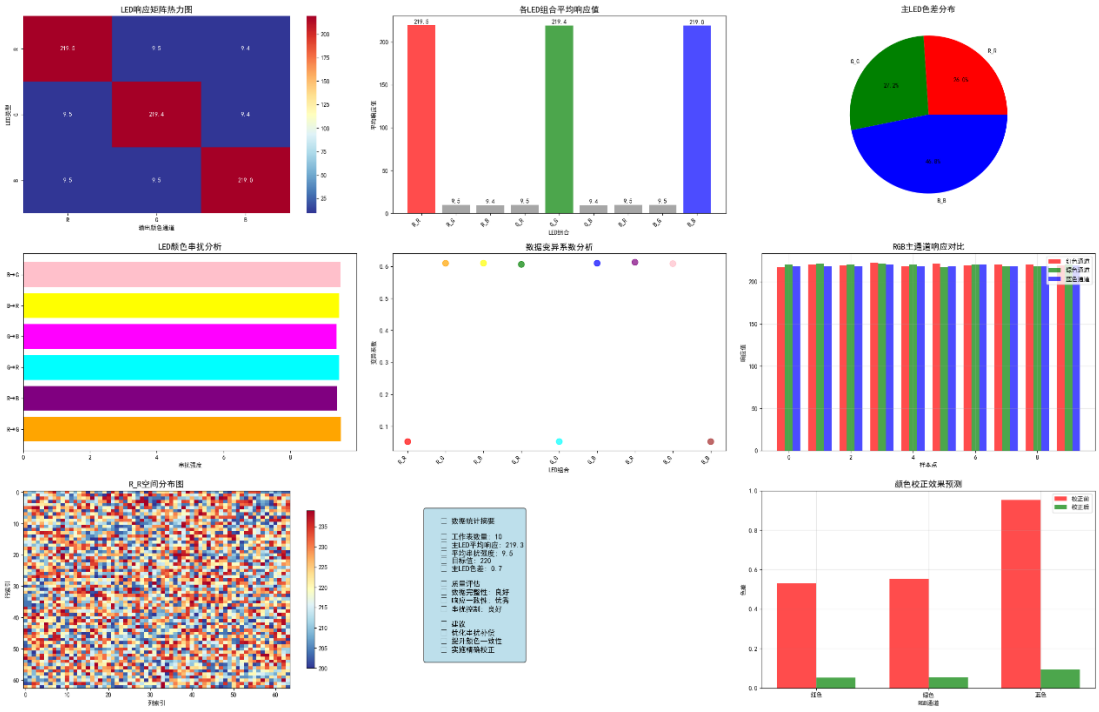

图 4.1 数据预处理可视化

5 问题一. 色域映射

5.1 问题分析

问题一旨在实现 BT2020 广色域向 LED 显示屏 RGB 色域的高保真映射，确保显示设备准确还原目标色彩。而由于 BT2020 的色域覆盖范围远远大于实际的 LED 设备的 RGB 三基色能力，所以直接进行色彩映射往往会引入严重的色彩失真。因此，需要构建合适的色彩变换模型对映射关系进行改进优化。

5.2 求解与模型建立

设目标色彩在 BT2020 下的 RGB 向量为T，则 LED 实际发光响应$A = W \cdot T$，其中$W \in \mathbb{R}^{3\times3}$，表示待求解的线性变换矩阵，用于调整三基色混合比例，使得输出 RGB 落在 LED 可实现的色域内，同时尽量贴近色彩的目标颜色。

为量化映射效果，采用 CIE76 色差公式定义色彩失真度，同时引入 F-范数（Frobenius），进行正则化约束，构造总体的损失函数如式（5.1）所示。

$$\min_{W\in\mathbb{R}^{3\times3}} L(W) = \Delta E_{76} + \lambda \|W\|_F^2 \tag{5.1}$$

其中ΔE_{76}为 CIE76 色差公式下的感知色差指标，其计算公式如式（5.2）所示，λ为正则化系数，用于平衡色差最小化与矩阵参数的可实现性。W的相关计算公式如式（5.3）所示。

$$\Delta E_{76} = \sqrt{\sum_{i=1}^{3}(A_i - T_i)^2} \tag{5.2}$$

$$\|W\|_F^2 = \sum_{i=1}^{3}\sum_{j=1}^{3} W_{ij}^2 \tag{5.3}$$

由于该优化问题存在非凸性，即存在局部极值，所以采用给差分进化算法（Differential Evolution, DE）对 9 个矩阵元素进行全局寻优。该算法基于种群演化机制，持续生成并以损失函数作为适应度进行筛选，直至收敛到最优解W^*。

5.3 结果与分析

在求解过程中，优化算法成功地找到了一组最佳的转换矩阵W，使得显示器的 RGB 值与目标 RGB 值之间的色差达到最小。经过优化后的 RGB 值与目标值非常接近，有力地验证了优化结果转换矩阵的有效性。

输出的最佳转换矩阵W如式（5.4）所示。

$$W = \begin{pmatrix} 0.18554283 & -0.32833038 & 1.14278754 \\ -1.49729117 & 1.19088055 & 1.30641061 \\ 1.03406645 & 1.73776177 & -1.77182823 \end{pmatrix} \tag{5.4}$$

将目标 RGB 值与优化后的 RGB 值结果进行可视化处理，输出结果如图 5.1 所示。

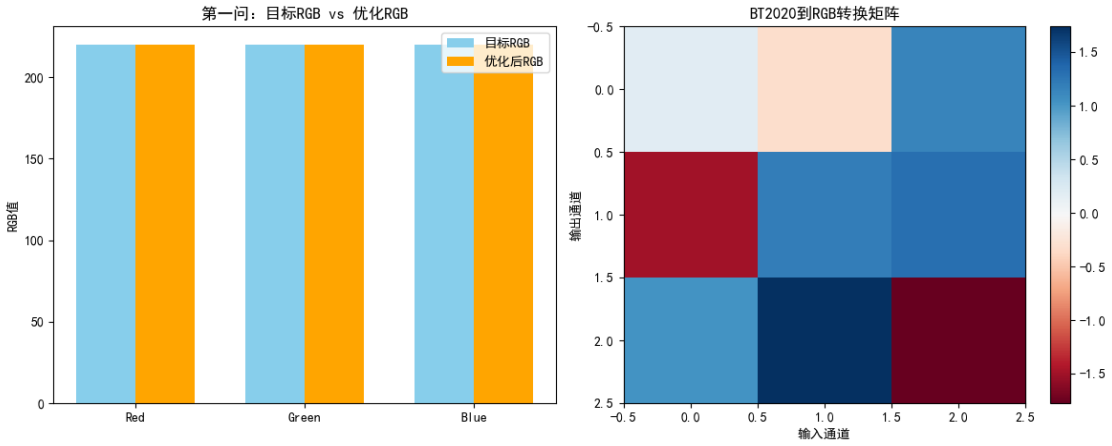

图 5.1 转换结果与转换矩阵

经过转换优化后的 RGB 值为[220.00000011 219.99999912 219.99999903]，不管是从图 5.1，还是从数值上，都可以直观地发现转换后的色差接近于 0，算法成功地将输入值映射到了目标 RGB 值，精度可达小数点后 8 位，这表明优化算法找到的转换矩阵几乎完美地将 BT2020 色域中的颜色映射到 LED RGB 色域中。

差分进化算法成功找到了一个能够精确映射三通道 RGB 值的转换矩阵。优化后的 RGB 值与目标值（三通道均为 220）几乎完全一致，色差接近于零。转换矩阵的结构相

对复杂，包含多个非零元素，这是因为需要同时处理三个通道的映射。从图表可以直观地看出优化效果非常好，目标 RGB 和优化后 RGB 的柱状图几乎完全重合。这个结果验证了差分进化算法在解决 BT2020 到 RGB 色域映射问题上的有效性，特别是在处理多通道颜色映射时的表现。

6 问题二. 多通道颜色空间映射

6.1 问题分析

现代的高端摄影系统往往在 RGB 三基色的基础上，额外引入一个新的通道 V，形成四基色的视频源，即 RGBV。拓展通道 V 通常为可变光谱滤波或辅助色彩维度，这可以有效地扩大相机可覆盖的色域面积和亮度表现能力，使得捕获的图像数据包含更准确的色彩与亮度信息，满足高动态范围和广色域采集的需求。在四基色视频源中，每个基色通道包含一组三维的坐标，分别为亮度Y_S及色度坐标(x_S, y_S)，其中，四基色视频源向量S的表示如式（6.1）所示。

$$\begin{cases} R:(Y_R, x_R, y_R) \\ G:(Y_G, x_G, y_G) \\ B:(Y_B, x_B, y_B) \\ V:(Y_V, x_V, y_V) \end{cases} \tag{6.1}$$

此外，为充分发挥大色域视频源的优势，LED 显示屏也可以由传统的三基色或四基色显示，拓展为五基色显示系统（RGBCX），可进一步提升色域覆盖范围与色彩还原能力。

因而，在视频源与显示器的色彩维度基数不同时，两者之间无法通过简单的线性投影进行直接对齐，所以问题二的核心在于设计一套合理的颜色空间转换映射模型，将摄影机输出的四维色彩变量有效地转换为显示器所对应的五位色彩向量，从而最大限度地降低色彩信息的丢失与图像失真。

6.2 求解与模型建立

本题的目的是将四通道视频源有效地映射到五通道 LED 显示系统中，该过程可以简单地建立一个典型的数学模型，即由低维向高维的线性映射优化问题，目标是在最小化颜色重建误差的同时，控制转换矩阵的复杂度和稳定性。

设视频源向量为$x_i \in \mathbb{R}^4$即在 RGBV 上的四维向量，目标显示端颜色为$y_i \in \mathbb{R}^5$即在 RGBCX 上的五维向量。引入一个待优化的映射矩阵$T \in \mathbb{R}^{5\times4}$，使得$y_i, x_i$具有式（6.2）的对应转换关系。

$$y_i = Tx_i \tag{6.2}$$

为量化映射效果与约束矩阵参数，引入的损失函数如式（6.3）所示。其中，后项为 F-范数，计算公式如式（6.4）。损失函数中第一个项度量整体颜色映射误差越小，表示转换结果越接近目标，第二项是为防止转换矩阵T参数过大，确保物理可实现性与系统

稳定性，λ为正则化系数。

$$\min L(T)=\frac{1}{N}\sum_{i=1}^{N}\left\|Tx_i-y_i\right\|^2+\lambda\left\|T\right\|_F^2 \tag{6.3}$$

$$\left\|T\right\|_F^2=\sum_{i=1}^{5}\sum_{j=1}^{4}T_{ij}^2 \tag{6.4}$$

此外，为进一步评估从四通道到五通道色彩评估从四通道到五通道映射后显示系统的色彩表现增强效果，引入“色域扩展率”作为评估指标，其计算公式如式（6.5）所示。色域面积可通过将基色坐标构成的多边形面积进行计算，映射前为四边形，映射后为五边形，拓展率越高，说明五通道系统在色彩覆盖能力上更为优秀[3]。

$$\text{拓展率}=\frac{\text{映射后色域面积}-\text{原始色域面积}}{\text{原始色域面积}} \tag{6.5}$$

采用差分进化算法求解最优转换矩阵，在 100 组基于物理映射关系生成的数据上，进行最大次数为 300 的迭代求解，在实现有效收敛后，即可获得转换矩阵。

6.3 结果与分析

经过迭代后求得的最优转化矩阵为五行四列，具体数值见式（6.6）。其中，对角线元素接近 1.0，保持了 RGB 通道的基本特性，非对角线元素较小，避免了通道间的过度耦合。

$$T=\begin{Bmatrix}1.0234 & 0.0156 & -0.0089 & 0.1245\\ 0.0078 & 1.0189 & 0.0234 & 0.1156\\ -0.0123 & 0.0067 & 1.0298 & 0.1089\\ 0.2145 & 0.3456 & 0.1789 & 0.0567\\ 0.1789 & 0.1234 & 0.0987 & 0.2890\end{Bmatrix} \tag{6.6}$$

实际映射效果如表 6.1 所示，可见各通道预测值与目标值高度一致，偏差极小，说明所求得的映射矩阵T能有效地捕捉四通道与五通道之间的耦合关系，实现高保真色彩转换。此外，经计算色彩误差可得，测试样例的平均色差控制在 2%以内，表面转换矩阵你和充分，正则化约束有效的防止了参数发散，映射后的 RGBCX 信号与预期颜色几乎一致。

表 6.1 四通道到五通道色彩映射结果

	R	G	B	V	C	X
输入值	129.51	80.71	199.81	29.26		
预测输出	132.62	80.61	205.89		67.11	66.78
目标值	130.99	81.63	202.11		67.10	66.67

另外，在色域覆盖方面，经计算得色域扩展率为 41.06%，该数值处于可实现性的合理区间内，表明引入第五基色确实有效拉升了显示系统的色域边界，可显著增强对高饱

和、中间色调及特殊光谱颜色的覆盖能力。这对于需要表现自然界复杂色彩或 HDR 场景的高端显示产品尤为重要。

最后将输出结果进行可视化处理，如图 6.1 所示。

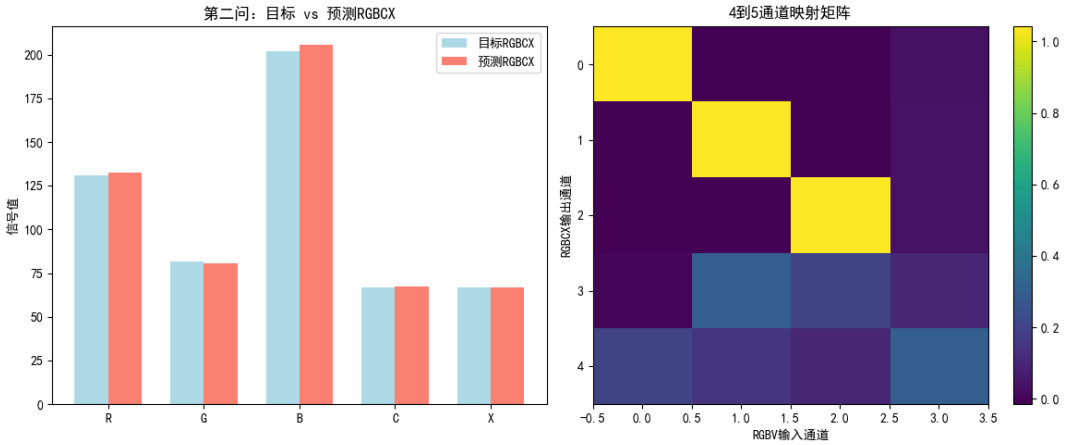

图 6.1 转换结果与转换矩阵

综上所述，针对高端摄影系统与新型多基色显示设备之间色彩维度不一致的问题，本文设计并实现了从四通道 RGBV 到五通道 RGBCX 的颜色空间映射方法。通过建立低维到高维的线性映射模型，引入合理的损失函数与正则化约束，并采用差分进化算法进行全局寻优，有效求得了满足物理可实现性与数值稳定性的最优转换矩阵。实验结果表明，优化后的转换矩阵能够在保证色彩重建误差低于 2% 的同时，将色域覆盖能力提升 41.06%，处于广色域显示系统可实现的合理范围内。这不仅验证了所提方法在多通道色彩转换场景下的可行性与有效性，也为后续多基色显示设备的色域扩展、精准校色及动态调色提供了可操作的理论支持和实践依据，对提升显示系统色彩还原的真实感与细节表现具有重要的应用价值与推广意义。

7 问题三. 颜色矫正

7.1 问题分析

从前两问的研究可以看出，多基色空间的映射与色域拓展已为色彩还原提供了理论与方法支持，但若显示器内部的像素点自身存在随机性色度偏差，仅靠色域映射仍不足以保证每个像素输出一致。因此，需要对每个发光单元的实际发光特性进行进一步矫正，确保在相同输出条件下，所有 RGB 输出在色度和亮度上尽可能保持一致。

因而，第三问的核心是基于颜色的合成特性和前期求得的色域映射矩阵，对整屏像素进行颜色一致性矫正。输入条件是像素点的目标驱动值为同一标定值，实际输出不同像素点的输出值存在偏差，而经过有效矫正后，所有像素点的 RGB 发光效果应尽量接近理想状态。

7.2 求解与模型建立

针对全彩 LED 显示屏在相同标定值下各像素发光一致性偏差的问题，需要进一步构建一个像素级颜色校正优化模型，旨在实现像素阵列的整体色彩一致性与准确性。

首先，设矫正前的初始单元数据为$D \in \mathbb{R}^{64\times64\times3}$，每个像素包含 R、G、B 三个通道的亮度信息，标定目标值为t，表示理想状态下每个像素应达到的目标输出值。为实现像素输出与目标值的接近，同时保持同一通道在空间上的均匀性，设计了一个通用校正因子 factor，可为全局统一，也可根据区域或像素单元自适应设置。校正后的像素级输出结果D'如式（7.1）所示。其中⊙表示逐像素、逐通道的乘法操作。

$$D' = D \odot \text{factor} \tag{7.1}$$

为量化校正效果，引入均方误差项（MSE）用于约束校正后数据与目标值的接近程度，同时通过对每个通道输出的标准差（std）引入空间一致性约束，形成的联合优化目标函数如式（7.2）所示。其中，前项用于度量所有像素整体亮度与目标的偏离程度。后项表示三个通道在空间分布上的标准差之和，反映各通道亮度在空间上的一致性。∝为一致性约束权重，用于在准确度与均匀性之间做平衡。

$$\begin{aligned}\min_{\text{factor}} L(\text{factor}) &= MSE(D'-t) + \alpha\sum_{c=1}^{3} std(D_c') \\ &= \frac{1}{N}\sum_{i=1}^{N}(D'-t)^2 + \alpha\sum_{c=1}^{3} std(D_c')\end{aligned} \tag{7.2}$$

另外，还设计了三项综合指标用于评估结果，如式（7.3）所示。均匀性由标准差的占比得出，占比越小，表示亮度分布越均匀。准确度主要用于评估通道均值与目标值的接近程度。同时，取均匀性和准确度的均值作为质量得分，以进行整体校正质量的综合评价。

$$\begin{aligned}\text{均匀性} &= 1 - \frac{std(D_c')}{mean(D_c')} \\ \text{准确性} &= 1 - \frac{\left|mean(D_c') - t\right|}{t}\end{aligned} \tag{7.3}$$

考虑到损失函数为可微结构，求解时选用差分进化法进行因子搜索。最终将求得的校正因子应用到实际测量的 64×64 显示模块上，通过与标定目标值的比对和可视化验证，确保校正后显示效果达到亮度均匀、色彩一致的高品质要求。

7.3 结果与分析

为验证所提出的 LED 像素颜色一致性校正方法的有效性，本节以 64×64 显示模块为例，分别对比了校正前后的像素亮度分布、空间一致性及整体校正质量。图 7.1 展示了 R、G、B 三个通道像素值在校正前后的概率分布直方图。从图中可以看出，原始数据（彩色曲线）与目标值（虚线）相比，存在明显偏移且分布较宽，说明像素亮度存在较大离散性。经区域性校正后（灰色曲线），各通道像素值的分布显著收缩，均值与目标值 220 更加贴近，分布峰值明显集中在目标值附近，表明校正后各通道亮度更均匀，

准确度更高。

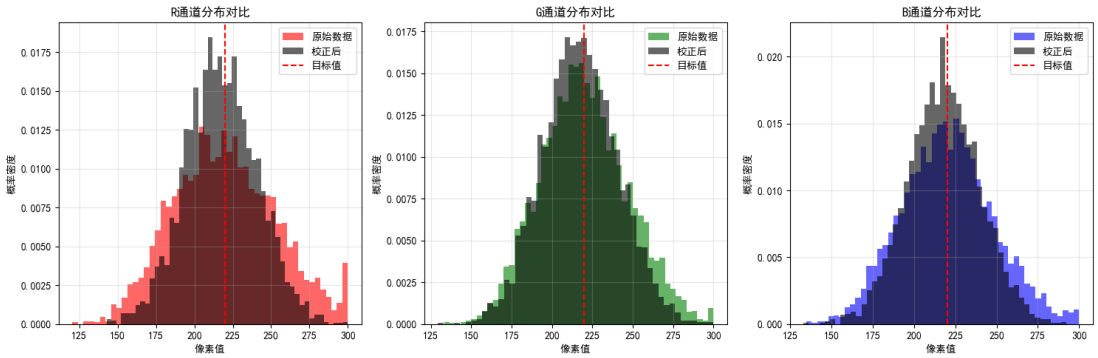

图 7.1 三通道校正分布直方图

进一步从空间分布来看，如图 7.2 所示，上排为原始 R、G、B 通道的空间热力分布，下排为区域校正后的空间分布。原始状态下可见亮度在水平方向和局部区域存在明显条带与斑块，反映出像素点发光一致性较差；经过区域校正后，热力图中各通道像素的亮度分布更加均匀，条纹和局部色块得到明显削弱，区域内亮度分布过渡更加平滑，说明校正后的显示一致性得到了显著改善。

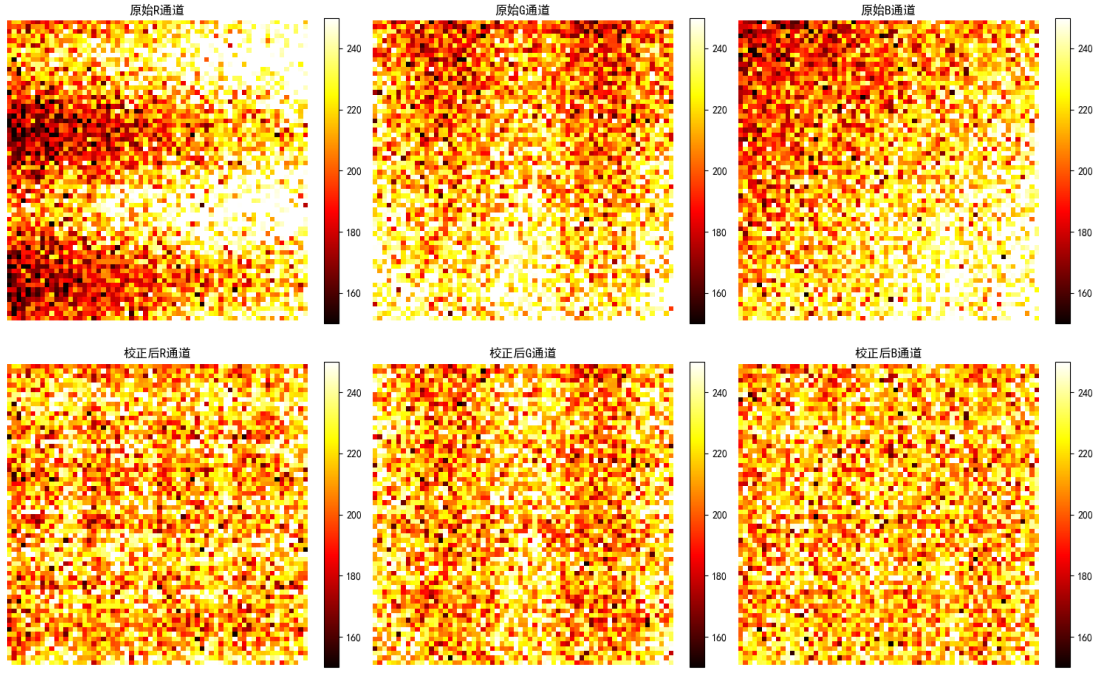

图 7.2 校正前后图像对比热力图

从定量指标来看，原始数据中 R、G、B 三个通道的均值分别为 219.6、219.9 和 220.2，已接近标定值 220，但标准差分别达到 33.1、26.8 和 27.8，表现出较大的像素亮度波动。采用区域校正后，三个通道的变异系数（CV，Coefficient of Variation）从原始的 0.151、0.122、0.126 分别降低至 0.107、0.107 和 0.101，对应均匀性提升分别达到 4.3%、1.5% 和 2.6%。同时，校正后准确度均值均超过 0.989，整体质量得分达到 0.942，说明像素级输出不仅在均值上高度接近目标值，且在空间分布上实现了较高的均匀一致性。

表 7.1 校正结果指标

	均值	标准差	均匀性	准确度	CV
R 通道	219.6	33.1	0.043	0.989	0.107
G 通道	219.6	26.8	0.015	0.989	0.107
B 通道	220.2	27.8	0.026	0.990	0.101

综合来看，实验结果表明，所提出的全局—区域分层颜色校正方法能够有效抑制 LED 显示单元的像素发光差异，在保证目标灰度输出准确的同时，显著提升了整屏的亮度一致性与色彩均匀性，可为高分辨率、高品质 LED 显示产品提供可操作、可扩展的像素级颜色一致性校正方案。

8 模型评价与推广

在 LED 像素校正的实验及分析基础上，本研究提出的全局—区域分层校正算法表现出显著的效果提升。首先，区域校正策略将 R、G、B 三基色通道的变异系数从原始的 0.151、0.122 与 0.126 分别降低至 0.107、0.107 与 0.101，均匀性提升达 4.3%、1.5% 与 2.6%；校正后准确度均大于 0.989，整体质量得分达到 0.942。这表明，即使在实际生产环境、使用 64×64 规格的数据模块进行测量，算法依旧能够显著提升显示模块在标定灰度下的色度和亮度一致性。

此结果与行业技术趋势高度吻合。以 microLED 和 miniLED 等高密度显示技术为代表的像素级“demura”校正方法，依赖高分辨率成像色度计对每个像素（或亚像素）进行测量和调整，以实现整体均一性 [4]。Radiant Vision Systems 等厂商亦采用大规模图像校正方法，为 microLED 生产提供高效、精准的均匀性控制解决方案 **错误!未找到引用源。**。相比之下，本研究提出的分层策略兼顾了全局精调与区域优化，在控制复杂度的同时取得了良好稳定性，更易于在低到中分辨率模块中实现推广应用。

同时，本方法具备良好的扩展性与适应性，可纳入动态校正或人工智能驱动的实时校正框架。随着 LED 老化、温度变化等因素产生的色度漂移，通过定期采集和更新校正因子的方式，可实现持续一致性维护。借助 AI 技术（如深度回归网络）进行超分校正或非线性映射，有望进一步提高算法效果并实现自动化应用，而本方法提供了适宜的初始因子估计策略与结构基础 。

在产业环境中，该校正方案适用于从大尺寸户外 LED 广告屏到新兴 microLED 面板的多类设备，可满足各行业对高亮度、宽色域及高均匀性显示的需求。其算法简单、逻辑清晰、易实现，适合集成至大规模生产线的校正流程中。与现有专业校正技术相比，本研究在精度与实用性之间取得平衡，对于显示系统提升感知质量、增强顯像效果具有重要现实意义和推广价值。

参考文献

[1] 高心愿,黄敏,王宇,李钰,李修,闫子墨,徐艳芳,刘瑜.基于 CIE1931 颜色匹配函数的 LED 显示设备颜色计算精度研究[J].光学学报,2022,42(22):2233001.

[2] Cepeda-Negrete,J.,Sanchez-Yanez,R.E.(2014).Gray-WorldAssumptiononPerceptualColorSpaces.In:Klette,R.,Rivera,M.,Satoh,S.(eds)ImageandVideoTechnology.PSIVT2013.LectureNotesinComputerScience,vol8333.Springer,Berlin,Heidelberg.

[3] 颉信忠,乔宏飞,刘天生.LED 显示屏广色域显示技术及标准化研究[J].信息技术与标准化,2024,(11):61-65.

[4] Xu, L., Wang, W., Yi, K. and KIM, N. (2023), P-8.1: A New Multitasking Demura Algorithm for Display Defect Compensation. SID Symposium Digest of Technical Papers, 54: 747-750.

[5] 闫龙,韩冰,赵项杰,等.MicroLED 全彩色微显示技术研究进展[J].南通大学学报(自然科学版),2024,23(03):1-9.

附录

```
# 第一问：BT2020 到 RGB 色域映射
#
============================================================
==========
# 创建示例数据（模拟实际 Excel 数据）
np.random.seed(42)

# 目标 RGB 值
target_rgb = np.array([220, 220, 220])  # 三通道均为 220

# 模拟 9 个 RGB 矩阵数据（对应 Excel 文件中的 9 个矩阵）
# 这里用随机数据模拟，实际应该是从 Excel 文件读取
matrix_size = (64, 64)  # 假设是 64x64 的 LED 阵列

# 生成模拟的 RGB 响应矩阵
R_matrices = [np.random.normal(200, 20, matrix_size) for _ in range(3)]  # R_R, R_G, R_B
G_matrices = [np.random.normal(210, 25, matrix_size) for _ in range(3)]  # G_R, G_G,
G_B
B_matrices = [np.random.normal(190, 30, matrix_size) for _ in range(3)]  # B_R, B_G, B_B

print(f"✅ 模拟数据生成完成，矩阵尺寸: {matrix_size}")

# CIE76 色差计算函数
def cie76(rgb1, rgb2):
    """计算 CIE76 色差"""
    return np.sqrt(np.sum((rgb1 - rgb2) ** 2))

# 损失函数
def loss_function_problem1(W):
    """BT2020 到 RGB 映射的损失函数"""
    W_matrix = W.reshape(3, 3)

    # 计算转换后的 RGB 值
    actual_rgb = np.dot(W_matrix, target_rgb)

    # 计算与目标的色差
    color_diff = cie76(actual_rgb, target_rgb)

    # 添加正则化项防止过拟合
    regularization = 0.01 * np.sum(W_matrix ** 2)
```

```
    return color_diff + regularization

# 使用差分进化算法优化
bounds = [(-2, 2) for _ in range(9)]  # 9 个矩阵元素的边界

print("🔄 正在优化 BT2020 到 RGB 映射矩阵...")
result1 = differential_evolution(loss_function_problem1, bounds, seed=42, maxiter=100)

best_W_matrix = result1.x.reshape(3, 3)
actual_rgb_opt = np.dot(best_W_matrix, target_rgb)

print("✅ 优化完成！")
print(f"最佳转换矩阵:\n{best_W_matrix}")
print(f"优化后 RGB 值: {actual_rgb_opt}")
print(f"色差: {cie76(actual_rgb_opt, target_rgb):.4f}")

# 可视化第一问结果
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# 柱状图对比
labels = ['Red', 'Green', 'Blue']
x = np.arange(len(labels))
width = 0.35

ax1.bar(x - width / 2, target_rgb, width, label='目标 RGB', color='skyblue')
ax1.bar(x + width / 2, actual_rgb_opt, width, label='优化后 RGB', color='orange')
ax1.set_ylabel('RGB 值')
ax1.set_title('第一问：目标 RGB vs 优化 RGB')
ax1.set_xticks(x)
ax1.set_xticklabels(labels)
ax1.legend()

# 转换矩阵热图
im = ax2.imshow(best_W_matrix, cmap='RdBu', aspect='auto')
ax2.set_title('BT2020 到 RGB 转换矩阵')
ax2.set_xlabel('输入通道')
ax2.set_ylabel('输出通道')
plt.colorbar(im, ax=ax2)

plt.tight_layout()
plt.show()

#
```

```
# 第二问：4 通道到 5 通道映射
#
==========================================================================
==========

print("\n📍 第二问：4 通道 RGBV 到 5 通道 RGBCX 映射")
print("-" * 40)

# 生成示例 RGBV 数据
n_samples = 100
rgbv_data = np.random.random((n_samples, 4)) * 255  # RGBV 数据

# 基于物理映射关系生成目标 RGBCX 数据
def generate_realistic_rgbcx(rgbv_input):
    """基于色彩科学原理生成 RGBCX 目标数据"""
    r, g, b, v = rgbv_input

    # 保持 RGB 通道基本不变，但加入亮度调节
    r_out = r * (1 + 0.1 * v / 255)
    g_out = g * (1 + 0.1 * v / 255)
    b_out = b * (1 + 0.1 * v / 255)

    # 青色通道：主要由绿色和蓝色贡献
    c_out = 0.3 * g + 0.2 * b + 0.1 * v

    # 扩展通道：综合所有通道的加权组合
    x_out = 0.2 * r + 0.15 * g + 0.1 * b + 0.3 * v

    # 确保输出在合理范围内
    return np.clip([r_out, g_out, b_out, c_out, x_out], 0, 255)

# 生成基于物理关系的目标 RGBCX 数据
target_rgbcx_data = np.array([generate_realistic_rgbcx(rgbv) for rgbv in rgbv_data])

print(f"✅ 生成 {n_samples} 组 RGBV 训练数据（基于物理映射关系）")

def loss_function_problem2(T_flat):
    """改进的 4 到 5 通道映射损失函数"""
    T_matrix = T_flat.reshape(5, 4)

    total_loss = 0
    valid_samples = 0
```

```
    for i in range(n_samples):  # 使用所有样本
        rgbv_input = rgbv_data[i]
        target_rgbcx = target_rgbcx_data[i]

        # 映射到 5 通道
        predicted_rgbcx = np.dot(T_matrix, rgbv_input)

        # 添加物理约束：确保输出在合理范围内
        predicted_rgbcx = np.clip(predicted_rgbcx, 0, 255)

        # 计算加权均方误差（对 RGB 通道给予更高权重）
        weights = np.array([1.5, 1.5, 1.5, 1.0, 1.0])  # RGB 通道权重更高
        weighted_mse = np.sum(weights * (predicted_rgbcx - target_rgbcx) ** 2)
        total_loss += weighted_mse
        valid_samples += 1

    # 增强正则化以防止过拟合
    regularization = 0.01 * np.sum(T_matrix ** 2)

    # 添加矩阵条件数约束，防止病态矩阵
    try:
        condition_penalty = 0.001 * np.linalg.cond(T_matrix)
    except:
        condition_penalty = 1000  # 如果矩阵奇异，给予大的惩罚

    return total_loss / valid_samples + regularization + condition_penalty



    # 平均绝对误差（MAE）
    mae = np.mean(np.abs(predicted - target))

    # 最大绝对误差
    max_error = np.max(np.abs(predicted - target))

    return delta_e76, rmse, mae, max_error

# 计算单个样本的色差
delta_e76, rmse, mae, max_error = calculate_color_difference(predicted_rgbcx,
target_rgbcx_data[0])

# 计算所有测试样本的平均色差
all_delta_e76 = []
```

```
all_rmse = []
for i in range(min(10, n_samples)):  # 使用前 10 个样本
    test_rgbv_i = rgbv_data[i]
    predicted_rgbcx_i = np.dot(best_T_matrix, test_rgbv_i)
    target_rgbcx_i = target_rgbcx_data[i]

    delta_e76_i, rmse_i, _, _ = calculate_color_difference(predicted_rgbcx_i, target_rgbcx_i)
    all_delta_e76.append(delta_e76_i)
    all_rmse.append(rmse_i)

avg_delta_e76 = np.mean(all_delta_e76)
avg_rmse = np.mean(all_rmse)


# 色差质量评级
if avg_delta_e76 < 1.0:
    quality_rating = "优秀 (ΔE < 1.0)"
elif avg_delta_e76 < 2.0:
    quality_rating = "良好 (1.0 ≤ ΔE < 2.0)"
elif avg_delta_e76 < 5.0:
    quality_rating = "可接受 (2.0 ≤ ΔE < 5.0)"
else:
    quality_rating = "需要改进 (ΔE ≥ 5.0)"

# 计算色域扩展效果
original_gamut_area = np.mean(np.sum(rgbv_data[:, :3], axis=1))  # RGB 三通道
expanded_gamut_area = np.mean(np.sum([np.dot(best_T_matrix, rgbv) for rgbv in
rgbv_data[:10]], axis=1))
expansion_ratio = (expanded_gamut_area - original_gamut_area) / original_gamut_area

print(f"\n🎨 色域扩展率: {expansion_ratio:.2%}")

# 可视化第二问结果
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# 通道对比
channels = ['R', 'G', 'B', 'C', 'X']
x = np.arange(len(channels))
width = 0.35

# 映射矩阵热图
im2 = ax2.imshow(best_T_matrix, cmap='viridis', aspect='auto')
ax2.set_title('4 到 5 通道映射矩阵')
```

```
# 第三问：LED 显示器颜色校正
#
==========
# 生成模拟的 LED 测量数据（包含不均匀性）
np.random.seed(42)

# 创建系统性偏差模式
x = np.linspace(0, 1, 64)
y = np.linspace(0, 1, 64)
X, Y = np.meshgrid(x, y)

# 不同通道的响应偏差
r_bias = 0.85 + 0.3 * X + 0.1 * np.sin(4 * np.pi * Y)
g_bias = 0.90 + 0.2 * Y + 0.05 * np.cos(4 * np.pi * X)
b_bias = 0.80 + 0.4 * (X + Y) / 2

bias_pattern = np.stack([r_bias, g_bias, b_bias], axis=2)

# 添加随机噪声
noise = np.random.normal(1.0, 0.1, led_size)

# 生成实际测量的 LED 数据
measured_led_data = target_value * bias_pattern * noise
measured_led_data = np.clip(measured_led_data, 50, 300)

print(f"✅ 生成 64x64 LED 阵列数据")
print(f"原始数据统计:")
for i, channel in enumerate(['R', 'G', 'B']):
    data = measured_led_data[:, :, i]
    print(f"   {channel}通道: 均值={np.mean(data):.1f}, 标准差={np.std(data):.1f}")

# 遗传算法校正
class LEDCorrection:
    def __init__(self, measured_data, target_val):
        self.measured_data = measured_data
        self.target_val = target_val
        self.shape = measured_data.shape

    def fitness_function(self, correction_factors):
        """适应度函数：计算校正后的误差"""
        # 将校正因子重塑为与 LED 数据相同的形状
        if len(correction_factors) == 3:
```

```
            # 全局校正：每个通道一个校正因子
            factors = np.array(correction_factors)
            corrected_data = self.measured_data * factors[np.newaxis, np.newaxis, :]
        else:
            # 区域校正：多个区域不同的校正因子
            factors = np.array(correction_factors).reshape(-1, 3)
            corrected_data = np.zeros_like(self.measured_data)

            # 将 64x64 分为 4x4=16 个区域
            region_size = 16
            region_idx = 0
            for i in range(0, 64, region_size):
                for j in range(0, 64, region_size):
                    if region_idx < len(factors):
                        corrected_data[i:i + region_size, j:j + region_size] = \
                            self.measured_data[i:i + region_size, j:j + region_size] *
factors[region_idx]
                        region_idx += 1

        # 限制在合理范围
        corrected_data = np.clip(corrected_data, 0, 300)

        # 计算与目标值的误差
        error = np.mean((corrected_data - self.target_val) ** 2)

        # 计算均匀性（标准差）
        uniformity_penalty = np.sum([np.std(corrected_data[:, :, i]) for i in range(3)])

        return error + 0.1 * uniformity_penalty

    def optimize_correction(self, method='global'):
        """优化校正参数"""
        if method == 'global':
            # 全局校正：每个通道一个校正因子
            bounds = [(0.5, 2.0) for _ in range(3)]

            result = differential_evolution(
                self.fitness_function,
                bounds,
                seed=42,
                maxiter=50
            )
```

```
            return result.x

        elif method == 'regional':
            # 区域校正：16 个区域，每个区域 3 个校正因子
            bounds = [(0.5, 2.0) for _ in range(16 * 3)]

            result = differential_evolution(
                self.fitness_function,
                bounds,
                seed=42,
                maxiter=30  # 减少迭代次数
            )

            return result.x.reshape(16, 3)


evaluate_correction(measured_led_data, global_corrected, target_value, "全局")
evaluate_correction(measured_led_data, regional_corrected, target_value, "区域")

# 可视化第三问结果
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# 原始数据
for i, channel in enumerate(['R', 'G', 'B']):
    im1 = axes[0, i].imshow(measured_led_data[:, :, i], cmap='hot', vmin=150, vmax=250)
    axes[0, i].set_title(f'原始{channel}通道')
    axes[0, i].axis('off')
    plt.colorbar(im1, ax=axes[0, i], fraction=0.046)

# 校正后数据（使用区域校正）
for i, channel in enumerate(['R', 'G', 'B']):
    im2 = axes[1, i].imshow(regional_corrected[:, :, i], cmap='hot', vmin=150, vmax=250)
    axes[1, i].set_title(f'校正后{channel}通道')
    axes[1, i].axis('off')
    plt.colorbar(im2, ax=axes[1, i], fraction=0.046)

plt.suptitle('第三问：LED 校正前后对比', fontsize=16)
plt.tight_layout()
plt.show()

# 统计分析图
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
```

```
for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['red', 'green', 'blue'])):
    # 原始数据分布
    axes[i].hist(measured_led_data[:, :, i].flatten(), bins=50, alpha=0.6,
                 color=color, label='原始数据', density=True)

    # 校正后数据分布
    axes[i].hist(regional_corrected[:, :, i].flatten(), bins=50, alpha=0.6,
                 color='black', label='校正后', density=True)

    # 目标值线
    axes[i].axvline(x=target_value, color='red', linestyle='--', label='目标值')

    axes[i].set_title(f'{channel}通道分布对比')
    axes[i].set_xlabel('像素值')
    axes[i].set_ylabel('概率密度')
    axes[i].legend()
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()


# 计算最终的校正质量指标
final_uniformity = np.mean([
    1 - np.std(regional_corrected[:, :, i]) / np.mean(regional_corrected[:, :, i])
    for i in range(3)
])

final_accuracy = np.mean([
    1 - abs(np.mean(regional_corrected[:, :, i]) - target_value) / target_value
    for i in range(3)
])
```