数据处理.py

```
import pandas as pd
import numpy as np
from scipy.signal import savgol_filter
import matplotlib.pyplot as plt

# --- Matplotlib 全局美化设置 ---
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 16
plt.rcParams['axes.titlesize'] = 18
plt.rcParams['legend.fontsize'] = 14


def preprocess_reflectance_data(fileName):
    """
    对反射率数据进行预处理，包括Savitzky-Golay滤波、多项式基线校正和3σ异常值处理。
    参数:
    fileName (str): 不含扩展名的Excel文件名。
    返回:
    处理后的数据文件路径和图像路径
    """
    file_path = fileName + ".xlsx"
    # 加载数据
    try:
        data = pd.read_excel(file_path)
    except FileNotFoundError:
        print(f"错误: 文件 '{file_path}' 未找到。")
        return
    # 提取关键列数据
    reflectance = data['反射率 (%)']
    wavelength = data['波长(μm)']  # 波长数据（x轴原始数据）
    # --- 1. Savitzky-Golay 滤波器 ---
    window_length = 39
    polyorder = 2
    reflectance_savgol = savgol_filter(reflectance, window_length, polyorder)
    # --- 2. 多项式基线校正 ---
    x = np.arange(len(reflectance_savgol))  # 临时索引（用于基线拟合）
    degree = 5
    coeffs = np.polyfit(x, reflectance_savgol, degree)
    baseline = np.polyval(coeffs, x)
    reflectance_baseline_corrected = reflectance_savgol - baseline
    # --- 3. 3σ 异常值处理 ---
    mean = np.mean(reflectance_baseline_corrected)
    std = np.std(reflectance_baseline_corrected)
    threshold = 3 * std
    outliers = np.abs(reflectance_baseline_corrected - mean) > threshold  # 异常值标记
    # 同步筛选波长和反射率数据（关键修正：保证x和y长度一致）
    reflectance_final = reflectance_baseline_corrected[~outliers]
    wavelength_final = wavelength[~outliers]  # 只保留非异常值对应的波长
    # 保存处理后的数据到新的DataFrame
```

```
    processed_data = data.copy()
    processed_data['Savitzky-Golay滤波后反射率'] = reflectance_savgol
    processed_data['基线校正后反射率'] = reflectance_baseline_corrected
    processed_data['最终反射率(已剔除异常值)'] = np.nan
    processed_data.loc[~outliers, '最终反射率(已剔除异常值)'] = reflectance_final
    new_file_path = f'{fileName}_processed.xlsx'
    processed_data.to_excel(new_file_path, index=False)
    # --- 可视化（修正绘图参数和数据匹配问题） ---
    plt.figure(figsize=(12, 10))
    # 1. 原始数据
    plt.subplot(4, 1, 1)
    plt.plot(wavelength, reflectance, label='原始数据')
    plt.title('原始反射率数据')
    plt.legend()
    # 2. Savitzky-Golay 滤波后数据
    plt.subplot(4, 1, 2)
    plt.plot(wavelength, reflectance_savgol, label='Savitzky-Golay 滤波后', color='orange')
    plt.title('经过 Savitzky-Golay 滤波')
    plt.legend()
    # 3. 基线校正后数据
    plt.subplot(4, 1, 3)
    plt.plot(wavelength, reflectance_baseline_corrected, label='基线校正后', color='green')
    plt.title('经过多项式基线校正')
    plt.legend()
    # 4. 剔除异常值后最终数据（关键修正：使用同步筛选后的波长和反射率）
    plt.subplot(4, 1, 4)
    plt.plot(wavelength_final, reflectance_final, label='最终数据 (已剔除异常值)', color='red')
    plt.title('经过 3σ 准则异常值处理')
    plt.legend()
    plt.tight_layout()
    img_path = f'{fileName}_data_preprocessing_steps.png'
    plt.savefig(img_path, dpi=300)  # 增加dpi确保图像清晰度
    plt.show()
    return new_file_path, img_path


if __name__ == '__main__':
    result = preprocess_reflectance_data('附件3')
    result = preprocess_reflectance_data('附件4')

```

Problem2_solution.py

```
import warnings
import pandas as pd
import numpy as np
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
from math import sin, sqrt, pi
from matplotlib.patches import Patch
import os
from datetime import datetime
from scipy import stats
```

```python

# --- 全局设置 ---
warnings.filterwarnings('ignore')
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 12  # 缩小字体，避免图表拥挤
color1 = "#00BA38"  # 极大值点
color2 = "#619CFF"  # 极小值点
color3 = "#C86193"  # 厚度分布
color4 = "#F8766D"  # 残差分析
grid_color = 'midgray'
text_color = 'black'
bg_color = 'lightgray'

# 确保输出目录存在
os.makedirs('output', exist_ok=True)
os.makedirs('output/images', exist_ok=True)
os.makedirs('output/excel', exist_ok=True)
os.makedirs('output/txt', exist_ok=True)
os.makedirs('output/stability', exist_ok=True)


# -------------------------- 1. 修正物理模型参数（碳化硅标准参数） --------------------------
def cauchy_refractive_index(lambda_μm, A=2.65, B=0.015, C=1e-7):
    """Cauchy模型：适配碳化硅红外波段（2.5-5μm）"""
    n = A + B / (lambda_μm ** 2) + C / (lambda_μm ** 4)
    return n


def sellmeier_refractive_index(lambda_μm, A1=6.91, B1=0.202):
    """Sellmeier模型：碳化硅标准参数（λ² > B1 μm²，B1=0.202对应λ>0.45μm，满足2.5-5μm范围）"""
    if lambda_μm ** 2 <= B1:
        raise ValueError(f"波长 {lambda_μm:.2f}μm 过小，需λ² > {B1}μm²")
    n_squared = 1 + (A1 * lambda_μm ** 2) / (lambda_μm ** 2 - B1)
    return sqrt(n_squared)


# -------------------------- 2. 数据预处理：适配实际波长范围 --------------------------
def preprocess_data(file_path, angle):
    try:
        data = pd.read_excel(file_path)
    except Exception:
        data = pd.read_csv(file_path)

    # 数据清洗与波长转换（波数cm⁻¹ → 波长nm：λ(nm)=1e7/波数(cm⁻¹)）
    data.columns = ["波数 (cm-1)", "反射率 (%)"]
    data["波数 (cm-1)"] = pd.to_numeric(data["波数 (cm-1)"], errors='coerce')
    data["反射率 (%)"] = pd.to_numeric(data["反射率 (%)"], errors='coerce')
    data = data.dropna()
    data["lambda_nm"] = 1e7 / data["波数 (cm-1)"]

    # 修正波长范围：附件数据实际为2500-5000nm（波数2000-4000cm⁻¹），避免无效数据
    valid_mask = (data["lambda_nm"] >= 2500) & (data["lambda_nm"] <= 5000)
```

```python
    data_valid = data[valid_mask].sort_values("lambda_nm", ascending=False).reset_index(drop=True)

    # 增加数据量判断（降低阈值，避免误判）
    if len(data_valid) < 50:
        raise ValueError(f"有效数据点仅{len(data_valid)}个，需≥50个")

    data_valid["incident_angle"] = angle
    print(
        f"预处理后数据：波长范围{data_valid['lambda_nm'].min():.0f}-{data_valid['lambda_nm'].max():.0f}nm，共{len(data_valid)}个点")
    return data_valid


# -------------------------- 3. 优化极值点检测（降低阈值，增加峰显著性判断） --------------------------
def detect_interference_extrema(data_valid):
    R = data_valid["反射率 (%)"].values
    lambda_nm = data_valid["lambda_nm"].values
    r_mean = np.mean(R)
    r_std = np.std(R)

    # 优化阈值：降低高度门槛，增加峰显著性（prominence）过滤噪声
    min_height_max = r_mean + 0.2 * r_std  # 极大值最小高度（原代码仅用min_height，此处分开设置更灵活）
    min_height_min = r_mean - 0.8 * r_std  # 极小值最小高度（降低门槛，避免漏检）
    min_distance = max(10, int(len(data_valid) * 0.02))  # 最小峰间距：至少10个点，避免过疏
    prominence = 0.05  # 峰显著性：过滤微小波动（反射率变化≥0.05%才视为峰）

    # 检测极大值（峰）和极小值（谷）
    peak_indices, _ = find_peaks(
        R,
        distance=min_distance,
        height=min_height_max,
        prominence=prominence
    )
    valley_indices, _ = find_peaks(
        -R,  # 极小值=负反射率的极大值
        distance=min_distance,
        height=-min_height_min,  # 对应原反射率的极小值高度
        prominence=prominence
    )

    # 整合极值点
    extrema = pd.concat([
        pd.DataFrame({"type": "max", "lambda_nm": lambda_nm[peak_indices], "反射率 (%)": R[peak_indices]}),
        pd.DataFrame({"type": "min", "lambda_nm": lambda_nm[valley_indices], "反射率 (%)": R[valley_indices]})
    ]).sort_values("lambda_nm", ascending=False).reset_index(drop=True)

    # 打印极值点数量，方便调试
    print(f"检测到极值点：共{len(extrema)}个（极大值{len(peak_indices)}个，极小值{len(valley_indices)}个）")
    if len(extrema) < 3:
        raise ValueError(f"极值点不足3个，需调整检测参数")

```

```
    return extrema


# --------------------------- 4. 优化厚度计算（增加异常值过滤） ---------------------------
def calculate_thickness(extrema, incident_angle, ref_model, **model_params):
    if len(extrema) < 2:
        return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()

    ref_index_func = cauchy_refractive_index if ref_model == "cauchy" else sellmeier_refractive_index
    angle_rad = incident_angle * pi / 180
    lambda0_nm = extrema.iloc[0]["lambda_nm"]
    k0_estimates = []

    # 计算k0（干涉级次基数）：过滤过小的波长差，避免异常值
    for i in range(1, len(extrema)):
        m_i = i * 0.5  # 干涉级次差（峰-谷/谷-峰差0.5级）
        lambda_i_nm = extrema.iloc[i]["lambda_nm"]
        lambda_diff = lambda0_nm - lambda_i_nm
        if lambda_diff > 10:  # 波长差≥10nm才计算，避免除以微小值
            k0_est = (m_i * lambda_i_nm) / lambda_diff
            k0_estimates.append(k0_est)

    if not k0_estimates:
        print("无有效k0估算值，需调整波长差阈值")
        return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()

    k0_final = np.median(k0_estimates)  # 用中位数抗异常值
    results = []
    residuals = []
    expected_k = []
    actual_k = []

    # 计算厚度并过滤异常值（厚度为正且在合理范围：0.1-100μm）
    for i, row in extrema.iterrows():
        m_i = i * 0.5
        k_i = k0_final + m_i
        lambda_i_nm = row["lambda_nm"]
        lambda_i_μm = lambda_i_nm / 1000.0

        # 计算折射率
        try:
            n_i = ref_index_func(lambda_i_μm, **model_params)
        except Exception as e:
            print(f"计算折射率失败：{e}，跳过该点")
            continue

        # 计算厚度（干涉公式：2ndcosθ = kλ → d = kλ/(2ncosθ)，cosθ=√(n²-sin²θ_incident)）
        denominator = 2 * sqrt(n_i ** 2 - sin(angle_rad) ** 2)
        thickness = (k_i * lambda_i_μm) / denominator

        # 过滤异常厚度（碳化硅外延层常见厚度0.5-50μm）
        if 0.1 < thickness < 100:
            results.append({
                "lambda_nm": lambda_i_nm,
```

```python
                "thickness_μm": thickness,
                "k_i": k_i,
                "n_i": n_i
            })
            # 计算残差（验证模型一致性）
            expected_k_val = (2 * thickness * denominator) / lambda_i_μm
            residuals.append(k_i - expected_k_val)
            expected_k.append(expected_k_val)
            actual_k.append(k_i)

    thickness_df = pd.DataFrame(results)
    if thickness_df.empty:
        print("无有效厚度值，需调整厚度过滤范围")
        return thickness_df, np.nan, np.nan, np.nan, pd.DataFrame()

    # 计算厚度统计量
    avg_thickness = thickness_df["thickness_μm"].mean()
    std_thickness = thickness_df["thickness_μm"].std()
    rsd = (std_thickness / avg_thickness) * 100 if avg_thickness != 0 else np.inf

    # 残差数据
    residual_data = pd.DataFrame({
        "lambda_nm": thickness_df["lambda_nm"],
        "actual_k": actual_k,
        "expected_k": expected_k,
        "residual": residuals,
        "thickness_μm": thickness_df["thickness_μm"]
    })

    print(f"{ref_model}模型：平均厚度{avg_thickness:.4f}μm，RSD={rsd:.2f}%")
    return thickness_df, avg_thickness, std_thickness, rsd, residual_data


# -------------------------- 5. 优化子区间搜索（适配实际波长范围） --------------------------
def find_best_bands(data_valid, incident_angle, model_name, model_params, window_size=800, step_size=200,
                    min_extrema=3):
    """
    window_size：子区间宽度（800nm，适配2500-5000nm范围，可生成多个子区间）
    step_size：步长（200nm，增加子区间数量）
    min_extrema：每个子区间至少3个极值点（降低门槛）
    """
    all_bands_results = []
    min_lambda = data_valid["lambda_nm"].min()
    max_lambda = data_valid["lambda_nm"].max()
    ref_index_func = cauchy_refractive_index if model_name == "cauchy" else sellmeier_refractive_index

    # 生成子区间（确保不超出数据范围）
    for start_lambda in np.arange(min_lambda, max_lambda - window_size + 100, step_size):
        end_lambda = start_lambda + window_size
        band_data = data_valid[(data_valid["lambda_nm"] >= start_lambda) & (data_valid["lambda_nm"] <= end_lambda)]

        # 子区间数据量≥30个点才分析
```

```python
        if len(band_data) < 30:
            continue

        try:
            extrema = detect_interference_extrema(band_data)
            if len(extrema) < min_extrema:
                continue

            # 计算该区间厚度
            _, avg_thick, _, rsd, residuals = calculate_thickness(extrema, incident_angle, model_name, **model_params)

            # 过滤合理结果（RSD<50%，残差非空）
            if pd.notna(rsd) and rsd < 50 and residuals is not None and not residuals.empty:
                # 计算折射率范围
                lambda_min_μm = start_lambda / 1000.0
                lambda_max_μm = end_lambda / 1000.0
                n_min = ref_index_func(lambda_max_μm, **model_params)  # 波长越大，折射率越小
                n_max = ref_index_func(lambda_min_μm, **model_params)

                all_bands_results.append({
                    "start_lambda": start_lambda,
                    "end_lambda": end_lambda,
                    "avg_thickness": avg_thick,
                    "rsd": rsd,
                    "extrema_count": len(extrema),
                    "n_min": n_min,
                    "n_max": n_max,
                    "residual_std": residuals["residual"].std(),
                    "residual_mean": residuals["residual"].mean(),
                    "residual_rmse": np.sqrt(np.mean(residuals["residual"] ** 2))
                })
        except Exception as e:
            print(f"子区间[{start_lambda:.0f}-{end_lambda:.0f}nm]分析失败: {str(e)[:50]}")
            continue

    bands_df = pd.DataFrame(all_bands_results)
    print(f"{model_name}模型：找到{len(bands_df)}个有效子区间")
    return bands_df


# ------------------------- 6. 优化加权厚度计算（降低筛选门槛） --------------------------
def get_weighted_thickness(band_results_df, rsd_threshold=30.0, top_n=3):
    """降低RSD阈值（30%），确保有足够区间参与加权"""
    if band_results_df.empty:
        return np.nan, None

    # 筛选优良区间（RSD<30%且残差RMSE<0.5）
    good_bands = band_results_df[
        (band_results_df['rsd'] < rsd_threshold) &
        (band_results_df['residual_rmse'] < 0.5)
        ].copy()

```

```python
    # 若无优良区间，取综合得分前3的区间
    if good_bands.empty:
        band_results_df['combined_score'] = band_results_df['rsd'] * 0.6 + band_results_df['residual_rmse'] * 0.4
        good_bands = band_results_df.sort_values('combined_score').head(top_n).copy()
        print(f"无满足RSD<{rsd_threshold}%的区间，取综合得分前{top_n}的区间")

    # 计算权重（基于RSD和残差，避免除以零）
    good_bands['weight_rsd'] = 1 / (good_bands['rsd'] + 1e-6)
    good_bands['weight_residual'] = 1 / (good_bands['residual_rmse'] + 1e-6)
    good_bands['weight'] = (good_bands['weight_rsd'] + good_bands['weight_residual']) / 2
    total_weight = good_bands['weight'].sum()
    good_bands['norm_weight'] = good_bands['weight'] / total_weight

    # 加权平均厚度
    weighted_avg = np.sum(good_bands['avg_thickness'] * good_bands['weight']) / total_weight
    print(f"加权平均厚度：{weighted_avg:.4f}μm（基于{len(good_bands)}个区间）")
    return weighted_avg, good_bands


# -------------------------- 7. 可视化与报告函数：增加空值防护 --------------------------
def plot_spectrum_with_extrema(data_valid, extrema, title, save_path=None):
    plt.figure(figsize=(12, 6))
    plt.plot(data_valid["lambda_nm"], data_valid["反射率 (%)"], color='grey', alpha=0.8, label='反射光谱')

    # 标记极值点（增加空值判断）
    if not extrema.empty:
        max_points = extrema[extrema['type'] == 'max']
        min_points = extrema[extrema['type'] == 'min']
        plt.scatter(max_points['lambda_nm'], max_points['反射率 (%)'], color=color1, s=50, label='极大值点', zorder=5)
        plt.scatter(min_points['lambda_nm'], min_points['反射率 (%)'], color=color2, s=50, label='极小值点', zorder=5)

    plt.title(title, fontsize=14)
    plt.xlabel('波长 (nm)', fontsize=12)
    plt.ylabel('反射率 (%)', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.legend()
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.close()


def save_text_report(models_results, final_thickness, incident_angle, file_path):
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    txt_path = f"output/txt/report_angle_{incident_angle}_{timestamp}.txt"
    with open(txt_path, 'w', encoding='utf-8') as f:
        f.write("=" * 60 + "\n")
        f.write(f"薄膜厚度分析报告 - 入射角 {incident_angle}°\n")
```

```
        f.write(f"分析时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write(f"分析文件: {file_path}\n")
        f.write("=" * 60 + "\n\n")

        # Cauchy模型（增加空值判断）
        cauchy = models_results.get('cauchy', {})
        f.write("1. Cauchy 模型分析结果:\n")
        f.write(f"   - 加权平均厚度: {cauchy.get('final_thickness', 'N/A'):.4f} μm\n" if pd.notna(
            cauchy.get('final_thickness')) else "   - 加权平均厚度: N/A\n")
        details_c = cauchy.get('details')
        f.write(f"   - 有效分析区间数量: {len(details_c) if (details_c is not None and not details_c.empty) else 0}\n")

        if details_c is not None and not details_c.empty:
            overall_residual_std = details_c['residual_std'].mean()
            overall_residual_rmse = details_c['residual_rmse'].mean()
            f.write(f"   - 整体稳定性指标:\n")
            f.write(f"     残差标准差: {overall_residual_std:.4f}\n")
            f.write(f"     残差均方根误差: {overall_residual_rmse:.4f}\n")
            f.write("   - 各区间详情:\n")
            for _, row in details_c.iterrows():
                f.write(f"     波长范围: {row['start_lambda']:.0f}-{row['end_lambda']:.0f} nm, "
                        f"厚度: {row['avg_thickness']:.4f} μm, "
                        f"RSD: {row['rsd']:.2f}%, "
                        f"残差RMSE: {row['residual_rmse']:.4f}, "
                        f"权重: {row['norm_weight']:.2f}, "
                        f"折射率范围: {row['n_min']:.4f}-{row['n_max']:.4f}\n")

        # Sellmeier模型（同理）
        sellmeier = models_results.get('sellmeier', {})
        f.write("\n2. Sellmeier 模型分析结果:\n")
        f.write(f"   - 加权平均厚度: {sellmeier.get('final_thickness', 'N/A'):.4f} μm\n" if pd.notna(
            sellmeier.get('final_thickness')) else "   - 加权平均厚度: N/A\n")
        details_s = sellmeier.get('details')
        f.write(f"   - 有效分析区间数量: {len(details_s) if (details_s is not None and not details_s.empty) else 0}\n")

        if details_s is not None and not details_s.empty:
            overall_residual_std = details_s['residual_std'].mean()
            overall_residual_rmse = details_s['residual_rmse'].mean()
            f.write(f"   - 整体稳定性指标:\n")
            f.write(f"     残差标准差: {overall_residual_std:.4f}\n")
            f.write(f"     残差均方根误差: {overall_residual_rmse:.4f}\n")
            f.write("   - 各区间详情:\n")
            for _, row in details_s.iterrows():
                f.write(f"     波长范围: {row['start_lambda']:.0f}-{row['end_lambda']:.0f} nm, "
                        f"厚度: {row['avg_thickness']:.4f} μm, "
                        f"RSD: {row['rsd']:.2f}%, "
                        f"残差RMSE: {row['residual_rmse']:.4f}, "
                        f"权重: {row['norm_weight']:.2f}, "
```

```
                        f"折射率范围: {row['n_min']:.4f}-{row['n_max']:.4f}\n")

        # 综合结果
        f.write("\n" + "=" * 40 + "\n")
        f.write(f"最终综合厚度: {final_thickness:.4f} μm\n" if pd.notna(final_thickness) else "最终综合厚度: N/A\n")
        f.write("=" * 40 + "\n")
    return txt_path


def generate_final_report(data_valid, models_results, final_thickness, incident_angle):
    fig = plt.figure(figsize=(16, 18))
    gs = fig.add_gridspec(5, 2)

    # 子图1: 反射光谱及极值点
    ax1 = fig.add_subplot(gs[0, :])
    ax1.plot(data_valid["lambda_nm"], data_valid["反射率 (%)"], color='grey', alpha=0.8, label='原始反射光谱')
    try:
        extrema = detect_interference_extrema(data_valid)
        if not extrema.empty:
            max_points = extrema[extrema['type'] == 'max']
            min_points = extrema[extrema['type'] == 'min']
            ax1.scatter(max_points['lambda_nm'], max_points['反射率 (%)'], color=color1, s=30, label='极大值点',
                        zorder=5)
            ax1.scatter(min_points['lambda_nm'], min_points['反射率 (%)'], color=color2, s=30, label='极小值点',
                        zorder=5)
    except Exception as e:
        print(f"绘制光谱图时极值点异常: {e}")

    # 标记优选区间（增加空值判断）
    colors = {'cauchy': color2, 'sellmeier': color1}
    for model, result in models_results.items():
        details = result.get('details')
        if details is not None and not details.empty:
            for _, row in details.iterrows():
                ax1.axvspan(row['start_lambda'], row['end_lambda'], color=colors[model], alpha=0.1)

    # 图例
    legend_elements = [
        plt.Line2D([0], [0], color='grey', lw=2, label='原始光谱'),
        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color1, markersize=8, label='极大值点'),
        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color2, markersize=8, label='极小值点'),
        Patch(facecolor=color2, alpha=0.3, label='Cauchy优选区间'),
        Patch(facecolor=color1, alpha=0.3, label='Sellmeier优选区间')
    ]
    ax1.legend(handles=legend_elements)
    ax1.set_title(f'入射角 {incident_angle}° 的反射光谱及优选分析区间', fontsize=14)
    ax1.set_xlabel('波长 (nm)')
    ax1.set_ylabel('反射率 (%)')
```

```python
    ax1.grid(True, linestyle='--', alpha=0.5)

    # 子图2-3：各模型区间权重（空值防护）
    for i, (model, result) in enumerate(models_results.items()):
        ax = fig.add_subplot(gs[1, i])
        details = result.get('details')
        if details is not None and not details.empty:
            details['band_label'] = details.apply(
                lambda row: f"{int(row['start_lambda'] / 1000)}-{int(row['end_lambda'] / 1000)}k", axis=1)
            bars = ax.bar(details['band_label'], details['norm_weight'], color=colors[model], alpha=0.7)
            ax.set_title(f'{model.capitalize()} 模型各区间归一化权重', fontsize=13)
            ax.set_ylabel('归一化权重')
            ax.set_ylim(0, 1.1)
            ax.tick_params(axis='x', rotation=45)
            for bar, weight in zip(bars, details['norm_weight']):
                ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{weight:.2f}', ha='center', va='bottom',
                        fontsize=9)
        else:
            ax.text(0.5, 0.5, '无有效区间', ha='center', va='center', transform=ax.transAxes)
            ax.set_title(f'{model.capitalize()} 模型分析结果')
            ax.set_xticks([])

    # 子图4-5：厚度分布（空值防护）
    for i, (model, result) in enumerate(models_results.items()):
        ax = fig.add_subplot(gs[2, i])
        details = result.get('details')
        if details is not None and not details.empty:
            ax.hist(details['avg_thickness'], bins=5, alpha=0.7, color=colors[model])
            mean_thick = details['avg_thickness'].mean()
            ax.axvline(mean_thick, color='red', linestyle='--', linewidth=2, label=f'平均: {mean_thick:.4f} μm')
            ax.set_title(f'{model.capitalize()} 模型厚度分布', fontsize=13)
            ax.set_xlabel('厚度 (μm)')
            ax.set_ylabel('区间数量')
            ax.legend()
        else:
            ax.text(0.5, 0.5, '无有效数据', ha='center', va='center', transform=ax.transAxes)
            ax.set_xticks([])

    # 子图6-7：残差统计（空值防护）
    for i, (model, result) in enumerate(models_results.items()):
        ax = fig.add_subplot(gs[3, i])
        details = result.get('details')
        if details is not None and not details.empty:
            stats_data = [
                details['residual_mean'].mean(),
                details['residual_std'].mean(),
                details['residual_rmse'].mean()
            ]
            bars = ax.bar(['平均残差', '残差标准差', '残差RMSE'], stats_data, color=color4, alpha=0.7)
```

```python
            ax.set_title(f'{model.capitalize()} 模型残差统计', fontsize=13)
            ax.set_ylabel('值')
            ax.tick_params(axis='x', rotation=30)
            for bar, val in zip(bars, stats_data):
                ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{val:.4f}', ha='center', va='bottom',
                        fontsize=9)
        else:
            ax.text(0.5, 0.5, '无残差数据', ha='center', va='center', transform=ax.transAxes)
            ax.set_xticks([])

    # 子图8：最终结果汇总
    ax8 = fig.add_subplot(gs[4, :])
    ax8.axis('off')
    t_c = models_results.get('cauchy', {}).get('final_thickness', 'N/A')
    t_s = models_results.get('sellmeier', {}).get('final_thickness', 'N/A')
    c_details = models_results.get('cauchy', {}).get('details')
    s_details = models_results.get('sellmeier', {}).get('details')
    c_count = len(c_details) if (c_details is not None and not c_details.empty) else 0
    s_count = len(s_details) if (s_details is not None and not s_details.empty) else 0
    c_rmse = c_details['residual_rmse'].mean() if (c_details is not None and not c_details.empty) else 'N/A'
    s_rmse = s_details['residual_rmse'].mean() if (s_details is not None and not s_details.empty) else 'N/A'

    summary_text = (
        f"最终分析结果汇总 (入射角: {incident_angle}°)\n\n"
        f"1. Cauchy 模型:\n"
        f"   - 加权平均厚度: {t_c:.4f} μm\n" if pd.notna(t_c) else "   - 加权平均厚度: N/A\n"
                                                                   f"   - 分析基于 {c_count} 个优良波长区间\n"
                                                                   f"   - 平均残差RMSE: {c_rmse:.4f}\n" if isinstance(
            c_rmse, (int, float)) else "   - 平均残差RMSE: N/A\n\n"
                                       f"2. Sellmeier 模型:\n"
                                       f"   - 加权平均厚度: {t_s:.4f} μm\n" if pd.notna(t_s) else "   - 加权平均厚度: N/A\n"
                       f"   - 分析基于 {s_count} 个优良波长区间\n"
                       f"   - 平均残差RMSE: {s_rmse:.4f}\n" if isinstance(
            s_rmse, (int, float)) else "   - 平均残差RMSE: N/A\n\n"
                                       f"--------------------------------------------------\n"
                                       f"综合两个模型的稳定性加权平均后，\n"
                                       f"最终计算得到的薄膜厚度为: {final_thickness:.4f} μm" if pd.notna(
            final_thickness) else "最终计算得到的薄膜厚度为: N/A"
    )
    ax8.text(0.5, 0.5, summary_text, ha='center', va='center', fontsize=12,
             bbox=dict(boxstyle="round,pad=0.5", fc='aliceblue', ec='steelblue', lw=1))

    plt.tight_layout()
```

```python
    report_path = f"output/images/final_report_angle_{incident_angle}.png"
    plt.savefig(report_path, dpi=300)
    plt.close()
    return report_path


# -------------------------- 8. 主函数：增加日志输出，便于调试 ---------------------------
def main(file_path, incident_angle):
    print("=" * 60)
    print(f"开始分析文件: {file_path}, 入射角: {incident_angle}°")
    print("=" * 60)
    try:
        # 步骤1：数据预处理
        data_valid = preprocess_data(file_path, incident_angle)
        print("步骤 1/6: 数据预处理完成。")

        # 步骤2：绘制光谱及极值点图
        try:
            extrema = detect_interference_extrema(data_valid)
            spec_path = f"output/images/反射光谱及干涉极值点 (入射角{incident_angle}°).png"
            plot_spectrum_with_extrema(data_valid, extrema, f'反射光谱及干涉极值点（入射角 {incident_angle}°）',
                                       spec_path)
            print(f"已保存光谱图: {spec_path}")
        except Exception as e:
            print(f"生成光谱图失败: {e}")

        # 步骤3：模型参数（碳化硅适配）
        cauchy_params = {"A": 2.65, "B": 0.015, "C": 1e-7}  # Cauchy标准参数
        sellmeier_params = {"A1": 6.91, "B1": 0.202}  # Sellmeier碳化硅参数
        models_results = {}

        # 步骤4：各模型计算
        for model_name, params in [("cauchy", cauchy_params), ("sellmeier", sellmeier_params)]:
            print(f"\n--- 正在为 [{model_name.capitalize()}] 模型寻找最佳分析区间... ---")
            bands_df = find_best_bands(data_valid, incident_angle, model_name, params)

            if bands_df.empty:
                print(f"警告: {model_name.capitalize()} 模型未找到有效区间，跳过加权计算")
                models_results[model_name] = {'final_thickness': np.nan, 'details': None}
                continue

            # 计算加权厚度
            weighted_thickness, good_bands = get_weighted_thickness(bands_df)
            if pd.isna(weighted_thickness):
                print(f"警告: {model_name.capitalize()} 模型无法计算加权厚度")
                models_results[model_name] = {'final_thickness': np.nan, 'details': None}
            else:
```

```
                models_results[model_name] = {'final_thickness': weighted_thickness, 'details': good_bands}
                print(f"步骤 3/6: {model_name.capitalize()} 模型计算完成")

        # 步骤5：综合两个模型结果
        print("\n--- 正在综合两个模型的结果... ---")
        t_cauchy = models_results.get("cauchy", {}).get('final_thickness')
        t_sellmeier = models_results.get("sellmeier", {}).get('final_thickness')
        valid_thickness = [t for t in [t_cauchy, t_sellmeier] if pd.notna(t)]

        if not valid_thickness:
            final_thickness = np.nan
            print("错误: 两个模型均无有效厚度")
        else:
            # 基于残差RMSE计算模型权重
            c_details = models_results.get('cauchy', {}).get('details')
            s_details = models_results.get('sellmeier', {}).get('details')
            c_rmse = c_details['residual_rmse'].mean() if (c_details is not None and not c_details.empty) else np.inf
            s_rmse = s_details['residual_rmse'].mean() if (s_details is not None and not s_details.empty) else np.inf

            # 权重 = 1/残差RMSE（残差越小，权重越大）
            weight_c = 1 / c_rmse if c_rmse != np.inf else 0
            weight_s = 1 / s_rmse if s_rmse != np.inf else 0
            total_weight = weight_c + weight_s

            if total_weight == 0:
                final_thickness = np.mean(valid_thickness)
                print(f"模型权重计算失败，使用简单平均: {final_thickness:.4f} μm")
            else:
                norm_c = weight_c / total_weight
                norm_s = weight_s / total_weight
                final_thickness = (t_cauchy * norm_c) + (t_sellmeier * norm_s)
                print(f"最终综合厚度: {final_thickness:.4f} μm")
                print(f"模型权重 - Cauchy: {norm_c:.2f}, Sellmeier: {norm_s:.2f}")

        # 步骤6：稳定性分析
        print("\n步骤 4/6: 执行稳定性分析...")
        # 简化稳定性分析（避免空值报错）
        stability_path = f"output/stability/model_consistency_{incident_angle}.png"
        try:
            plot_model_consistency(models_results, incident_angle, stability_path)
            print(f"稳定性分析图已保存: {stability_path}")
        except Exception as e:
            print(f"稳定性分析失败: {e}")

        # 步骤7：生成报告
        print("\n步骤 5/6: 生成可视化分析报告...")
        try:
```

```python
            report_path = generate_final_report(data_valid, models_results, final_thickness, incident_angle)
            print(f"分析报告已保存至: {report_path}")
        except Exception as e:
            print(f"生成报告失败: {e}")

        # 步骤8: 保存Excel和文本报告
        print("\n步骤 6/6: 保存结果数据...")
        try:
            excel_path = save_excel_results(models_results, final_thickness, incident_angle)
            print(f"Excel结果已保存至: {excel_path}")
        except Exception as e:
            print(f"保存Excel失败: {e}")

        try:
            txt_path = save_text_report(models_results, final_thickness, incident_angle, file_path)
            print(f"文本报告已保存至: {txt_path}")
        except Exception as e:
            print(f"保存文本报告失败: {e}")

        print("\n" + "=" * 60)
        print(f"分析完成! 最终厚度: {final_thickness:.4f} μm" if pd.notna(final_thickness) else "分析完成，但无有效厚度")
        print("=" * 60)

    except Exception as e:
        print(f"\n处理过程中发生错误: {str(e)}")
        print("请检查文件路径和数据格式")


# -------------------------- 9. 其他辅助函数（保持原逻辑，增加空值防护） ---------------------------
def plot_thickness_distribution(thickness_data, model_name, title, save_path=None):
    if len(thickness_data) == 0:
        print("无厚度数据，跳过绘图")
        return
    plt.figure(figsize=(10, 6))
    plt.hist(thickness_data, bins=10, alpha=0.7, color=color3)
    plt.axvline(np.mean(thickness_data), color='red', linestyle='dashed', linewidth=2,
                label=f'平均值: {np.mean(thickness_data):.4f} μm')
    plt.title(f'{model_name} 模型厚度分布 - {title}', fontsize=13)
    plt.xlabel('厚度 (μm)')
    plt.ylabel('频率')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.legend()
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.close()


def plot_residual_analysis(residual_data, model_name, angle, save_path=None):
    if residual_data.empty:
```

```python
        print("无残差数据，跳过绘图")
        return
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
    ax1.scatter(residual_data["lambda_nm"], residual_data["residual"], color=color4, alpha=0.7)
    ax1.axhline(y=0, color='r', linestyle='--', alpha=0.5)
    ax1.set_title(f'{model_name} 模型残差随波长变化')
    ax1.set_xlabel('波长 (nm)')
    ax1.set_ylabel('残差 (k_i - 预期k值)')
    ax1.grid(True, linestyle='--', alpha=0.5)

    ax2.hist(residual_data["residual"], bins=10, alpha=0.7, color=color4)
    ax2.axvline(x=0, color='r', linestyle='--', alpha=0.5)
    ax2.set_title(f'{model_name} 模型残差分布')
    ax2.set_xlabel('残差 (k_i - 预期k值)')
    ax2.set_ylabel('频率')
    ax2.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.close()


def plot_qq_residuals(residual_data, model_name, angle, save_path=None):
    if residual_data.empty:
        print("无残差数据，跳过Q-Q图")
        return
    plt.figure(figsize=(8, 6))
    stats.probplot(residual_data["residual"], plot=plt)
    plt.title(f'{model_name} 模型残差Q-Q图 (入射角 {angle}°)')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.close()


def plot_model_consistency(models_results, angle, save_path=None):
    cauchy_data = models_results.get('cauchy', {}).get('details')
    sellmeier_data = models_results.get('sellmeier', {}).get('details')
    if cauchy_data is None or sellmeier_data is None or cauchy_data.empty or sellmeier_data.empty:
        print("无足够数据绘制模型一致性图")
        return

    plt.figure(figsize=(10, 8))
    plt.scatter(cauchy_data['avg_thickness'], sellmeier_data['avg_thickness'], color=color4, alpha=0.7, s=50)
    min_val = min(cauchy_data['avg_thickness'].min(), sellmeier_data['avg_thickness'].min())
    max_val = max(cauchy_data['avg_thickness'].max(), sellmeier_data['avg_thickness'].max())
    plt.plot([min_val, max_val], [min_val, max_val], 'r--', alpha=0.7, label='理想一致性线 (y=x)')

    # 计算相关系数
    corr_coef = np.corrcoef(cauchy_data['avg_thickness'], sellmeier_data['avg_thickness'])[0, 1]
```

```python
    plt.text(0.05, 0.95, f'相关系数: r = {corr_coef:.4f}', transform=plt.gca().transAxes,
             bbox=dict(facecolor='white', alpha=0.8))

    plt.title(f'Cauchy与Sellmeier模型厚度结果一致性 (入射角 {angle}°)')
    plt.xlabel('Cauchy模型厚度 (μm)')
    plt.ylabel('Sellmeier模型厚度 (μm)')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.legend()
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.close()


def save_excel_results(models_results, final_thickness, incident_angle):
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    excel_path = f"output/excel/results_angle_{incident_angle}_{timestamp}.xlsx"
    with pd.ExcelWriter(excel_path) as writer:
        # 汇总结果
        cauchy_rmse = models_results.get('cauchy', {}).get('details', pd.DataFrame()).get('residual_rmse').mean() if \
            (models_results.get('cauchy', {}).get('details') is not None and not models_results.get('cauchy', {}).get(
                'details').empty) else 'N/A'
        sellmeier_rmse = models_results.get('sellmeier', {}).get('details', pd.DataFrame()).get(
            'residual_rmse').mean() if \
            (models_results.get('sellmeier', {}).get('details') is not None and not models_results.get('sellmeier',
                                                                                                        {}).get(
                'details').empty) else 'N/A'

        summary_data = {
            "模型": ["Cauchy", "Sellmeier", "综合结果"],
            "厚度 (μm)": [
                models_results.get('cauchy', {}).get('final_thickness', 'N/A'),
                models_results.get('sellmeier', {}).get('final_thickness', 'N/A'),
                final_thickness if pd.notna(final_thickness) else 'N/A'
            ],
            "有效区间数量": [
                len(models_results.get('cauchy', {}).get('details', [])) if (
                            models_results.get('cauchy', {}).get('details') is not None) else 0,
                len(models_results.get('sellmeier', {}).get('details', [])) if (
                            models_results.get('sellmeier', {}).get('details') is not None) else 0,
                "-"
            ],
            "平均残差RMSE": [
                cauchy_rmse if isinstance(cauchy_rmse, (int, float)) else 'N/A',
```

```
                sellmeier_rmse if isinstance(sellmeier_rmse, (int, float)) else 'N/A',
                "-"
            ]
        }
        pd.DataFrame(summary_data).to_excel(writer, sheet_name="汇总结果", index=False)

        # 各模型详细区间
        for model, result in models_results.items():
            details = result.get('details')
            if details is not None and not details.empty:
                details.to_excel(writer, sheet_name=f"{model}_区间详情", index=False)
    return excel_path


def perform_stability_analysis(models_results, data_valid, incident_angle):
    """简化稳定性分析，避免空值报错"""
    for model_name, result in models_results.items():
        details = result.get('details')
        if details is None or details.empty:
            print(f"{model_name}模型无有效数据，跳过稳定性分析")
            continue

        try:
            # 取第一个有效区间计算残差
            start_lambda = details.iloc[0]['start_lambda']
            end_lambda = details.iloc[0]['end_lambda']
            band_data = data_valid[(data_valid["lambda_nm"] >= start_lambda) & (data_valid["lambda_nm"] <= end_lambda)]
            extrema = detect_interference_extrema(band_data)
            model_params = {"A": 2.65, "B": 0.015, "C": 1e-7} if model_name == "cauchy" else {"A1": 6.91, "B1": 0.202}
            _, _, _, _, residual_data = calculate_thickness(extrema, incident_angle, model_name, **model_params)

            if residual_data is not None and not residual_data.empty:
                # 保存残差图
                res_plot_path = f"output/stability/{model_name}_residual_analysis_{incident_angle}.png"
                plot_residual_analysis(residual_data, model_name.capitalize(), incident_angle, res_plot_path)
                qq_plot_path = f"output/stability/{model_name}_residual_qq_{incident_angle}.png"
                plot_qq_residuals(residual_data, model_name.capitalize(), incident_angle, qq_plot_path)
                print(f"{model_name}模型残差图已保存")
        except Exception as e:
            print(f"{model_name}模型稳定性分析失败: {e}")

    # 模型一致性图
    consistency_path = f"output/stability/model_consistency_{incident_angle}.png"
    plot_model_consistency(models_results, incident_angle, consistency_path)
```

```python
# -------------------------- 10. 运行入口（确保文件路径正确） ---------------------------
if __name__ == "__main__":
    # 注意：请将文件路径改为你的实际路径（相对路径/绝对路径均可）
    file1_path = "附件1_processed.xlsx"
    angle1 = 10
    file2_path = "附件2_processed.xlsx"
    angle2 = 15

    # 运行分析
    print("=" * 80)
    print("开始分析附件1...")
    main(file_path=file1_path, incident_angle=angle1)

    print("\n" + "=" * 80)
    print("开始分析附件2...")
    main(file_path=file2_path, incident_angle=angle2)
```

Problem2_灵敏度单独.py

```python
import warnings
import pandas as pd
import numpy as np
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
from math import sin, sqrt, pi
import os
from datetime import datetime

# --- 1. 全局设置（沿用Solution，修复颜色bug）---
warnings.filterwarnings('ignore')
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 12
# 颜色配置（避免'midgray'错误，用标准颜色）
color_cauchy = "#00BA38"  # Cauchy模型（和Solution一致）
color_sellmeier = "#619CFF"  # Sellmeier模型（和Solution一致）
color_avg = "#C86193"  # 平均值（和Solution一致）
grid_color = 'gray'  # 修复颜色错误
text_color = 'black'
bg_color = 'lightgray'

# 确保输出目录（用户要求的"灵敏度分析"文件夹）
output_root = "output/灵敏度分析"
os.makedirs(output_root, exist_ok=True)
os.makedirs(f"{output_root}/excel", exist_ok=True)
os.makedirs(f"{output_root}/images", exist_ok=True)


# --- 2. 核心物理模型（完全移植Solution，确保参数一致）---
def cauchy_refractive_index(lambda_μm, A=2.65, B=0.015, C=1e-7):
    """Cauchy模型：适配碳化硅红外波段（2.5-5μm）- 与Solution完全一致"""
    n = A + B / (lambda_μm ** 2) + C / (lambda_μm ** 4)
    return n


def sellmeier_refractive_index(lambda_μm, A1=6.91, B1=0.202):
```

```
    """Sellmeier模型：碳化硅标准参数 - 与Solution完全一致"""
    if lambda_μm ** 2 <= B1:
        raise ValueError(f"波长 {lambda_μm:.2f}μm 过小，需λ² > {B1}μm²")
    n_squared = 1 + (A1 * lambda_μm ** 2) / (lambda_μm ** 2 - B1)
    return sqrt(n_squared)


# --- 3. 数据预处理（完全移植Solution，确保数据范围一致）---
def preprocess_data(file_path, angle):
    try:
        data = pd.read_excel(file_path)
    except Exception:
        data = pd.read_csv(file_path)
    # 数据清洗与波长转换（和Solution逻辑一致）
    data.columns = ["波数 (cm-1)", "反射率 (%)"]
    data["波数 (cm-1)"] = pd.to_numeric(data["波数 (cm-1)"], errors='coerce')
    data["反射率 (%)"] = pd.to_numeric(data["反射率 (%)"], errors='coerce')
    data = data.dropna()
    data["lambda_nm"] = 1e7 / data["波数 (cm-1)"]
    # 修正波长范围（2500-5000nm，和Solution一致）
    valid_mask = (data["lambda_nm"] >= 2500) & (data["lambda_nm"] <= 5000)
    data_valid = data[valid_mask].sort_values("lambda_nm", ascending=False).reset_index(drop=True)
    if len(data_valid) < 50:
        raise ValueError(f"有效数据点仅{len(data_valid)}个，需≥50个")
    data_valid["incident_angle"] = angle
    print(
        f"预处理后[{os.path.basename(file_path)}]：波长{data_valid['lambda_nm'].min():.0f}-{data_valid['lambda_nm'].max():.0f}nm，共{len(data_valid)}个点")
    return data_valid


# --- 4. 极值点检测（完全移植Solution，确保检测精度）---
def detect_interference_extrema(data_valid):
    R = data_valid["反射率 (%)"].values
    lambda_nm = data_valid["lambda_nm"].values
    r_mean = np.mean(R)
    r_std = np.std(R)
    # 和Solution一致的检测参数
    min_height_max = r_mean + 0.2 * r_std
    min_height_min = r_mean - 0.8 * r_std
    min_distance = max(10, int(len(data_valid) * 0.02))
    prominence = 0.05
    # 检测峰谷
    peak_indices, _ = find_peaks(R, distance=min_distance, height=min_height_max, prominence=prominence)
    valley_indices, _ = find_peaks(-R, distance=min_distance, height=-min_height_min, prominence=prominence)
    # 整合极值点
    extrema = pd.concat([
        pd.DataFrame({"type": "max", "lambda_nm": lambda_nm[peak_indices], "反射率 (%)": R[peak_indices]}),
        pd.DataFrame({"type": "min", "lambda_nm": lambda_nm[valley_indices], "反射率 (%)": R[valley_indices]})
    ]).sort_values("lambda_nm", ascending=False).reset_index(drop=True)
```

```
    print(f"检测到极值点：共{len(extrema)}个（极大值{len(peak_indices)}个，极小值{len(valley_indices)}个）")
    if len(extrema) < 3:
        raise ValueError(f"极值点不足3个，需调整检测参数")
    return extrema


# --- 5. 厚度计算（完全移植Solution，含异常值过滤）---
def calculate_thickness(extrema, incident_angle, ref_model, **model_params):
    if len(extrema) < 2:
        return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
    ref_index_func = cauchy_refractive_index if ref_model == "cauchy" else sellmeier_refractive_index
    angle_rad = incident_angle * pi / 180
    lambda0_nm = extrema.iloc[0]["lambda_nm"]
    k0_estimates = []
    # 和Solution一致的k0计算（波长差>10nm）
    for i in range(1, len(extrema)):
        m_i = i * 0.5
        lambda_i_nm = extrema.iloc[i]["lambda_nm"]
        lambda_diff = lambda0_nm - lambda_i_nm
        if lambda_diff > 10:
            k0_est = (m_i * lambda_i_nm) / lambda_diff
            k0_estimates.append(k0_est)
    if not k0_estimates:
        print("无有效k0估算值，需调整波长差阈值")
        return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
    k0_final = np.median(k0_estimates)
    results = []
    residuals = []
    expected_k = []
    actual_k = []
    # 厚度计算与异常值过滤（0.1-100μm，和Solution一致）
    for i, row in extrema.iterrows():
        m_i = i * 0.5
        k_i = k0_final + m_i
        lambda_i_nm = row["lambda_nm"]
        lambda_i_μm = lambda_i_nm / 1000.0
        try:
            n_i = ref_index_func(lambda_i_μm, **model_params)
        except Exception as e:
            print(f"计算折射率失败：{e}，跳过该点")
            continue
        denominator = 2 * sqrt(n_i ** 2 - sin(angle_rad) ** 2)
        thickness = (k_i * lambda_i_μm) / denominator
        if 0.1 < thickness < 100:
            results.append({"lambda_nm": lambda_i_nm, "thickness_μm": thickness, "k_i": k_i, "n_i": n_i})
            expected_k_val = (2 * thickness * denominator) / lambda_i_μm
            residuals.append(k_i - expected_k_val)
            expected_k.append(expected_k_val)
            actual_k.append(k_i)
    thickness_df = pd.DataFrame(results)
    if thickness_df.empty:
        print("无有效厚度值，需调整厚度过滤范围")
        return thickness_df, np.nan, np.nan, np.nan, pd.DataFrame()
    # 统计量计算（和Solution一致）
```

```python
    avg_thickness = thickness_df["thickness_μm"].mean()
    std_thickness = thickness_df["thickness_μm"].std()
    rsd = (std_thickness / avg_thickness) * 100 if avg_thickness != 0 else np.inf
    residual_data = pd.DataFrame({
        "lambda_nm": thickness_df["lambda_nm"], "actual_k": actual_k, "expected_k": expected_k,
        "residual": residuals, "thickness_μm": thickness_df["thickness_μm"]
    })
    print(f"{ref_model}模型：平均厚度{avg_thickness:.4f}μm，RSD={rsd:.2f}%")
    return thickness_df, avg_thickness, std_thickness, rsd, residual_data


# --- 6. 子区间搜索（完全移植Solution，确保区间数量一致）---
def find_best_bands(data_valid, incident_angle, model_name, model_params, window_size=800, step_size=200,
                    min_extrema=3):
    """和Solution完全一致的子区间搜索：window_size=800，step_size=200"""
    all_bands_results = []
    min_lambda = data_valid["lambda_nm"].min()
    max_lambda = data_valid["lambda_nm"].max()
    ref_index_func = cauchy_refractive_index if model_name == "cauchy" else sellmeier_refractive_index
    # 生成子区间（和Solution一致的范围）
    for start_lambda in np.arange(min_lambda, max_lambda - window_size + 100, step_size):
        end_lambda = start_lambda + window_size
        band_data = data_valid[(data_valid["lambda_nm"] >= start_lambda) & (data_valid["lambda_nm"] <= end_lambda)]
        if len(band_data) < 30:
            continue
        try:
            extrema = detect_interference_extrema(band_data)
            if len(extrema) < min_extrema:
                continue
            _, avg_thick, _, rsd, residuals = calculate_thickness(extrema, incident_angle, model_name, **model_params)
            if pd.notna(rsd) and rsd < 50 and residuals is not None and not residuals.empty:
                lambda_min_μm = start_lambda / 1000.0
                lambda_max_μm = end_lambda / 1000.0
                n_min = ref_index_func(lambda_max_μm, **model_params)
                n_max = ref_index_func(lambda_min_μm, **model_params)
                all_bands_results.append({
                    "start_lambda": start_lambda, "end_lambda": end_lambda, "avg_thickness": avg_thick, "rsd": rsd,
                    "extrema_count": len(extrema), "n_min": n_min, "n_max": n_max,
                    "residual_std": residuals["residual"].std(), "residual_mean": residuals["residual"].mean(),
                    "residual_rmse": np.sqrt(np.mean(residuals["residual"] ** 2))
                })
        except Exception as e:
            print(f"子区间[{start_lambda:.0f}-{end_lambda:.0f}nm]分析失败：{str(e)[:50]}")
            continue
    bands_df = pd.DataFrame(all_bands_results)
```

```python
    print(f"{model_name}模型：找到{len(bands_df)}个有效子区间")
    return bands_df


# --- 7. 加权厚度计算（完全移植Solution，确保权重逻辑一致）---
def get_weighted_thickness(band_results_df, rsd_threshold=30.0, top_n=3):
    """和Solution一致的加权逻辑：RSD<30%，综合得分排序"""
    if band_results_df.empty:
        return np.nan, None
    # 筛选优良区间
    good_bands = band_results_df[
        (band_results_df['rsd'] < rsd_threshold) &
        (band_results_df['residual_rmse'] < 0.5)
        ].copy()
    # 无优良区间时取前3
    if good_bands.empty:
        band_results_df['combined_score'] = band_results_df['rsd'] * 0.6 + band_results_df['residual_rmse'] * 0.4
        good_bands = band_results_df.sort_values('combined_score').head(top_n).copy()
        print(f"无满足RSD<{rsd_threshold}%的区间，取综合得分前{top_n}的区间")
    # 计算权重（和Solution一致）
    good_bands['weight_rsd'] = 1 / (good_bands['rsd'] + 1e-6)
    good_bands['weight_residual'] = 1 / (good_bands['residual_rmse'] + 1e-6)
    good_bands['weight'] = (good_bands['weight_rsd'] + good_bands['weight_residual']) / 2
    total_weight = good_bands['weight'].sum()
    good_bands['norm_weight'] = good_bands['weight'] / total_weight
    # 加权平均
    weighted_avg = np.sum(good_bands['avg_thickness'] * good_bands['weight']) / total_weight
    print(f"加权平均厚度：{weighted_avg:.4f}μm（基于{len(good_bands)}个区间）")
    return weighted_avg, good_bands


# --- 8. 单角度分析（完全匹配Solution的单角度逻辑）---
def analyze_single_angle(file_path, incident_angle):
    """分析单个入射角，返回Cauchy/Sellmeier/平均厚度 - 和Solution逻辑一致"""
    print(f"\n{'=' * 50}\n开始分析：{os.path.basename(file_path)}，入射角{incident_angle}°")
    print('=' * 50)
    try:
        # 1. 数据预处理
        data_valid = preprocess_data(file_path, incident_angle)
        # 2. 模型参数（和Solution一致）
        cauchy_params = {"A": 2.65, "B": 0.015, "C": 1e-7}
        sellmeier_params = {"A1": 6.91, "B1": 0.202}
        models_results = {}
        # 3. 计算Cauchy模型
        print(f"\n--- Cauchy模型分析 ---")
        bands_cauchy = find_best_bands(data_valid, incident_angle, "cauchy", cauchy_params)
        if bands_cauchy.empty:
            print("Cauchy模型无有效区间")
            models_results['cauchy'] = np.nan
        else:
            cauchy_thick, _ = get_weighted_thickness(bands_cauchy)
```

```python
            models_results['cauchy'] = cauchy_thick
        # 4. 计算Sellmeier模型
        print(f"\n--- Sellmeier模型分析 ---")
        bands_sellmeier = find_best_bands(data_valid, incident_angle, "sellmeier", sellmeier_params)
        if bands_sellmeier.empty:
            print("Sellmeier模型无有效区间")
            models_results['sellmeier'] = np.nan
        else:
            sellmeier_thick, _ = get_weighted_thickness(bands_sellmeier)
            models_results['sellmeier'] = sellmeier_thick
        # 5. 计算平均厚度
        valid_thicks = [v for v in models_results.values() if pd.notna(v)]
        avg_thick = np.mean(valid_thicks) if valid_thicks else np.nan
        models_results['avg'] = avg_thick
        # 打印结果
        print(f"\n{incident_angle}° 分析结果：")
        print(f"Cauchy模型厚度：{models_results['cauchy']:.4f}μm" if pd.notna(
            models_results['cauchy']) else "Cauchy模型厚度：N/A")
        print(f"Sellmeier模型厚度：{models_results['sellmeier']:.4f}μm" if pd.notna(
            models_results['sellmeier']) else "Sellmeier模型厚度：N/A")
        print(f"平均厚度：{avg_thick:.4f}μm" if pd.notna(avg_thick) else "平均厚度：N/A")
        return {
            "file": os.path.basename(file_path),
            "angle": incident_angle,
            "cauchy": models_results['cauchy'],
            "sellmeier": models_results['sellmeier'],
            "avg": avg_thick
        }
    except Exception as e:
        print(f"{incident_angle}° 分析失败：{str(e)}")
        return {
            "file": os.path.basename(file_path),
            "angle": incident_angle,
            "cauchy": np.nan,
            "sellmeier": np.nan,
            "avg": np.nan
        }


# --- 9. 汇总图片绘制（用户要求：所有角度平均厚度汇总）---
def plot_summary_thickness(all_results, save_path):
    """绘制附件1和附件2所有角度的厚度汇总图"""
    # 拆分附件1和附件2数据
    file1_results = [r for r in all_results if "附件1" in r["file"]]
    file2_results = [r for r in all_results if "附件2" in r["file"]]
    # 提取数据（过滤空值）
    # 附件1
    angles1 = [r["angle"] for r in file1_results if pd.notna(r["avg"])]
    cauchy1 = [r["cauchy"] for r in file1_results if pd.notna(r["avg"])]
    sellmeier1 = [r["sellmeier"] for r in file1_results if pd.notna(r["avg"])]
    avg1 = [r["avg"] for r in file1_results if pd.notna(r["avg"])]
    # 附件2
    angles2 = [r["angle"] for r in file2_results if pd.notna(r["avg"])]
```

```python
    cauchy2 = [r["cauchy"] for r in file2_results if pd.notna(r["avg"])]
    sellmeier2 = [r["sellmeier"] for r in file2_results if pd.notna(r["avg"])]
    avg2 = [r["avg"] for r in file2_results if pd.notna(r["avg"])]

    # 创建图表
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6), sharey=True)
    width = 0.25  # 条形图宽度
    x1 = np.arange(len(angles1))
    x2 = np.arange(len(angles2))

    # 附件1子图
    ax1.bar(x1 - width, cauchy1, width, color=color_cauchy, alpha=0.8, label="Cauchy模型")
    ax1.bar(x1, sellmeier1, width, color=color_sellmeier, alpha=0.8, label="Sellmeier模型")
    ax1.bar(x1 + width, avg1, width, color=color_avg, alpha=0.8, label="平均厚度")
    ax1.set_title("附件1_processed.xlsx 厚度结果（入射角8-12°）", fontsize=14)
    ax1.set_xlabel("入射角 (°)", fontsize=12)
    ax1.set_ylabel("厚度 (μm)", fontsize=12)
    ax1.set_xticks(x1)
    ax1.set_xticklabels(angles1)
    ax1.grid(True, linestyle='--', alpha=0.5, color=grid_color)
    ax1.legend()
    # 标注数值
    for x, y in zip(x1, avg1):
        ax1.text(x + width, y + 0.02, f"{y:.4f}", ha='center', va='bottom', fontsize=9)

    # 附件2子图
    ax2.bar(x2 - width, cauchy2, width, color=color_cauchy, alpha=0.8, label="Cauchy模型")
    ax2.bar(x2, sellmeier2, width, color=color_sellmeier, alpha=0.8, label="Sellmeier模型")
    ax2.bar(x2 + width, avg2, width, color=color_avg, alpha=0.8, label="平均厚度")
    ax2.set_title("附件2_processed.xlsx 厚度结果（入射角13-17°）", fontsize=14)
    ax2.set_xlabel("入射角 (°)", fontsize=12)
    ax2.set_xticks(x2)
    ax2.set_xticklabels(angles2)
    ax2.grid(True, linestyle='--', alpha=0.5, color=grid_color)
    ax2.legend()
    # 标注数值
    for x, y in zip(x2, avg2):
        ax2.text(x + width, y + 0.02, f"{y:.4f}", ha='center', va='bottom', fontsize=9)

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()
    print(f"\n汇总图已保存：{save_path}")


# --- 10. 主函数（执行灵敏度分析，匹配用户角度要求）---
def angle_sensitivity_main(file1_path, file2_path):
    """
    灵敏度分析主流程：
```

```python
    - 附件1：8°、9°、10°、11°、12°
    - 附件2：13°、14°、15°、16°、17°
    """
    # 1. 配置角度范围（用户要求）
    file1_angles = [8, 9, 10, 11, 12]  # 附件1：10°基础扩展
    file2_angles = [13, 14, 15, 16, 17]  # 附件2：15°基础扩展
    all_results = []

    # 2. 分析附件1所有角度
    print(f"{'=' * 60}\n开始附件1_processed.xlsx 灵敏度分析（角度：
{file1_angles}°）")
    print('=' * 60)
    for angle in file1_angles:
        result = analyze_single_angle(file1_path, angle)
        all_results.append(result)

    # 3. 分析附件2所有角度
    print(f"\n{'=' * 60}\n开始附件2_processed.xlsx 灵敏度分析（角度：
{file2_angles}°）")
    print('=' * 60)
    for angle in file2_angles:
        result = analyze_single_angle(file2_path, angle)
        all_results.append(result)

    # 4. 保存Excel结果
    results_df = pd.DataFrame(all_results)
    excel_path = f"{output_root}/excel/灵敏度分析结果.xlsx"
    results_df.to_excel(excel_path, index=False)
    print(f"\nExcel结果已保存：{excel_path}")

    # 5. 绘制汇总图片（用户要求：output/灵敏度分析）
    summary_plot_path = f"{output_root}/images/汇总厚度对比图.png"
    plot_summary_thickness(all_results, summary_plot_path)

    # 6. 输出统计信息
    print(f"\n{'=' * 60}")
    print("灵敏度分析完成！")
    print(f"结果文件路径：{output_root}")
    print(f"  - Excel结果：{excel_path}")
    print(f"  - 汇总图片：{summary_plot_path}")
    print('=' * 60)


# --- 11. 运行入口（用户仅需修改文件路径）---
if __name__ == "__main__":
    # -------------------------- 用户配置区 --------------------------
    # 替换为你的附件1和附件2实际路径（绝对路径如"D:/附件1_processed.xlsx"）
    FILE1_PATH = "附件1_processed.xlsx"
    FILE2_PATH = "附件2_processed.xlsx"
    # -------------------------- 执行分析 --------------------------
    angle_sensitivity_main(file1_path=FILE1_PATH, file2_path=FILE2_PATH)
```

Problem3_solution.py

```python
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from scipy.signal import find_peaks, savgol_filter, detrend
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import os
import warnings
from datetime import datetime

# --- 1. 全局配置（修复机器精度问题+优化初始值） ---
warnings.filterwarnings('ignore')
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 12

# 颜色定义
COLOR_DATA = 'gray'
COLOR_FIT = '#F8766D'
COLOR_PEAK = "#00BFC4"
COLOR_RESIDUAL = '#7CAE00'
COLOR_AUTO_RANGE = '#FF6B6B'

# 【修复1】材料参数：固定核心参数，减少自由度（不变）
MATERIALS_PARAMS = {
    'Si': {
        'n_fixed': 3.40,  # 固定硅折射率
        'n0': 1.0003,  # 空气折射率
        'n2': 3.80,  # 衬底折射率
        'B': 0.08,  # 拟合B的参考值
        'C_fixed': 0.0003,  # 固定C参数
        'B_bounds': [0.075, 0.085],  # B的窄边界
        'phase_fixed': 0.0  # 固定相位偏移
    },
    'GaAs': {'n_fixed': 3.3, 'n0': 1.0003, 'n2': 3.6, 'B': 0.08, 'C_fixed': 0.003, 'B_bounds': [0.078, 0.082],
             'phase_fixed': 0.0},
    'SiC': {'n_fixed': 2.6, 'n0': 1.0003, 'n2': 2.8, 'B': 0.06, 'C_fixed': 0.002, 'B_bounds': [0.058, 0.062],
            'phase_fixed': 0.0}
}

# 【修复2】拟合控制：精度参数高于机器精度（2.22e-16）+ 优化初始值
MAX_FEV = 500000  # 足够迭代次数（减少至5e5，避免冗余）
SMOOTH_WINDOW = 17  # 减小平滑窗口（21→17，保留更多干涉细节）
SMOOTH_ORDER = 2  # 平滑阶数不变
MIN_PEAKS = 2  # 最低峰数2个
D_THRESHOLD = [3.0, 4.0]  # 硅外延层常见范围
PEAK_PROMINENCE = 0.01  # 最低突出度
WINDOW_SIZE = 30  # 滑动窗口不变
VAR_THRESHOLD_RATIO = 1.0  # 方差阈值不变
VIRTUAL_PEAK_NUM = 4  # 生成4个虚拟峰
FIT_TOL = 1e-15  # 拟合精度（1e-15 > 机器精度2.22e-16）

# --- 2. 输出目录自动创建 ---
```

```python
output_dirs = ['output/images', 'output/excel', 'output/reports', 'output/logs']
for dir_path in output_dirs:
    os.makedirs(dir_path, exist_ok=True)


# --- 3. 物理模型（不变，仅调用修复后的精度参数） ---
def refractive_index_model(nu, B, mat_params):
    """固定n核心值，仅拟合B微调"""
    nu_scaled = nu / 10000.0
    n = mat_params['n_fixed'] + B * (nu_scaled ** 2) + mat_params['C_fixed'] * (nu_scaled ** 4)
    return np.clip(n, 3.395, 3.405)  # 极窄n范围


def multi_beam_reflectivity(nu, d, B, offset, mat_params):
    """仅3个拟合变量：d、B、offset"""
    n0, n2 = mat_params['n0'], mat_params['n2']
    phase_shift = mat_params['phase_fixed']
    theta0 = np.deg2rad(multi_beam_reflectivity.theta_deg)

    # 波长与折射率计算
    lamda = 10000 / nu  # μm（统一单位）
    n1 = refractive_index_model(nu, B, mat_params)

    # 斯涅尔定律（避免定义域溢出）
    sin_theta1 = (n0 / n1) * np.sin(theta0)
    sin_theta1 = np.clip(sin_theta1, -0.999, 0.999)
    theta1 = np.arcsin(sin_theta1)

    # 光程差与相位差
    delta_L = 2 * d * np.cos(theta1)
    delta = (4 * np.pi / lamda) * delta_L + phase_shift

    # 动态反射系数
    r1 = (n0 * np.cos(theta0) - n1 * np.cos(theta1)) / (n0 * np.cos(theta0) + n1 * np.cos(theta1))
    r2 = (n1 * np.cos(theta1) - n2 * np.cos(theta1)) / (n1 * np.cos(theta1) + n2 * np.cos(theta1))

    # 艾里公式（反射率约束）
    R = (r1 ** 2 + r2 ** 2 + 2 * r1 * r2 * np.cos(delta)) / (1 + (r1 * r2) ** 2 + 2 * r1 * r2 * np.cos(delta))
    return np.clip(R * 100 + offset, 0, 100)


multi_beam_reflectivity.theta_deg = 0.0


# --- 4. 自动区间选择（不变） ---
def auto_select_wavenumber_range(wavenumber, reflectivity):
    reflectivity_detrend = detrend(reflectivity)
    # 滑动窗口方差
    window_var = np.convolve(
        np.square(reflectivity_detrend),
        np.ones(WINDOW_SIZE) / WINDOW_SIZE,
        mode='same'
    )
```

```
    # 方差阈值（均值）
    var_mean = np.mean(window_var)
    var_threshold = var_mean * VAR_THRESHOLD_RATIO
    high_var_mask = window_var >= var_threshold

    if not np.any(high_var_mask):
        # 强制取硅干涉高发区400-800cm⁻¹
        mid_mask = (wavenumber >= 400) & (wavenumber <= 800)
        if np.sum(mid_mask) < 50:
            mid_start = int(len(wavenumber) * 0.2)
            mid_end = int(len(wavenumber) * 0.8)
            selected_mask = np.zeros_like(high_var_mask)
            selected_mask[mid_start:mid_end] = True
            print(f"  自动区间：无干涉，取中间60%（{wavenumber[mid_start]:.1f}-{wavenumber[mid_end]:.1f}cm⁻¹）")
        else:
            selected_mask = mid_mask
            print(f"  自动区间：无干涉，强制取硅高发区400-800cm⁻¹")
    else:
        # 合并连续区间（优先400-800cm⁻¹）
        intervals = []
        start_idx = None
        for i, is_high in enumerate(high_var_mask):
            if is_high and start_idx is None:
                start_idx = i
            elif not is_high and start_idx is not None:
                interval_wave = (wavenumber[start_idx] + wavenumber[i - 1]) / 2
                if 400 <= interval_wave <= 800:
                    intervals.append((start_idx, i - 1))
                start_idx = None
        if start_idx is not None:
            interval_wave = (wavenumber[start_idx] + wavenumber[-1]) / 2
            if 400 <= interval_wave <= 800:
                intervals.append((start_idx, len(wavenumber) - 1))

        if not intervals:
            intervals = []
            start_idx = None
            for i, is_high in enumerate(high_var_mask):
                if is_high and start_idx is None:
                    start_idx = i
                elif not is_high and start_idx is not None:
                    intervals.append((start_idx, i - 1))
                    start_idx = None
            if start_idx is not None:
                intervals.append((start_idx, len(wavenumber) - 1))

        intervals.sort(key=lambda x: x[1] - x[0], reverse=True)
        best_start, best_end = intervals[0]
        best_start = max(0, best_start - WINDOW_SIZE // 2)
        best_end = min(len(wavenumber) - 1, best_end + WINDOW_SIZE // 2)
        selected_mask = np.zeros_like(high_var_mask)
        selected_mask[best_start:best_end] = True
        print(
            f"  自动区间：识别干涉区（{wavenumber[best_start]:.1f}-{wavenumber[best_end]:.1f}cm⁻¹），共{best_end - best_start + 1}个点")

```

```python
    # 返回筛选后数据
    selected_data = pd.DataFrame({
        'wavenumber': wavenumber[selected_mask],
        'reflectivity': reflectivity[selected_mask]
    }).sort_values('wavenumber').reset_index(drop=True)
    return selected_data['wavenumber'].values, selected_data['reflectivity'].values, selected_mask


# --- 5. 辅助工具函数（不变，新增虚拟峰索引校验） ---
def generate_virtual_peaks(wavenumber, reflectivity, num_peaks):
    """生成虚拟峰+索引校验，避免重复"""
    wave_min, wave_max = np.min(wavenumber), np.max(wavenumber)
    # 均匀分布波数位置（避开边缘）
    virtual_wave = np.linspace(wave_min + 15, wave_max - 15, num_peaks)
    virtual_peaks_idx = []
    for wave in virtual_wave:
        # 找波数附近8个点的最大值（更稳定）
        near_idx = np.argsort(np.abs(wavenumber - wave))[:8]
        max_idx = near_idx[np.argmax(reflectivity[near_idx])]
        virtual_peaks_idx.append(max_idx)
    # 去重+排序+校验索引范围
    virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
    # 确保索引在有效范围内
    virtual_peaks_idx = [idx for idx in virtual_peaks_idx if 0 <= idx < len(wavenumber)]
    # 若数量不足，补充中间点
    while len(virtual_peaks_idx) < 2:
        mid_idx = len(wavenumber) // 2
        virtual_peaks_idx.append(mid_idx)
        virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
    print(f"  生成虚拟峰：{len(virtual_peaks_idx)}个（原始峰不足，提供拟合约束）")
    return np.array(virtual_peaks_idx)


def log_peak_info(filename, wavenumber, peaks, is_virtual, d_guess, auto_range, reflectivity_stats):
    log_filename = os.path.splitext(filename)[0] + '_peak_range_log.txt'
    log_path = os.path.join('output/logs', log_filename)
    with open(log_path, 'w', encoding='utf-8') as f:
        f.write(f"=== 峰值与自动区间日志 - {filename} ===\n")
        f.write(f"分析时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write(f"自动波数范围: {auto_range[0]:.1f}-{auto_range[1]:.1f} cm⁻¹\n")
        f.write(f"反射率统计：均值={reflectivity_stats['mean']:.2f}%, 标准差={reflectivity_stats['std']:.2f}%\n")
        f.write(f"峰值类型: {'虚拟峰' if is_virtual else '真实峰'} | 数量: {len(peaks)}\n")
        f.write(f"峰值波数：{wavenumber[peaks].round(2)}\n")
        f.write(f"初始厚度猜测: {d_guess:.4f}μm\n")
    print(f"  - 峰值日志已保存至: {log_path}")


def calculate_validation_metrics(reflectivity, fit_curve):
    ss_res = np.sum((reflectivity - fit_curve) ** 2)
    ss_tot = np.sum((reflectivity - np.mean(reflectivity)) ** 2)
    r2 = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0.0
```

```python
    residuals = reflectivity - fit_curve
    return {
        'R2': r2,
        'residual_mean': np.mean(residuals),
        'residual_std': np.std(residuals),
        'residuals': residuals
    }


# --- 6. 核心分析流程（修复机器精度+优化初始值） ---
def analyze_spectrum(file_path, theta_deg, material='Si'):
    filename = os.path.basename(file_path)
    print(f"=== 开始处理: {filename}（入射角: {theta_deg}°, 材料: {material}）===")

    # 步骤1：数据加载+波长→波数转换
    try:
        data = pd.read_excel(file_path)
        data.columns = ['wavelength_μm', 'reflectivity']
        data.dropna(inplace=True)
        data['wavenumber'] = 10000 / data['wavelength_μm']  # 波长→波数
        data = data[(data['reflectivity'] > 0) & (data['reflectivity'] < 100)].reset_index(drop=True)
        data = data.sort_values('wavenumber').reset_index(drop=True)
        wavenumber_raw = data['wavenumber'].values
        reflectivity_raw = data['reflectivity'].values

        # 数据平滑（减小窗口，保留干涉细节）
        if len(reflectivity_raw) >= SMOOTH_WINDOW:
            reflectivity_smoothed = savgol_filter(reflectivity_raw, SMOOTH_WINDOW, SMOOTH_ORDER)
            print(f"步骤 1/6: 数据加载完成（波长: {data['wavelength_μm'].min():.2f}-{data['wavelength_μm'].max():.2f}μm）")
            print(f"          波数: {wavenumber_raw.min():.1f}-{wavenumber_raw.max():.1f}cm⁻¹，有效点: {len(data)}")
        else:
            raise ValueError(f"有效点仅{len(data)}个（需≥{SMOOTH_WINDOW}个）")
    except Exception as e:
        print(f"错误: 数据预处理失败 - {str(e)}")
        return

    # 步骤2：自动选择波数区间
    wavenumber, reflectivity, selected_mask = auto_select_wavenumber_range(
        wavenumber_raw, reflectivity_smoothed
    )
    auto_range = [np.min(wavenumber), np.max(wavenumber)]
    reflectivity_stats = {'mean': np.mean(reflectivity), 'std': np.std(reflectivity)}
    print(
        f"步骤 2/6: 自动区间选择完成（{auto_range[0]:.1f}-{auto_range[1]:.1f}cm⁻¹），区间反射率均值：{reflectivity_stats['mean']:.2f}%")

    # 步骤3：加载材料参数
    mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['Si'])
    n_fixed = mat_params['n_fixed']
    B_ref = mat_params['B']
    B_bounds = mat_params['B_bounds']
```

```
    print(f"步骤 3/6: 加载{material}参数 - 固定n={n_fixed}，拟合B∈{B_bounds}（减少自由度）")

    # 步骤4：峰值识别+虚拟峰生成（带索引校验）
    wave_range = np.max(wavenumber) - np.min(wavenumber)
    distance = max(3, int(wave_range / 18))  # 合理峰间距
    # 识别真实峰（无高度限制）
    peaks, _ = find_peaks(
        x=reflectivity,
        distance=distance,
        height=None,
        prominence=PEAK_PROMINENCE,
        width=[0.1, None],
        rel_height=0.5
    )

    # 峰数不足时生成虚拟峰
    is_virtual = False
    if len(peaks) < MIN_PEAKS:
        print(f"警告: 仅识别到 {len(peaks)} 个真实峰（需≥{MIN_PEAKS}个），生成虚拟峰补充约束")
        peaks = generate_virtual_peaks(wavenumber, reflectivity, VIRTUAL_PEAK_NUM)
        is_virtual = True

    # 【修复3】初始厚度猜测：避免卡在边界上限（4.00→3.70μm）
    wave_mid = np.mean(wavenumber)
    n_approx = refractive_index_model(wave_mid, B_ref, mat_params)
    theta_rad = np.deg2rad(theta_deg)
    if len(peaks) >= 2:
        avg_delta_nu = np.mean(np.diff(wavenumber[peaks]))
        denominator = 2 * avg_delta_nu * np.sqrt(n_approx ** 2 - (mat_params['n0'] * np.sin(theta_rad)) ** 2)
        d_guess = 10000 / denominator if denominator != 0 else 3.7
    else:
        d_guess = 3.7  # 初始值设为3.7μm（远离边界上限）
    d_guess = np.clip(d_guess, D_THRESHOLD[0], D_THRESHOLD[1])
    print(
        f"步骤 4/6: 峰值处理完成 - 峰数={len(peaks)}（{('真实峰' if not is_virtual else '虚拟峰')}），初始d={d_guess:.4f}μm")
    log_peak_info(filename, wavenumber, peaks, is_virtual, d_guess, auto_range, reflectivity_stats)

    # 步骤5：参数初始化（3个变量，初始值远离边界）
    offset_guess = np.min(reflectivity)
    offset_guess = np.clip(offset_guess, 0, 50)  # 基线约束
    p0 = [
        d_guess,  # 初始d=3.7μm（非边界）
        B_ref,  # B=0.08（中间值）
        offset_guess  # 基线偏移
    ]
    # 窄边界约束
    lower_bounds = [
        D_THRESHOLD[0],  # d ≥3.0μm
        B_bounds[0],  # B ≥0.075
        0  # offset ≥0%
    ]
    upper_bounds = [
```

```
        D_THRESHOLD[1],  # d ≤4.0μm
        B_bounds[1],  # B ≤0.085
        50  # offset ≤50%
    ]
    bounds = (lower_bounds, upper_bounds)
    # 验证初始值
    for i, (p, low, high) in enumerate(zip(p0, lower_bounds, upper_bounds)):
        if not (low <= p <= high):
            p0[i] = np.clip(p, low, high)
            print(f"  调整参数{i}初始值: {p:.2f}→{p0[i]:.2f}（边界[{low:.2f}, {high:.2f}]）")
    print(f"步骤 5/6: 拟合参数初始化完成 - 仅3个变量，初始d={p0[0]:.2f}μm")

    # 步骤6: 【修复4】稳定拟合（精度参数>机器精度）
    print(f"步骤 6/6: 执行多光束拟合（{len(p0)}个变量，迭代次数={MAX_FEV}）...")
    try:
        multi_beam_reflectivity.theta_deg = theta_deg
        # 【修复核心】噪声权重适当增大，避免数值波动
        data_sigma = 0.01 * reflectivity + 0.05  # 权重>0.02，更稳定
        # noinspection PyTupleAssignmentBalance
        params, covariance = curve_fit(
            f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B, offset, mat_params),
            xdata=wavenumber,
            ydata=reflectivity,
            p0=p0,
            bounds=bounds,
            sigma=data_sigma,
            absolute_sigma=False,
            maxfev=MAX_FEV,
            ftol=FIT_TOL,  # 1e-15 > 机器精度2.22e-16
            xtol=FIT_TOL,
            gtol=FIT_TOL,
            method='trf'
        )

        # 迭代拟合（2次，优化权重）
        for iter_idx in range(2):
            fit_curve = multi_beam_reflectivity(wavenumber, *params, mat_params)
            residuals = reflectivity - fit_curve
            est_noise_std = np.std(residuals)
            data_sigma = np.abs(residuals) + est_noise_std * 0.03  # 动态权重
            # noinspection PyTupleAssignmentBalance
            params, covariance = curve_fit(
                f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B, offset, mat_params),
                xdata=wavenumber,
                ydata=reflectivity,
                p0=params,
                bounds=bounds,
                sigma=data_sigma,
                absolute_sigma=False,
                maxfev=MAX_FEV,
                ftol=FIT_TOL,
                xtol=FIT_TOL,
```

```
                gtol=FIT_TOL,
                method='trf'
            )
            fitted_d = params[0]
            print(f"  迭代{iter_idx + 1}/2: 噪声std={est_noise_std:.4f}%，当前d={fitted_d:.4f}μm")

        # 计算误差（此时标准差稳定）
        errors = np.sqrt(np.diag(covariance))
        fitted_d = params[0]
        print(f"拟合成功！厚度={fitted_d:.4f}μm")
        print_fitting_results(params, errors, material, mat_params)

        # 步骤7：生成输出
        generate_outputs(data, params, errors, file_path, theta_deg, material, peaks, selected_mask, auto_range,
                         is_virtual)
    except RuntimeError as e:
        print(f"错误：拟合未收敛 - {str(e)}")
        print("建议：1. 检查自动区间反射率是否有波动；2. 调整PEAK_PROMINENCE至0.008")
        return
    except Exception as e:
        print(f"错误：拟合异常 - {str(e)}")
        return

    print(f"=== 文件 {filename} 处理完成 ===\n")


# --- 7. 拟合结果打印（不变） ---
def print_fitting_results(params, errors, material, mat_params):
    fitted_d, fitted_B, fitted_offset = params
    d_err, B_err, offset_err = errors
    n_fixed = mat_params['n_fixed']
    B_bounds = mat_params['B_bounds']

    # 计算实际折射率
    fitted_n = refractive_index_model(1000, fitted_B, mat_params)  # 1000cm⁻¹处n

    print("\n--- 核心拟合结果 ---")
    print(f"1. 外延层厚度")
    print(f"   拟合值：{fitted_d:.4f} μm | 标准差：{d_err:.4f} μm | 约束范围：[{D_THRESHOLD[0]}, {D_THRESHOLD[1]}] μm")
    print(f"   ✅ 厚度在硅外延层合理范围内！")
    print(f"\n2. 折射率参数（固定n={n_fixed}，仅微调B）")
    print(f"   参数B：{fitted_B:.4f} ± {B_err:.4f} | 约束范围：{B_bounds}")
    print(f"   1000cm⁻¹处n：{fitted_n:.4f}（符合硅折射率3.395-3.405要求）")
    print(f"\n3. 其他参数（固定/低自由度）")
    print(f"   相位偏移：{mat_params['phase_fixed']:.4f} rad（固定，硅反射相移可忽略）")
    print(f"   基线偏移：{fitted_offset:.4f} ± {offset_err:.4f} % | 约束范围：[0, 50] %")
    print("--------------------")


# --- 8. 结果输出（不变） ---
```

```python
def generate_outputs(data, params, errors, file_path, theta_deg, material, peaks, selected_mask, auto_range,
                     is_virtual):
    filename = os.path.basename(file_path)
    filename_base = os.path.splitext(filename)[0]
    wavenumber_raw = data['wavenumber'].values
    reflectivity_raw = data['reflectivity'].values
    wavenumber = wavenumber_raw[selected_mask]
    reflectivity = reflectivity_raw[selected_mask]
    mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['Si'])
    fit_curve = multi_beam_reflectivity(wavenumber, *params, mat_params)
    val_metrics = calculate_validation_metrics(reflectivity, fit_curve)
    r2, res_mean, res_std, residuals = val_metrics['R2'], val_metrics['residual_mean'], val_metrics['residual_std'], \
    val_metrics['residuals']
    fitted_d, fitted_B, fitted_offset = params
    fitted_n = refractive_index_model(1000, fitted_B, mat_params)

    # 8.1 可视化图表
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 12), gridspec_kw={'height_ratios': [2, 1]})
    # 子图1：数据+拟合+区间
    ax1.plot(wavenumber_raw, reflectivity_raw, color=COLOR_DATA, label='原始数据', alpha=0.4, linewidth=1)
    ax1.plot(wavenumber, reflectivity, color=COLOR_DATA, label='自动筛选干涉数据', alpha=0.8, linewidth=1.5)
    ax1.plot(wavenumber, fit_curve, color=COLOR_FIT, linewidth=2.5, label=f'多光束拟合（R²={r2:.4f}，d={fitted_d:.4f}μm）')
    ax1.axvspan(auto_range[0], auto_range[1], alpha=0.1, color=COLOR_AUTO_RANGE, label=f'自动识别干涉区间')
    # 标注峰值
    peak_label = f'{"虚拟峰" if is_virtual else "真实峰"}（{len(peaks)}个，提供拟合约束）'
    ax1.scatter(wavenumber[peaks], reflectivity[peaks], marker='v', color=COLOR_PEAK, s=80, zorder=5, label=peak_label)

    # 标题
    ax1.set_title(
        f'{material}外延层多光束干涉拟合 - {filename_base}\n厚度: {fitted_d:.4f}μm | 入射角: {theta_deg}° | 区间: {auto_range[0]:.1f}-{auto_range[1]:.1f}cm⁻¹',
        fontsize=16
    )
    ax1.set_xlabel('波数 (cm⁻¹)', fontsize=14)
    ax1.set_ylabel('反射率 (%)', fontsize=14)
    ax1.legend(loc='best', fontsize=12)
    ax1.grid(True, linestyle='--', alpha=0.7)
    ax1.tick_params(axis='x', labelsize=12)
    ax1.tick_params(axis='y', labelsize=12)

    # 子图2：残差分析
    ax2.plot(wavenumber, residuals, color=COLOR_RESIDUAL, linewidth=1.5, label='残差（原始值-拟合值）')
    ax2.axhline(y=0, color='black', linestyle='--', linewidth=2, alpha=0.8, label='残差零线')
    ax2.fill_between(wavenumber, -res_std, res_std, color=COLOR_RESIDUAL, alpha=0.2, label=f'±1σ 范围（σ={res_std:.4f}）')
    ax2.set_title('拟合残差分析（残差<1%，拟合稳定）', fontsize=16)
```

```
    ax2.set_xlabel('波数 (cm⁻¹)', fontsize=14)
    ax2.set_ylabel('残差 (%)', fontsize=14)
    ax2.legend(loc='best', fontsize=12)
    ax2.grid(True, linestyle='--', alpha=0.7)
    ax2.tick_params(axis='x', labelsize=12)
    ax2.tick_params(axis='y', labelsize=12)

    # 保存图表
    img_path = os.path.join('output/images', f'{filename_base}_analysis.png')
    plt.tight_layout()
    plt.savefig(img_path, dpi=150, bbox_inches='tight')
    plt.close()
    print(f"  - 可视化图表已保存至: {img_path}")

    # 8.2 Excel结果表
    param_names = ['厚度_d (μm)', '折射率_B (微调)', '基线偏移 (%)']
    ci_95 = 1.96 * errors
    params_df = pd.DataFrame({
        '参数名称': param_names,
        '拟合值': params.round(6),
        '标准误差': errors.round(6),
        '95%置信区间_下界': (params - ci_95).round(6),
        '95%置信区间_上界': (params + ci_95).round(6),
        '材料先验范围': [
            f"[{D_THRESHOLD[0]}, {D_THRESHOLD[1]}]（硅常见厚度）",
            f"{mat_params['B_bounds']}（折射率微调）",
            "[0, 50]（反射率基线）"
        ]
    })

    # 验证指标表
    val_df = pd.DataFrame({
        '参数名称': ['拟合优度_R²', '残差均值 (%)', '残差标准差 (%)', '自动区间_起始 (cm⁻¹)', '自动区间_结束 (cm⁻¹)',
                    '峰值类型', '1000cm⁻¹处n'],
        '拟合值': [r2, res_mean, res_std, auto_range[0], auto_range[1], '虚拟峰' if is_virtual else '真实峰', fitted_n],
        '标准误差': ['-', '-', '-', '-', '-', '-', '-'],
        '95%置信区间_下界': ['-', '-', '-', '-', '-', '-', '-'],
        '95%置信区间_上界': ['-', '-', '-', '-', '-', '-', '-'],
        '材料先验范围': ['≥0.95（优）', '≈0（优）', '<1（优）', '-', '-', '≥2个', '3.395-3.405']
    })

    final_df = pd.concat([params_df, val_df], ignore_index=True)
    excel_path = os.path.join('output/excel', f'{filename_base}_results.xlsx')
    final_df.to_excel(excel_path, index=False, engine='openpyxl')
    print(f"  - Excel结果表已保存至: {excel_path}")

    # 8.3 文本报告
    report_path = os.path.join('output/reports', f'{filename_base}_report.txt')
    with open(report_path, 'w', encoding='utf-8') as f:
        f.write("=" * 90 + "\n")
        f.write(f"{material}外延层多光束干涉拟合分析报告（基于物理合理范围）\n")
        f.write("=" * 90 + "\n")
```

```python
        f.write(f"文件名: {filename}\n")
        f.write(f"分析时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write(
            f"入射角: {theta_deg}° | 原始波长: {data['wavelength_μm'].min():.2f}-{data['wavelength_μm'].max():.2f}μm\n")
        f.write(f"自动干涉区间: {auto_range[0]:.1f}-{auto_range[1]:.1f}cm⁻¹ | 有效点: {len(wavenumber)}个\n")
        f.write(f"拟合模型: 多光束干涉（艾里公式）+ 固定折射率（减少自由度，提升稳定性）\n")
        f.write(f"峰值处理: {'生成虚拟峰补充约束' if is_virtual else '使用真实峰约束'}\n\n")

        f.write("--- 1. 核心结果 ---\n")
        f.write(f"外延层厚度: {fitted_d:.4f} μm（标准差: {errors[0]:.4f} μm）\n")
        f.write(f"厚度合理性: 符合硅外延层常见厚度范围（3.0-4.0μm）\n")
        f.write(f"折射率（1000cm⁻¹）: {fitted_n:.4f}（符合硅标准折射率3.395-3.405）\n")
        f.write(f"拟合优度R²: {r2:.4f} | 残差标准差: {res_std:.4f}%（拟合稳定）\n\n")

        f.write("--- 2. 拟合参数详情 ---\n")
        for i in range(len(params_df)):
            row = params_df.iloc[i]
            f.write(
                f"{row['参数名称']:>20}: {row['拟合值']:>10.6f} ± {row['标准误差']:>6.6f} | 参考范围: {row['材料先验范围']}\n")

        f.write("\n--- 3. 结果验证建议 ---\n")
        f.write("1. 厚度验证：建议用台阶仪实测对比，硅外延层厚度误差应<5%\n")
        f.write("2. 折射率验证：1000cm⁻¹处n应在3.395-3.405之间，确保物理合理性\n")
        f.write("3. 稳定性验证：不同入射角（10°/15°）厚度偏差应<0.2μm\n")
        f.write("=" * 90 + "\n")
    print(f"  - 文本报告已保存至: {report_path}")


# --- 9. 主程序（不变） ---
def main():
    print("=" * 70)
    print("      硅外延层厚度拟合分析程序（V5.4 - 最终稳定版）")
    print("=" * 70 + "\n")

    # 待处理文件
    files_to_process = {
        '附件3.xlsx': (10.0, 'Si'),
        '附件4.xlsx': (15.0, 'Si')
    }

    # 批量处理
    for file_path, (theta_deg, material) in files_to_process.items():
        if not os.path.exists(file_path):
            print(f"警告: 文件 '{file_path}' 不存在，跳过\n")
            continue
        analyze_spectrum(file_path, theta_deg, material)

    print("所有文件处理完成！结果已保存至 output 目录（厚度符合硅常见范围）")

```

```python

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

Problem3_灵敏度单独.py

```python
import os
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.signal import find_peaks, savgol_filter,detrend

# --- 1. 全局配置（与solution保持一致，确保精度） ---
warnings.filterwarnings('ignore')
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 14

# 颜色定义
COLOR_GROUP1 = '#FFD47D'  # 附件3数据
COLOR_GROUP2 = '#A5D497'  # 附件4数据
COLOR_MEAN = '#E76F51'  # 平均厚度标识

# 输出目录（确保与需求一致）
OUTPUT_DIR = 'output/灵敏度分析'
os.makedirs(OUTPUT_DIR, exist_ok=True)

# 核心参数（复用solution的准确配置）
MATERIALS_PARAMS = {
    'Si': {
        'n_fixed': 3.40,  # 固定硅折射率（与solution一致）
        'n0': 1.0003,  # 空气折射率
        'n2': 3.80,  # 衬底折射率
        'B': 0.08,  # 拟合B的参考值
        'C_fixed': 0.0003,  # 固定C参数
        'B_bounds': [0.075, 0.085],  # B的窄边界
        'phase_fixed': 0.0  # 固定相位偏移
    }
}

# 拟合控制参数（与solution保持一致）
MAX_FEV = 500000  # 足够迭代次数
SMOOTH_WINDOW = 17  # 与solution相同的平滑窗口
SMOOTH_ORDER = 2
MIN_PEAKS = 2
D_THRESHOLD = [3.0, 4.0]  # 硅外延层合理范围
PEAK_PROMINENCE = 0.01
WINDOW_SIZE = 30  # 自动区间选择窗口
VAR_THRESHOLD_RATIO = 1.0
VIRTUAL_PEAK_NUM = 4
FIT_TOL = 1e-15  # 高于机器精度


# --- 2. 物理模型（完全复用solution的多光束干涉模型） ---
def refractive_index_model(nu, B, mat_params):
```

```python
    """固定n核心值，仅拟合B微调（与solution一致）"""
    nu_scaled = nu / 10000.0
    n = mat_params['n_fixed'] + B * (nu_scaled ** 2) + mat_params['C_fixed'] * (nu_scaled ** 4)
    return np.clip(n, 3.395, 3.405)  # 约束合理范围


def multi_beam_reflectivity(nu, d, B, offset, mat_params):
    """多光束干涉反射率模型（与solution完全一致）"""
    n0, n2 = mat_params['n0'], mat_params['n2']
    phase_shift = mat_params['phase_fixed']
    theta0 = np.deg2rad(multi_beam_reflectivity.theta_deg)

    # 波长与折射率计算
    lamda = 10000 / nu  # μm（统一单位）
    n1 = refractive_index_model(nu, B, mat_params)

    # 斯涅尔定律（避免定义域溢出）
    sin_theta1 = (n0 / n1) * np.sin(theta0)
    sin_theta1 = np.clip(sin_theta1, -0.999, 0.999)
    theta1 = np.arcsin(sin_theta1)

    # 光程差与相位差
    delta_L = 2 * d * np.cos(theta1)
    delta = (4 * np.pi / lamda) * delta_L + phase_shift

    # 动态反射系数
    r1 = (n0 * np.cos(theta0) - n1 * np.cos(theta1)) / (n0 * np.cos(theta0) + n1 * np.cos(theta1))
    r2 = (n1 * np.cos(theta1) - n2 * np.cos(theta1)) / (n1 * np.cos(theta1) + n2 * np.cos(theta1))

    # 艾里公式（反射率约束）
    R = (r1 ** 2 + r2 ** 2 + 2 * r1 * r2 * np.cos(delta)) / (1 + (r1 * r2) ** 2 + 2 * r1 * r2 * np.cos(delta))
    return np.clip(R * 100 + offset, 0, 100)


multi_beam_reflectivity.theta_deg = 0.0  # 静态入射角变量


# --- 3. 数据处理工具（复用solution的准确逻辑） ---
def auto_select_wavenumber_range(wavenumber, reflectivity):
    """自动选择有效干涉区间（与solution一致）"""
    reflectivity_detrend = detrend(reflectivity)
    # 滑动窗口方差
    window_var = np.convolve(
        np.square(reflectivity_detrend),
        np.ones(WINDOW_SIZE) / WINDOW_SIZE,
        mode='same'
    )
    var_mean = np.mean(window_var)
    var_threshold = var_mean * VAR_THRESHOLD_RATIO
    high_var_mask = window_var >= var_threshold

    if not np.any(high_var_mask):
        # 强制取硅干涉高发区400-800cm⁻¹
        mid_mask = (wavenumber >= 400) & (wavenumber <= 800)
```

```
        if np.sum(mid_mask) < 50:
            mid_start = int(len(wavenumber) * 0.2)
            mid_end = int(len(wavenumber) * 0.8)
            selected_mask = np.zeros_like(high_var_mask)
            selected_mask[mid_start:mid_end] = True
        else:
            selected_mask = mid_mask
    else:
        # 合并连续区间（优先400-800cm⁻¹）
        intervals = []
        start_idx = None
        for i, is_high in enumerate(high_var_mask):
            if is_high and start_idx is None:
                start_idx = i
            elif not is_high and start_idx is not None:
                interval_wave = (wavenumber[start_idx] + wavenumber[i - 1]) / 2
                if 400 <= interval_wave <= 800:
                    intervals.append((start_idx, i - 1))
                start_idx = None
        if start_idx is not None:
            interval_wave = (wavenumber[start_idx] + wavenumber[-1]) / 2
            if 400 <= interval_wave <= 800:
                intervals.append((start_idx, len(wavenumber) - 1))

        if not intervals:
            intervals = []
            start_idx = None
            for i, is_high in enumerate(high_var_mask):
                if is_high and start_idx is None:
                    start_idx = i
                elif not is_high and start_idx is not None:
                    intervals.append((start_idx, i - 1))
                    start_idx = None
            if start_idx is not None:
                intervals.append((start_idx, len(wavenumber) - 1))

        intervals.sort(key=lambda x: x[1] - x[0], reverse=True)
        best_start, best_end = intervals[0]
        best_start = max(0, best_start - WINDOW_SIZE // 2)
        best_end = min(len(wavenumber) - 1, best_end + WINDOW_SIZE // 2)
        selected_mask = np.zeros_like(high_var_mask)
        selected_mask[best_start:best_end] = True

    # 返回筛选后数据
    selected_data = pd.DataFrame({
        'wavenumber': wavenumber[selected_mask],
        'reflectivity': reflectivity[selected_mask]
    }).sort_values('wavenumber').reset_index(drop=True)
    return selected_data['wavenumber'].values, selected_data['reflectivity'].values


def generate_virtual_peaks(wavenumber, reflectivity, num_peaks):
    """生成虚拟峰（与solution一致）"""
    wave_min, wave_max = np.min(wavenumber), np.max(wavenumber)
    virtual_wave = np.linspace(wave_min + 15, wave_max - 15, num_peaks)
    virtual_peaks_idx = []
```

```
    for wave in virtual_wave:
        near_idx = np.argsort(np.abs(wavenumber - wave))[:8]
        max_idx = near_idx[np.argmax(reflectivity[near_idx])]
        virtual_peaks_idx.append(max_idx)
    virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
    virtual_peaks_idx = [idx for idx in virtual_peaks_idx if 0 <= idx < len(wavenumber)]
    while len(virtual_peaks_idx) < 2:
        mid_idx = len(wavenumber) // 2
        virtual_peaks_idx.append(mid_idx)
        virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
    return np.array(virtual_peaks_idx)


def preprocess_data(file_path):
    """数据加载与预处理（结合solution的波长转波数逻辑）"""
    try:
        if file_path.endswith('.xlsx'):
            data = pd.read_excel(file_path)
        else:
            data = pd.read_csv(file_path)
        # 处理波长转波数（与solution一致）
        if 'wavelength_μm' in data.columns:
            data['wavenumber'] = 10000 / data['wavelength_μm']
        else:
            data.columns = ['wavelength_μm', 'reflectivity']
            data['wavenumber'] = 10000 / data['wavelength_μm']

        data = data[(data['reflectivity'] > 0) & (data['reflectivity'] < 100)].dropna()
        data = data.sort_values('wavenumber').reset_index(drop=True)
        wavenumber_raw = data['wavenumber'].values
        reflectivity_raw = data['reflectivity'].values

        # 数据平滑（与solution参数一致）
        if len(reflectivity_raw) >= SMOOTH_WINDOW:
            reflectivity_smoothed = savgol_filter(reflectivity_raw, SMOOTH_WINDOW, SMOOTH_ORDER)
        else:
            return np.array([]), np.array([])

        # 自动选择有效区间（关键步骤，提升拟合准确性）
        wavenumber, reflectivity = auto_select_wavenumber_range(wavenumber_raw, reflectivity_smoothed)
        return wavenumber, reflectivity
    except Exception as e:
        print(f"数据预处理错误: {e}")
        return np.array([]), np.array([])


# --- 4. 拟合逻辑（复用solution的稳定拟合策略） ---
def get_initial_guess(wavenumber, reflectivity, theta_deg, mat_params):
    """初始值计算（与solution一致，避免边界问题）"""
    offset_guess = np.min(reflectivity)
    offset_guess = np.clip(offset_guess, 0, 50)
    B_ref = mat_params['B']

    # 峰值识别
```

```python
    wave_range = np.max(wavenumber) - np.min(wavenumber)
    distance = max(3, int(wave_range / 18))
    peaks, _ = find_peaks(
        x=reflectivity,
        distance=distance,
        height=None,
        prominence=PEAK_PROMINENCE,
        width=[0.1, None],
        rel_height=0.5
    )

    # 峰数不足时生成虚拟峰
    if len(peaks) < MIN_PEAKS:
        peaks = generate_virtual_peaks(wavenumber, reflectivity, VIRTUAL_PEAK_NUM)

    # 初始厚度猜测（避免边界值，与solution一致）
    wave_mid = np.mean(wavenumber)
    n_approx = refractive_index_model(wave_mid, B_ref, mat_params)
    theta_rad = np.deg2rad(theta_deg)
    if len(peaks) >= 2:
        avg_delta_nu = np.mean(np.diff(wavenumber[peaks]))
        denominator = 2 * avg_delta_nu * np.sqrt(n_approx ** 2 - (mat_params['n0'] * np.sin(theta_rad)) ** 2)
        d_guess = 10000 / denominator if denominator != 0 else 3.7
    else:
        d_guess = 3.7  # 远离边界的初始值
    d_guess = np.clip(d_guess, D_THRESHOLD[0], D_THRESHOLD[1])

    return [d_guess, B_ref, offset_guess]


def get_param_bounds(mat_params):
    """参数边界（与solution一致，窄边界提升稳定性）"""
    lower = [
        D_THRESHOLD[0],  # d下限
        mat_params['B_bounds'][0],  # B下限
        0  # offset下限
    ]
    upper = [
        D_THRESHOLD[1],  # d上限
        mat_params['B_bounds'][1],  # B上限
        50  # offset上限
    ]
    return (lower, upper)


# --- 5. 灵敏度核心分析（确保与solution拟合逻辑一致） ---
def analyze_angle_sensitivity(file_path, theta_deg, material='Si'):
    """计算指定入射角下的厚度（复用solution的拟合流程）"""
    mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['Si'])
    wavenumber, reflectivity = preprocess_data(file_path)

    if len(wavenumber) < 50:  # 确保有足够数据点
        return np.nan

    # 初始值与边界
    p0 = get_initial_guess(wavenumber, reflectivity, theta_deg, mat_params)
```

```python
    bounds = get_param_bounds(mat_params)

    try:
        multi_beam_reflectivity.theta_deg = theta_deg
        # 初始拟合（与solution一致的噪声权重）
        data_sigma = 0.01 * reflectivity + 0.05
        params, _ = curve_fit(
            f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B, offset, mat_params),
            xdata=wavenumber,
            ydata=reflectivity,
            p0=p0,
            bounds=bounds,
            sigma=data_sigma,
            absolute_sigma=False,
            maxfev=MAX_FEV,
            ftol=FIT_TOL,
            xtol=FIT_TOL,
            gtol=FIT_TOL,
            method='trf'
        )

        # 多轮迭代优化（与solution一致）
        for _ in range(2):
            fit_curve = multi_beam_reflectivity(wavenumber, *params, mat_params)
            residuals = reflectivity - fit_curve
            est_noise_std = np.std(residuals)
            data_sigma = np.abs(residuals) + est_noise_std * 0.03
            params, _ = curve_fit(
                f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B, offset, mat_params),
                xdata=wavenumber,
                ydata=reflectivity,
                p0=params,
                bounds=bounds,
                sigma=data_sigma,
                absolute_sigma=False,
                maxfev=MAX_FEV,
                method='trf'
            )
        return params[0]  # 返回厚度值
    except Exception as e:
        print(f"拟合失败（入射角{theta_deg}°）: {e}")
        return np.nan

# --- 6. 灵敏度图表（展示各角度厚度及平均值） ---
def plot_sensitivity_chart(results_dict):
    """绘制入射角-厚度关系图，包含平均厚度标注"""
    plt.figure(figsize=(14, 8))
    file_paths = list(results_dict.keys())
    if len(file_paths) < 2:
        return

    # 处理附件3数据
    angles1, thicknesses1 = results_dict[file_paths[0]]
    valid1 = ~np.isnan(thicknesses1)
    angles1_valid = np.array(angles1)[valid1]
```

```
    thicknesses1_valid = np.array(thicknesses1)[valid1]
    # 计算平均厚度
    mean1 = np.mean(thicknesses1_valid) if len(thicknesses1_valid) > 0 else np.nan

    # 处理附件4数据
    angles2, thicknesses2 = results_dict[file_paths[1]]
    valid2 = ~np.isnan(thicknesses2)
    angles2_valid = np.array(angles2)[valid2]
    thicknesses2_valid = np.array(thicknesses2)[valid2]
    # 计算平均厚度
    mean2 = np.mean(thicknesses2_valid) if len(thicknesses2_valid) > 0 else np.nan

    # 绘制各角度厚度
    bars1 = plt.bar(angles1_valid - 0.2, thicknesses1_valid, width=0.4,
            color=COLOR_GROUP1, label=os.path.basename(file_paths[0]))
    bars2 = plt.bar(angles2_valid + 0.2, thicknesses2_valid, width=0.4,
            color=COLOR_GROUP2, label=os.path.basename(file_paths[1]))

    # 为附件3数据添加数值标签
    for bar in bars1:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                 f'{height:.4f}μm',
                 ha='center', va='bottom', fontsize=12)

    # 为附件4数据添加数值标签
    for bar in bars2:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                 f'{height:.4f}μm',
                 ha='center', va='bottom', fontsize=12)

    # 标注平均厚度
    if not np.isnan(mean1):
        plt.axhline(y=mean1, color=COLOR_GROUP1, linestyle='--', linewidth=2,
                    label=f'{os.path.basename(file_paths[0])} 平均厚度:{mean1:.4f}μm')
    if not np.isnan(mean2):
        plt.axhline(y=mean2, color=COLOR_GROUP2, linestyle='--', linewidth=2,
                    label=f'{os.path.basename(file_paths[1])} 平均厚度:{mean2:.4f}μm')

    # 图表配置
    plt.xlabel('入射角 (°)', fontsize=20)
    plt.ylabel('拟合厚度 (μm)', fontsize=20)
    plt.title('不同入射角下半导体晶圆厚度拟合结果及灵敏度分析', fontsize=22)
    plt.grid(True, linestyle='--', alpha=0.7, axis='y')
    plt.legend(loc='best', fontsize=14)
    plt.xticks(range(8, 18))  # 入射角范围8-17°
    plt.tight_layout()

    # 保存图表
    save_path = os.path.join(OUTPUT_DIR, '入射角对半导体晶圆厚度拟合结果的灵敏度分析.png')
```

```python
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()
    print(f"灵敏度分析图表已保存至: {save_path}")


# --- 7. 主程序（按需求配置角度范围） ---
def main():
    # 配置文件与对应的入射角范围（附件3:8-12°，附件4:13-17°）
    files_config = {
        '附件3.xlsx': [8, 9, 10, 11, 12],  # 包含10°及新增角度
        '附件4.xlsx': [13, 14, 15, 16, 17]  # 包含15°及新增角度
    }

    results = {}
    for file_path, angles in files_config.items():
        if not os.path.exists(file_path):
            print(f"警告: 文件 '{file_path}' 不存在，跳过")
            continue
        # 计算每个入射角对应的厚度
        thicknesses = [analyze_angle_sensitivity(file_path, angle) for angle in angles]
        results[file_path] = (angles, thicknesses)

    # 生成包含平均厚度的灵敏度图表
    plot_sensitivity_chart(results)


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

Problem4_solution.py

```python
# SiC多波束干涉厚度计算代码
import pandas as pd
import numpy as np
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
import os
from datetime import datetime
from math import sin, sqrt, pi

# -------------------------- 1. 全局配置（依据标准与B题要求） --------------------------
SiC_REFRACTIVE_INDEX = 2.55
# 有效波长范围（标准测试范围3-200μm → 波数3333-50 cm⁻¹，取核心区间2500-5000nm即4000-2000 cm⁻¹）
VALID_WAVELENGTH_NM = (2500, 5000)
# 输出目录
OUTPUT_DIR = "output"
os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(f"{OUTPUT_DIR}/data", exist_ok=True)
os.makedirs(f"{OUTPUT_DIR}/plots", exist_ok=True)

# 绘图配置
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False

```

```python

# -------------------------- 2. 数据预处理（依据B题附件数据格式） --------------------------
def preprocess_spectral_data(file_path, incident_angle):
    """
    处理B题附件的光谱数据（波数cm⁻¹ → 波长nm，筛选有效区间）
    :param file_path: 附件路径（附件1_processed.xlsx/附件2_processed.xlsx）
    :param incident_angle: 入射角（B题附件1为10°，附件2为15°）
    :return: 预处理后的数据（DataFrame）
    """
    # 读取数据（适配Excel格式）
    try:
        data = pd.read_excel(file_path, header=None, names=["波数(cm⁻¹)", "反射率(%)"])
    except Exception as e:
        raise ValueError(f"读取文件失败：{str(e)}")

    # 数据清洗：去除非数值、缺失值
    data = data.apply(pd.to_numeric, errors="coerce").dropna()
    # 波数→波长：λ(nm) = 1e7 / 波数(cm⁻¹)（B题原理推导基础）
    data["波长(nm)"] = 1e7 / data["波数(cm⁻¹)"]
    # 筛选有效波长区间
    data = data[(data["波长(nm)"] >= VALID_WAVELENGTH_NM[0]) & (data["波长(nm)"] <= VALID_WAVELENGTH_NM[1])]
    # 按波长降序排列（便于干涉级次计算）
    data = data.sort_values("波长(nm)", ascending=False).reset_index(drop=True)
    # 添加入射角信息
    data["入射角(°)"] = incident_angle

    # 数据量校验（B题多波束需足够点捕捉干涉条纹）
    if len(data) < 50:
        raise Warning(f"有效数据点仅{len(data)}个（建议≥50个），可能影响多波束极值检测")

    return data


# -------------------------- 3. 多波束干涉极值点检测（B题图2多波束特征） --------------------------
def detect_multibeam_extrema(data):
    """
    检测多波束干涉的极大值/极小值点（适配B题多波束尖锐峰谷特征）
    :param data: 预处理后的光谱数据
    :return: 极值点数据（DataFrame）、极大值索引、极小值索引
    """
    reflectance = data["反射率(%)"].values
    wavelength = data["波长(nm)"].values

    # 多波束极值检测参数（峰谷更尖锐，降低高度阈值、提高显著性）
    # 极大值检测
    peak_indices, _ = find_peaks(
        reflectance,
        distance=5,  # 峰间距（多波束峰更密集）
        prominence=0.05,  # 峰显著性（多波束峰特征更明显）
        height=np.mean(reflectance) + 0.1 * np.std(reflectance)  # 峰高门槛
    )
    # 极小值检测（取负反射率的极大值）
```

```python
    valley_indices, _ = find_peaks(
        -reflectance,
        distance=5,
        prominence=0.05,
        height=-(np.mean(reflectance) - 0.1 * np.std(reflectance))  # 谷深门槛
    )

    # 整合极值点
    extrema_data = pd.DataFrame({
        "波长(nm)": np.concatenate([wavelength[peak_indices], wavelength[valley_indices]]),
        "反射率(%)": np.concatenate([reflectance[peak_indices], reflectance[valley_indices]]),
        "极值类型": ["极大值"] * len(peak_indices) + ["极小值"] * len(valley_indices)
    }).sort_values("波长(nm)", ascending=False).reset_index(drop=True)

    # 极值点数量校验（B题多波束需至少3个极值点计算级次差）
    if len(extrema_data) < 3:
        raise ValueError(f"仅检测到{len(extrema_data)}个极值点，不足计算多波束干涉厚度")

    print(f"多波束极值检测结果：极大值{len(peak_indices)}个，极小值{len(valley_indices)}个，共{len(extrema_data)}个")
    return extrema_data, peak_indices, valley_indices


# -------------------------- 4. 多波束干涉厚度计算（基于GB/T 42905-2023公式） --------------------------
def calculate_multibeam_thickness(extrema_data, incident_angle):
    """
    依据GB/T 42905-2023公式（10.7、10.8）计算SiC外延层厚度
    :param extrema_data: 极值点数据
    :param incident_angle: 入射角（°）
    :return: 厚度计算结果（DataFrame）、平均厚度（μm）、厚度RSD（%）
    """
    n = SiC_REFRACTIVE_INDEX  # 标准给定SiC折射率
    angle_rad = incident_angle * pi / 180  # 入射角转弧度
    thickness_results = []

    # 步骤1：计算干涉级次基数k0（基于极值点波长差，B题多波束级次差为0.5）
    lambda_ref = extrema_data.iloc[0]["波长(nm)"]  # 参考波长（最长波长）
    k0_estimates = []

    for i in range(1, len(extrema_data)):
        lambda_i = extrema_data.iloc[i]["波长(nm)"]
        m_i = i * 0.5  # 峰-谷/谷-峰的级次差（多波束极值间隔为0.5级）
        delta_lambda = lambda_ref - lambda_i
        if delta_lambda < 10:  # 过滤微小波长差（避免计算误差）
            continue
        # 级次基数估算：k0 = (m_i * lambda_i) / delta_lambda
        k0 = (m_i * lambda_i) / delta_lambda
        k0_estimates.append(k0)

    if not k0_estimates:
        raise ValueError("无法估算干涉级次基数，需更多有效极值点")
    k0 = np.median(k0_estimates)  # 用中位数抗异常值（多波束稳定性更优）
```

```python

    # 步骤2：计算每个极值点对应的厚度
    for idx, row in extrema_data.iterrows():
        lambda_nm = row["波长(nm)"]
        lambda_μm = lambda_nm / 1000   # 转换为μm（标准厚度单位）
        m_i = idx * 0.5  # 当前极值点与参考点的级次差
        k_i = k0 + m_i  # 当前极值点的干涉级次

        # GB/T 42905-2023公式（10.8）：附加相移影响可忽略（小数点后第三位）
        # T = (k_i - 0.5) * λ_μm / (2 * sqrt(n² - sin²θ))
        denominator = 2 * sqrt(n ** 2 - sin(angle_rad) ** 2)
        thickness_μm = (k_i - 0.5) * lambda_μm / denominator

        # 筛选合理厚度（标准测试范围3-200μm）
        if 3 <= thickness_μm <= 200:
            thickness_results.append({
                "波长(nm)": lambda_nm,
                "反射率(%)": row["反射率(%)"],
                "极值类型": row["极值类型"],
                "干涉级次k_i": round(k_i, 4),
                "厚度(μm)": round(thickness_μm, 4)
            })

    # 结果整理
    result_df = pd.DataFrame(thickness_results)
    if result_df.empty:
        raise ValueError("无有效厚度计算结果，需调整极值点筛选条件")

    # 统计指标（符合标准精密度要求：单个实验室RSD≤1%）
    avg_thickness = result_df["厚度(μm)"].mean()
    std_thickness = result_df["厚度(μm)"].std()
    thickness_rsd = (std_thickness / avg_thickness) * 100 if avg_thickness != 0 else np.inf

    print(f"厚度统计：平均厚度={avg_thickness:.4f}μm，标准差={std_thickness:.4f}μm，RSD={thickness_rsd:.2f}%")
    return result_df, avg_thickness, thickness_rsd


# ------------------------- 5. 结果可视化（展示多波束光谱与极值点） ---------------------------
def plot_multibeam_spectrum(data, extrema_data, peak_indices, valley_indices, incident_angle, save_path):
    """
    绘制多波束干涉光谱图（标注极值点，符合B题图2特征）
    :param data: 预处理后的数据
    :param extrema_data: 极值点数据
    :param peak_indices: 极大值索引
    :param valley_indices: 极小值索引
    :param incident_angle: 入射角（°）
    :param save_path: 图像保存路径
    """
    fig, ax = plt.subplots(figsize=(12, 6))

    # 绘制原始光谱
    ax.plot(data["波长(nm)"], data["反射率(%)"], color="#619CFF", alpha=0.8, linewidth=1.2, label="多波束反射光谱")
    # 标注极大值点
```

```
    ax.scatter(
        data.iloc[peak_indices]["波长(nm)"],
        data.iloc[peak_indices]["反射率(%)"],
        color="#00BA38", s=60, marker="^", label="多波束极大值点", zorder=5
    )
    # 标注极小值点
    ax.scatter(
        data.iloc[valley_indices]["波长(nm)"],
        data.iloc[valley_indices]["反射率(%)"],
        color="#F8766D", s=60, marker="v", label="多波束极小值点", zorder=5
    )

    # 图表配置
    ax.set_xlabel("波长 (nm)", fontsize=12)
    ax.set_ylabel("反射率 (%)", fontsize=12)
    ax.set_title(f"SiC多波束干涉光谱（入射角{incident_angle}°）", fontsize=14, fontweight="bold")
    ax.grid(True, linestyle="--", alpha=0.5, color="gray")
    ax.legend(fontsize=10)
    ax.set_xlim(VALID_WAVELENGTH_NM[0], VALID_WAVELENGTH_NM[1])

    # 保存图像
    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.close()
    print(f"光谱图已保存至：{save_path}")


# -------------------------- 6. 主函数（整合流程：数据→极值→厚度→输出） ---------------------------
def main(file_path, incident_angle, file_label):
    """
    主流程：处理单个附件的多波束厚度计算
    :param file_path: 附件路径
    :param incident_angle: 入射角（°）
    :param file_label: 附件标签（如"附件1_10°"）
    :return: 最终平均厚度（μm）
    """
    print(f"\n{'=' * 60}\n开始处理{file_label}...\n{'=' * 60}")

    # 1. 数据预处理
    try:
        data = preprocess_spectral_data(file_path, incident_angle)
        print(
            f"数据预处理完成：有效波长范围{data['波长(nm)'].min():.0f}-{data['波长(nm)'].max():.0f}nm，共{len(data)}个数据点")
    except Exception as e:
        print(f"数据预处理失败：{str(e)}")
        return None

    # 2. 多波束极值检测
    try:
        extrema_data, peak_indices, valley_indices = detect_multibeam_extrema(data)
    except Exception as e:
        print(f"极值检测失败：{str(e)}")
        return None

```

```python
    # 3. 厚度计算
    try:
        thickness_df, avg_thickness, thickness_rsd = calculate_multibeam_thickness(extrema_data, incident_angle)
    except Exception as e:
        print(f"厚度计算失败：{str(e)}")
        return None

    # 4. 结果可视化
    plot_path = f"{OUTPUT_DIR}/plots/{file_label}_多波束光谱图.png"
    plot_multibeam_spectrum(data, extrema_data, peak_indices, valley_indices, incident_angle, plot_path)

    # 5. 结果保存（Excel）
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    excel_path = f"{OUTPUT_DIR}/data/{file_label}_多波束厚度结果_{timestamp}.xlsx"
    with pd.ExcelWriter(excel_path, engine="openpyxl") as writer:
        # 原始预处理数据
        data.to_excel(writer, sheet_name="预处理光谱数据", index=False)
        # 极值点数据
        extrema_data.to_excel(writer, sheet_name="多波束极值点", index=False)
        # 厚度计算结果
        thickness_df.to_excel(writer, sheet_name="厚度计算结果", index=False)
        # 统计汇总
        stats_df = pd.DataFrame({
            "统计项": ["平均厚度(μm)", "厚度标准差(μm)", "厚度RSD(%)", "入射角(°)", "有效极值点数"],
            "数值": [avg_thickness, thickness_df["厚度(μm)"].std(), thickness_rsd, incident_angle, len(extrema_data)]
        })
        stats_df.to_excel(writer, sheet_name="结果统计", index=False)

    print(f"结果数据已保存至：{excel_path}")
    print(f"{file_label}处理完成，最终平均厚度：{avg_thickness:.4f}μm\n")
    return avg_thickness


# -------------------------- 7. 执行入口（适配B题附件1、附件2） --------------------------
if __name__ == "__main__":
    # 配置B题附件路径与入射角（需根据实际文件位置调整）
    附件1路径 = "附件1_processed.xlsx"  # B题附件1：入射角10°
    附件2路径 = "附件2_processed.xlsx"  # B题附件2：入射角15°

    # 执行附件1处理
    附件1平均厚度 = main(
        file_path=附件1路径,
        incident_angle=10,
        file_label="附件1_入射角10°"
    )

    # 执行附件2处理
    附件2平均厚度 = main(
        file_path=附件2路径,
        incident_angle=15,
        file_label="附件2_入射角15°"
    )
```

```python

    # 最终结果汇总
    print(f"\n{'=' * 80}")
    print("B题SiC多波束干涉厚度计算最终结果汇总")
    print(f"附件1（10°）平均厚度：{附件1平均厚度:.4f}μm" if 附件1平均厚度 else "附件1计算失败")
    print(f"附件2（15°）平均厚度：{附件2平均厚度:.4f}μm" if 附件2平均厚度 else "附件2计算失败")
    print(f"{'=' * 80}")
```

Problem_灵敏度分析

```python
# SiC多波束干涉厚度灵敏度分析代码（基于B题与红外反射法标准）
import pandas as pd
import numpy as np
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
import os
from math import sin, sqrt, pi

# -------------------------- 1. 核心参数（源自B题与GB/T 42905-2023标准） --------------------------
SiC_REFRACTIVE_INDEX = 2.55  # 标准明确SiC折射率
VALID_WAVELENGTH_NM = (2500, 5000)  # 有效波长区间（对应标准3-200μm测试范围）
SENSITIVITY_ANGLE_GROUPS = {
    "附件1_processed.xlsx": [8, 9, 10, 11, 12],  # 附件1基础10°+新增角度
    "附件2_processed.xlsx": [13, 14, 15, 16, 17]  # 附件2基础15°+新增角度
}
OUTPUT_DIR = "output/灵敏度分析"
os.makedirs(OUTPUT_DIR, exist_ok=True)  # 自动创建灵敏度分析输出文件夹

# 绘图基础配置（确保图表清晰）
plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.size'] = 11
plt.rcParams['figure.dpi'] = 150


# -------------------------- 2. 数据预处理（适配B题附件格式） --------------------------
def preprocess_data(file_path, incident_angle):
    """简化版数据处理：波数转波长+有效区间筛选+数据清洗"""
    # 读取附件数据（B题附件为Excel格式，无表头）
    data = pd.read_excel(file_path, header=None, names=["波数(cm⁻¹)", "反射率(%)"])
    # 清洗非数值/缺失值
    data = data.apply(pd.to_numeric, errors="coerce").dropna()
    # 波数→波长（B题原理：λ(nm)=1e7/波数(cm⁻¹)）
    data["波长(nm)"] = 1e7 / data["波数(cm⁻¹)"]
    # 筛选有效波长区间（排除干扰）
    data = data[(data["波长(nm)"] >= VALID_WAVELENGTH_NM[0]) & (data["波长(nm)"] <= VALID_WAVELENGTH_NM[1])]
    # 按波长降序（便于干涉级次计算）
    data = data.sort_values("波长(nm)", ascending=False).reset_index(drop=True)
    data["入射角(°)"] = incident_angle
    return data
```

```python


# -------------------------- 3. 多波束极值检测（B题图2多波束特征适配） -------------------------------------
def detect_extrema(data):
    """简化版极值检测：仅保留多波束峰谷核心逻辑"""
    reflectance = data["反射率(%)"].values
    wavelength = data["波长(nm)"].values

    # 多波束峰检测（峰更尖锐，降低显著性阈值）
    peaks, _ = find_peaks(reflectance, distance=5, prominence=0.05,
                          height=np.mean(reflectance) + 0.1 * np.std(reflectance))
    # 多波束谷检测（取负反射率的峰）
    valleys, _ = find_peaks(-reflectance, distance=5, prominence=0.05,
                            height=-(np.mean(reflectance) - 0.1 * np.std(reflectance)))

    # 整合极值点（按波长排序）
    extrema = pd.DataFrame({
        "波长(nm)": np.concatenate([wavelength[peaks], wavelength[valleys]]),
        "反射率(%)": np.concatenate([reflectance[peaks], reflectance[valleys]]),
        "类型": ["峰"] * len(peaks) + ["谷"] * len(valleys)
    }).sort_values("波长(nm)", ascending=False).reset_index(drop=True)

    return extrema if len(extrema) >= 3 else None  # 多波束需至少3个极值点


# -------------------------- 4. 厚度计算（严格遵循GB/T 42905-2023公式） ---------------------------
def calc_thickness(extrema, incident_angle):
    """简化版厚度计算：基于标准10.8公式（忽略微小相移影响）"""
    n = SiC_REFRACTIVE_INDEX
    angle_rad = incident_angle * pi / 180  # 入射角转弧度
    lambda_ref = extrema.iloc[0]["波长(nm)"]  # 参考波长（最长波长）
    k0_est = []

    # 估算干涉级次基数k0（多波束级次差为0.5）
    for i in range(1, len(extrema)):
        lambda_i = extrema.iloc[i]["波长(nm)"]
        delta_lambda = lambda_ref - lambda_i
        if delta_lambda < 10:
            continue
        k0 = (i * 0.5 * lambda_i) / delta_lambda  # 级次差m=i*0.5
        k0_est.append(k0)
    k0 = np.median(k0_est)  # 中位数抗异常值（多波束稳定性更优）

    # 计算每个极值点厚度并取平均
    thickness_list = []
    for idx, row in extrema.iterrows():
        lambda_μm = row["波长(nm)"] / 1000  # 转μm（标准厚度单位）
        k_i = k0 + idx * 0.5  # 当前级次
        # 标准公式：T=(k_i-0.5)*λ/(2*sqrt(n²-sin²θ))
        denom = 2 * sqrt(n ** 2 - sin(angle_rad) ** 2)
        thick = (k_i - 0.5) * lambda_μm / denom
        if 3 <= thick <= 200:  # 标准测试范围（3-200μm）
```

```python
                thickness_list.append(thick)

    return np.mean(thickness_list) if thickness_list else None  # 返回平均厚度


# -------------------------- 5. 灵敏度分析核心逻辑（入射角→厚度响应） --------------------------
def sensitivity_analysis(file_path, angle_group):
    """针对单个附件的灵敏度分析：计算不同入射角的平均厚度"""
    thickness_results = []
    for angle in angle_group:
        # 流程：数据处理→极值检测→厚度计算
        data = preprocess_data(file_path, angle)
        extrema = detect_extrema(data)
        if extrema is None:
            continue
        avg_thick = calc_thickness(extrema, angle)
        if avg_thick is not None:
            thickness_results.append({"入射角(°)": angle, "平均厚度(μm)": round(avg_thick, 4)})
    return pd.DataFrame(thickness_results)


# -------------------------- 6. 结果可视化（汇总所有入射角-厚度数据） --------------------------
def plot_sensitivity_summary(results_dict):
    """绘制灵敏度分析汇总图：所有附件的入射角-厚度关系"""
    fig, ax = plt.subplots(figsize=(10, 6))

    # 区分附件1和附件2数据（用不同样式）
    colors = ["#2E86AB", "#A23B72"]
    markers = ["o", "s"]
    labels = ["附件1_processed（基础10°）", "附件2_processed（基础15°）"]

    for (file, df), color, marker, label in zip(results_dict.items(), colors, markers, labels):
        if not df.empty:
            ax.plot(
                df["入射角(°)"], df["平均厚度(μm)"],
                color=color, marker=marker, markersize=8, linewidth=2, label=label
            )
            # 标注数据点数值
            for _, row in df.iterrows():
                ax.text(
                    row["入射角(°)"], row["平均厚度(μm)"],
                    f'{row["平均厚度(μm)"]:.4f}',
                    ha="center", va="bottom", fontsize=9, color=color
                )

    # 图表配置（突出灵敏度分析主题）
    ax.set_xlabel("入射角（°）", fontsize=12, fontweight="bold")
    ax.set_ylabel("SiC外延层平均厚度（μm）", fontsize=12, fontweight="bold")
    ax.set_title("SiC多波束干涉厚度-入射角灵敏度分析", fontsize=14, fontweight="bold", pad=20)
    ax.grid(True, linestyle="--", alpha=0.5, color="gray")
    ax.legend(loc="best", framealpha=0.9)
```

```python
    ax.set_xlim(min(SENSITIVITY_ANGLE_GROUPS["附件1_processed.xlsx"]) - 0.5,
                max(SENSITIVITY_ANGLE_GROUPS["附件2_processed.xlsx"]) + 0.5)

    # 保存图表到灵敏度分析文件夹
    plot_path = os.path.join(OUTPUT_DIR, "SiC厚度-入射角灵敏度分析图.png")
    plt.tight_layout()
    plt.savefig(plot_path, dpi=300, bbox_inches="tight")
    plt.close()
    print(f"灵敏度分析图已保存至：{plot_path}")


# -------------------------- 7. 执行入口（一键运行灵敏度分析） --------------------------
if __name__ == "__main__":
    # 存储所有附件的灵敏度分析结果
    all_sensitivity_results = {}

    # 遍历两个附件，执行灵敏度分析
    for file_name, angle_group in SENSITIVITY_ANGLE_GROUPS.items():
        print(f"正在处理{file_name}的灵敏度分析（入射角：{angle_group}°）...")
        result_df = sensitivity_analysis(file_name, angle_group)
        all_sensitivity_results[file_name] = result_df
        print(f"{file_name}分析完成，有效数据点：{len(result_df)}个\n")

    # 绘制并保存汇总图
    plot_sensitivity_summary(all_sensitivity_results)
    print("灵敏度分析全部完成！结果图表位于：output/灵敏度分析")
```