

数据处理.py

```
1 import pandas as pd
2 import numpy as np
3 from scipy.signal import savgol_filter
4 import matplotlib.pyplot as plt
5
6 # --- Matplotlib 全局美化设置 ---
7 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
8 plt.rcParams['axes.unicode_minus'] = False
9 plt.rcParams['font.size'] = 14
10 plt.rcParams['axes.labelsize'] = 16
11 plt.rcParams['axes.titlesize'] = 18
12 plt.rcParams['legend.fontsize'] = 14
13
14
15 def preprocess_reflectance_data(fileName):
16     """
17         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。
18         参数：
19             fileName (str): 不含扩展名的Excel文件名。
20         返回：
21             处理后的数据文件路径和图像路径
22     """
23     file_path = fileName + ".xlsx"
24     # 加载数据
25     try:
26         data = pd.read_excel(file_path)
27     except FileNotFoundError:
28         print(f"错误：文件 '{file_path}' 未找到。")
29         return
30     # 提取关键列数据
31     reflectance = data['反射率 (%)']
32     wavelength = data['波长(μm)'] # 波长数据（x轴原始数据）
33     # --- 1. Savitzky-Golay 滤波器 ---
34     window_length = 39
35     polyorder = 2
36     reflectance_savgol = savgol_filter(reflectance, window_length,
37                                         polyorder)
38     # --- 2. 多项式基线校正 ---
39     x = np.arange(len(reflectance_savgol)) # 临时索引（用于基线拟合）
40     degree = 5
41     coeffs = np.polyfit(x, reflectance_savgol, degree)
42     baseline = np.polyval(coeffs, x)
43     reflectance_baseline_corrected = reflectance_savgol - baseline
44     # --- 3. 3σ 异常值处理 ---
45     mean = np.mean(reflectance_baseline_corrected)
46     std = np.std(reflectance_baseline_corrected)
47     threshold = 3 * std
48     outliers = np.abs(reflectance_baseline_corrected - mean) > threshold # 异常值标记
49     # 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）
50     reflectance_final = reflectance_baseline_corrected[~outliers]
51     wavelength_final = wavelength[~outliers] # 只保留非异常值对应的波长
52     # 保存处理后的数据到新的DataFrame
```

```

52     processed_data = data.copy()
53     processed_data['Savitzky-Golay滤波后反射率'] = reflectance_savgo1
54     processed_data['基线校正后反射率'] = reflectance_baseline_corrected
55     processed_data['最终反射率(已剔除异常值)'] = np.nan
56     processed_data.loc[~outliers, '最终反射率(已剔除异常值)'] =
reflectance_final
57     new_file_path = f'{fileName}_processed.xlsx'
58     processed_data.to_excel(new_file_path, index=False)
59     # --- 可视化（修正绘图参数和数据匹配问题） ---
60     plt.figure(figsize=(12, 10))
61     # 1. 原始数据
62     plt.subplot(4, 1, 1)
63     plt.plot(wavelength, reflectance, label='原始数据')
64     plt.title('原始反射率数据')
65     plt.legend()
66     # 2. Savitzky-Golay 滤波后数据
67     plt.subplot(4, 1, 2)
68     plt.plot(wavelength, reflectance_savgo1, label='Savitzky-Golay 滤波后',
color='orange')
69     plt.title('经过 Savitzky-Golay 滤波')
70     plt.legend()
71     # 3. 基线校正后数据
72     plt.subplot(4, 1, 3)
73     plt.plot(wavelength, reflectance_baseline_corrected, label='基线校正后',
color='green')
74     plt.title('经过多项式基线校正')
75     plt.legend()
76     # 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）
77     plt.subplot(4, 1, 4)
78     plt.plot(wavelength_final, reflectance_final, label='最终数据 (已剔除异常
值)', color='red')
79     plt.title('经过 3σ 准则异常值处理')
80     plt.legend()
81     plt.tight_layout()
82     img_path = f'{fileName}_data_preprocessing_steps.png'
83     plt.savefig(img_path, dpi=300) # 增加dpi确保图像清晰度
84     plt.show()
85     return new_file_path, img_path
86
87
88 if __name__ == '__main__':
89     result = preprocess_reflectance_data('附件3')
90     result = preprocess_reflectance_data('附件4')
91

```

Problem2_solution.py

```

1 import warnings
2 import pandas as pd
3 import numpy as np
4 from scipy.signal import find_peaks
5 import matplotlib.pyplot as plt
6 from math import sin, sqrt, pi
7 from matplotlib.patches import Patch
8 import os
9 from datetime import datetime
10 from scipy import stats

```

```

11
12 # --- 全局设置 ---
13 warnings.filterwarnings('ignore')
14 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
15 plt.rcParams['axes.unicode_minus'] = False
16 plt.rcParams['font.size'] = 12 # 缩小字体，避免图表拥挤
17 color1 = "#00BA38" # 极大值点
18 color2 = "#619cff" # 极小值点
19 color3 = "#C8E193" # 厚度分布
20 color4 = "#F8766D" # 残差分析
21 grid_color = 'midgray'
22 text_color = 'black'
23 bg_color = 'lightgray'
24
25 # 确保输出目录存在
26 os.makedirs('output', exist_ok=True)
27 os.makedirs('output/images', exist_ok=True)
28 os.makedirs('output/excel', exist_ok=True)
29 os.makedirs('output/txt', exist_ok=True)
30 os.makedirs('output/stability', exist_ok=True)
31
32
33 # ----- 1. 修正物理模型参数（碳化硅标准参数） -----
34
35 def cauchy_refractive_index(lambda_μm, A=2.65, B=0.015, C=1e-7):
36     """Cauchy模型：适配碳化硅红外波段（2.5-5μm）"""
37     n = A + B / (lambda_μm ** 2) + C / (lambda_μm ** 4)
38     return n
39
40 def sellmeier_refractive_index(lambda_μm, A1=6.91, B1=0.202):
41     """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）"""
42     if lambda_μm ** 2 <= B1:
43         raise ValueError(f"波长 {lambda_μm:.2f}μm 过小，需λ² > {B1}μm²")
44     n_squared = 1 + (A1 * lambda_μm ** 2) / (lambda_μm ** 2 - B1)
45     return sqrt(n_squared)
46
47
48 # ----- 2. 数据预处理：适配实际波长范围 -----
49
50 def preprocess_data(file_path, angle):
51     try:
52         data = pd.read_excel(file_path)
53     except Exception:
54         data = pd.read_csv(file_path)
55
56     # 数据清洗与波长转换（波数cm⁻¹ → 波长nm: λ(nm)=1e7/波数(cm⁻¹)）
57     data.columns = ["波数 (cm⁻¹)", "反射率 (%)"]
58     data["波数 (cm⁻¹)"] = pd.to_numeric(data["波数 (cm⁻¹)"],
59                                         errors='coerce')
60     data["反射率 (%)"] = pd.to_numeric(data["反射率 (%)"], errors='coerce')
61     data = data.dropna()
62     data["λambda_nm"] = 1e7 / data["波数 (cm⁻¹)"]
63
64     # 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
65     valid_mask = (data["λambda_nm"] >= 2500) & (data["λambda_nm"] <= 5000)

```

```

64     data_valid = data[valid_mask].sort_values("lambda_nm",
65     ascending=False).reset_index(drop=True)
66
67     # 增加数据量判断（降低阈值，避免误判）
68     if len(data_valid) < 50:
69         raise ValueError(f"有效数据点仅{len(data_valid)}个，需≥50个")
70
71     data_valid["incident_angle"] = angle
72     print(
73         f"预处理后数据：波长范围{data_valid['lambda_nm'].min():.0f}~"
74         f"{data_valid['lambda_nm'].max():.0f}nm，共{len(data_valid)}个点")
75
76     return data_valid
77
78 # ----- 3. 优化极值点检测（降低阈值，增加峰显著性判断） -----
79
80 def detect_interference_extrema(data_valid):
81     R = data_valid["反射率 (%)"].values
82     lambda_nm = data_valid["lambda_nm"].values
83     r_mean = np.mean(R)
84     r_std = np.std(R)
85
86     # 优化阈值：降低高度门槛，增加峰显著性（prominence）过滤噪声
87     min_height_max = r_mean + 0.2 * r_std # 极大值最小高度（原代码仅用
88     min_height，此处分开设置更灵活）
89     min_height_min = r_mean - 0.8 * r_std # 极小值最小高度（降低门槛，避免漏检）
90     min_distance = max(10, int(len(data_valid) * 0.02)) # 最小峰间距：至少10
91     个点，避免过疏
92     prominence = 0.05 # 峰显著性：过滤微小波动（反射率变化≥0.05%才视为峰）
93
94     # 检测极大值（峰）和极小值（谷）
95     peak_indices, _ = find_peaks(
96         R,
97         distance=min_distance,
98         height=min_height_max,
99         prominence=prominence
100    )
101
102     valley_indices, _ = find_peaks(
103         -R, # 极小值=负反射率的极大值
104         distance=min_distance,
105         height=-min_height_min, # 对应原反射率的极小值高度
106         prominence=prominence
107    )
108
109     # 整合极值点
110     extrema = pd.concat([
111         pd.DataFrame({"type": "max", "lambda_nm": lambda_nm[peak_indices],
112         "反射率 (%)": R[peak_indices]}),
113         pd.DataFrame({"type": "min", "lambda_nm":
114             lambda_nm[valley_indices], "反射率 (%)": R[valley_indices]})])
115     ].sort_values("lambda_nm", ascending=False).reset_index(drop=True)
116
117     # 打印极值点数量，方便调试
118     print(f"检测到极值点：共{len(extrema)}个（极大值{len(peak_indices)}个，极小值
119     {len(valley_indices)}个）")
120     if len(extrema) < 3:
121         raise ValueError(f"极值点不足3个，需调整检测参数")
122
123

```

```

114     return extrema
115
116
117 # ----- 4. 优化厚度计算（增加异常值过滤） -----
118
119 def calculate_thickness(extrema, incident_angle, ref_model,
120                         **model_params):
121     if len(extrema) < 2:
122         return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
123
124     ref_index_func = cauchy_refractive_index if ref_model == "cauchy" else
125     sellmeier_refractive_index
126     angle_rad = incident_angle * pi / 180
127     lambda0_nm = extrema.iloc[0]["lambda_nm"]
128     k0_estimates = []
129
130     # 计算k0（干涉级次基数）：过滤过小的波长差，避免异常值
131     for i in range(1, len(extrema)):
132         m_i = i * 0.5 # 干涉级次差（峰-谷/谷-峰差0.5级）
133         lambda_i_nm = extrema.iloc[i]["lambda_nm"]
134         lambda_diff = lambda0_nm - lambda_i_nm
135         if lambda_diff > 10: # 波长差≥10nm才计算，避免除以微小值
136             k0_est = (m_i * lambda_i_nm) / lambda_diff
137             k0_estimates.append(k0_est)
138
139     if not k0_estimates:
140         print("无有效k0估算值，需调整波长差阈值")
141         return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
142
143     k0_final = np.median(k0_estimates) # 用中位数抗异常值
144     results = []
145     residuals = []
146     expected_k = []
147     actual_k = []
148
149     # 计算厚度并过滤异常值（厚度为正且在合理范围：0.1-100μm）
150     for i, row in extrema.iterrows():
151         m_i = i * 0.5
152         k_i = k0_final + m_i
153         lambda_i_nm = row["lambda_nm"]
154         lambda_i_um = lambda_i_nm / 1000.0
155
156         # 计算折射率
157         try:
158             n_i = ref_index_func(lambda_i_um, **model_params)
159         except Exception as e:
160             print(f"计算折射率失败: {e}, 跳过该点")
161             continue
162
163         # 计算厚度（干涉公式: 2ndcosθ = kλ → d = kλ/(2ncosθ), cosθ=√(n²-
164         sin²θ_incident))
165         denominator = 2 * sqrt(n_i ** 2 - sin(angle_rad) ** 2)
166         thickness = (k_i * lambda_i_um) / denominator
167
168         # 过滤异常厚度（碳化硅外延层常见厚度0.5-50μm）
169         if 0.1 < thickness < 100:
170             results.append({
171                 "lambda_nm": lambda_i_nm,

```

```

168         "thickness_μm": thickness,
169         "k_i": k_i,
170         "n_i": n_i
171     })
172     # 计算残差（验证模型一致性）
173     expected_k_val = (2 * thickness * denominator) / λambda_i_μm
174     residuals.append(k_i - expected_k_val)
175     expected_k.append(expected_k_val)
176     actual_k.append(k_i)
177
178     thickness_df = pd.DataFrame(results)
179     if thickness_df.empty:
180         print("无有效厚度值，需调整厚度过滤范围")
181         return thickness_df, np.nan, np.nan, np.nan, pd.DataFrame()
182
183     # 计算厚度统计量
184     avg_thickness = thickness_df["thickness_μm"].mean()
185     std_thickness = thickness_df["thickness_μm"].std()
186     rsd = (std_thickness / avg_thickness) * 100 if avg_thickness != 0 else
187     np.inf
188
189     # 残差数据
190     residual_data = pd.DataFrame({
191         "λambda_nm": thickness_df["λambda_nm"],
192         "actual_k": actual_k,
193         "expected_k": expected_k,
194         "residual": residuals,
195         "thickness_μm": thickness_df["thickness_μm"]
196     })
197
198     print(f"{ref_model}模型：平均厚度{avg_thickness:.4f}μm, RSD={rsd:.2f}%")
199     return thickness_df, avg_thickness, std_thickness, rsd, residual_data
200
201 # ----- 5. 优化子区间搜索（适配实际波长范围） -----
202
203 def find_best_bands(data_valid, incident_angle, model_name, model_params,
204                      window_size=800, step_size=200,
205                      min_extrema=3):
206     """
207     window_size: 子区间宽度（800nm, 适配2500–5000nm范围, 可生成多个子区间）
208     step_size: 步长（200nm, 增加子区间数量）
209     min_extrema: 每个子区间至少3个极值点（降低门槛）
210     """
211
212     all_bands_results = []
213     min_lambda = data_valid["λambda_nm"].min()
214     max_lambda = data_valid["λambda_nm"].max()
215     ref_index_func = cauchy_refractive_index if model_name == "cauchy" else
216     sellmeier_refractive_index
217
218     # 生成子区间（确保不超出数据范围）
219     for start_lambda in np.arange(min_lambda, max_lambda - window_size +
220                                    100, step_size):
221         end_lambda = start_lambda + window_size
222         band_data = data_valid[(data_valid["λambda_nm"] >= start_lambda) &
223                                (data_valid["λambda_nm"] <= end_lambda)]
224
225         # 子区间数据量≥30个点才分析

```

```

220     if len(band_data) < 30:
221         continue
222
223     try:
224         extrema = detect_interference_extrema(band_data)
225         if len(extrema) < min_extrema:
226             continue
227
228         # 计算该区间厚度
229         _, avg_thick, _, rsd, residuals = calculate_thickness(extrema,
230         incident_angle, model_name, **model_params)
231
232         # 过滤合理结果 (RSD<50%, 残差非空)
233         if pd.notna(rsd) and rsd < 50 and residuals is not None and not
234         residuals.empty:
235             # 计算折射率范围
236             lambda_min_μm = start_lambda / 1000.0
237             lambda_max_μm = end_lambda / 1000.0
238             n_min = ref_index_func(lambda_max_μm, **model_params) # 波
239             n_max = ref_index_func(lambda_min_μm, **model_params)
240
241             all_bands_results.append({
242                 "start_lambda": start_lambda,
243                 "end_lambda": end_lambda,
244                 "avg_thickness": avg_thick,
245                 "rsd": rsd,
246                 "extrema_count": len(extrema),
247                 "n_min": n_min,
248                 "n_max": n_max,
249                 "residual_std": residuals["residual"].std(),
250                 "residual_mean": residuals["residual"].mean(),
251                 "residual_rmse": np.sqrt(np.mean(residuals["residual"])
252             ** 2))
253             })
254         except Exception as e:
255             print(f"子区间 [{start_lambda:.0f}-{end_lambda:.0f}nm] 分析失败:
256             {str(e)[:50]}")
257             continue
258
259
260     bands_df = pd.DataFrame(all_bands_results)
261     print(f"{model_name}模型: 找到{len(bands_df)}个有效子区间")
262     return bands_df
263
264
265
266 # ----- 6. 优化加权厚度计算 (降低筛选门槛) -----
267
268 def get_weighted_thickness(band_results_df, rsd_threshold=30.0, top_n=3):
269     """降低RSD阈值 (30%)，确保有足够区间参与加权"""
270     if band_results_df.empty:
271         return np.nan, None
272
273     # 筛选优良区间 (RSD<30%且残差RMSE<0.5)
274     good_bands = band_results_df[
275         (band_results_df['rsd'] < rsd_threshold) &
276         (band_results_df['residual_rmse'] < 0.5)
277     ].copy()

```

```

272     # 若无优良区间，取综合得分前3的区间
273     if good_bands.empty:
274         band_results_df['combined_score'] = band_results_df['rsd'] * 0.6 +
275             band_results_df['residual_rmse'] * 0.4
276         good_bands =
277         band_results_df.sort_values('combined_score').head(top_n).copy()
278         print(f"无满足RSD<{rsd_threshold}%的区间，取综合得分前{top_n}的区间")
279
280     # 计算权重（基于RSF和残差，避免除以零）
281     good_bands['weight_rsd'] = 1 / (good_bands['rsd'] + 1e-6)
282     good_bands['weight_residual'] = 1 / (good_bands['residual_rmse'] + 1e-
283     6)
284     good_bands['weight'] = (good_bands['weight_rsd'] +
285     good_bands['weight_residual']) / 2
286     total_weight = good_bands['weight'].sum()
287     good_bands['norm_weight'] = good_bands['weight'] / total_weight
288
289     # 加权平均厚度
290     weighted_avg = np.sum(good_bands['avg_thickness'] *
291     good_bands['weight']) / total_weight
292     print(f"加权平均厚度: {weighted_avg:.4f}μm (基于{len(good_bands)}个区间)")
293     return weighted_avg, good_bands
294
295
296 # ----- 7. 可视化与报告函数: 增加空值防护 -----
297
298 def plot_spectrum_with_extrema(data_valid, extrema, title, save_path=None):
299     plt.figure(figsize=(12, 6))
300     plt.plot(data_valid["Lambda_nm"], data_valid["反射率 (%)"],
301     color='grey', alpha=0.8, label='反射光谱')
302
303     # 标记极值点（增加空值判断）
304     if not extrema.empty:
305         max_points = extrema[extrema['type'] == 'max']
306         min_points = extrema[extrema['type'] == 'min']
307         plt.scatter(max_points['Lambda_nm'], max_points['反射率 (%)'],
308         color=color1, s=50, label='极大值点', zorder=5)
309         plt.scatter(min_points['Lambda_nm'], min_points['反射率 (%)'],
310         color=color2, s=50, label='极小值点', zorder=5)
311
312         plt.title(title, fontsize=14)
313         plt.xlabel('波长 (nm)', fontsize=12)
314         plt.ylabel('反射率 (%)', fontsize=12)
315         plt.grid(True, linestyle='--', alpha=0.5)
316         plt.legend()
317         plt.tight_layout()
318         if save_path:
319             plt.savefig(save_path, dpi=300)
320             plt.close()
321
322
323 def save_text_report(models_results, final_thickness, incident_angle,
324 file_path):
325     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
326     txt_path = f"output/txt/report_angle_{incident_angle}_{timestamp}.txt"
327     with open(txt_path, 'w', encoding='utf-8') as f:
328         f.write("=" * 60 + "\n")
329         f.write(f"薄膜厚度分析报告 - 入射角 {incident_angle}°\n")

```

```
320         f.write(f"分析时间: {datetime.now().strftime('%Y-%m-%d\n%H:%M:%S')}\n")
321         f.write(f"分析文件: {file_path}\n")
322         f.write("=" * 60 + "\n\n")
323
324         # Cauchy模型（增加空值判断）
325         cauchy = models_results.get('cauchy', {})
326         f.write("1. Cauchy 模型分析结果:\n")
327         f.write(f"    - 加权平均厚度: {cauchy.get('final_thickness', 'N/A'):.4f} μm\n" if pd.notna(
328             cauchy.get('final_thickness')) else "    - 加权平均厚度: N/A\n")
329         details_c = cauchy.get('details')
330         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")
331
332         if details_c is not None and not details_c.empty:
333             overall_residual_std = details_c['residual_std'].mean()
334             overall_residual_rmse = details_c['residual_rmse'].mean()
335             f.write(f"        - 整体稳定性指标:\n")
336             f.write(f"        残差标准差: {overall_residual_std:.4f}\n")
337             f.write(f"        残差均方根误差: {overall_residual_rmse:.4f}\n")
338             f.write("        - 各区间详情:\n")
339             for _, row in details_c.iterrows():
340                 f.write(f"            波长范围: {row['start_lambda']:.0f}-
{row['end_lambda']:.0f} nm, "
341                     f"厚度: {row['avg_thickness']:.4f} μm, "
342                     f"RSD: {row['rsd']:.2f}%, "
343                     f"残差RMSE: {row['residual_rmse']:.4f}, "
344                     f"权重: {row['norm_weight']:.2f}, "
345                     f"折射率范围: {row['n_min']:.4f}-
{row['n_max']:.4f}\n")
346
347         # Sellmeier模型（同理）
348         sellmeier = models_results.get('sellmeier', {})
349         f.write("\n2. Sellmeier 模型分析结果:\n")
350         f.write(f"    - 加权平均厚度: {sellmeier.get('final_thickness', 'N/A'):.4f} μm\n" if pd.notna(
351             sellmeier.get('final_thickness')) else "    - 加权平均厚度:
N/A\n")
352         details_s = sellmeier.get('details')
353         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")
354
355         if details_s is not None and not details_s.empty:
356             overall_residual_std = details_s['residual_std'].mean()
357             overall_residual_rmse = details_s['residual_rmse'].mean()
358             f.write(f"        - 整体稳定性指标:\n")
359             f.write(f"        残差标准差: {overall_residual_std:.4f}\n")
360             f.write(f"        残差均方根误差: {overall_residual_rmse:.4f}\n")
361             f.write("        - 各区间详情:\n")
362             for _, row in details_s.iterrows():
363                 f.write(f"            波长范围: {row['start_lambda']:.0f}-
{row['end_lambda']:.0f} nm, "
364                     f"厚度: {row['avg_thickness']:.4f} μm, "
365                     f"RSD: {row['rsd']:.2f}%, "
366                     f"残差RMSE: {row['residual_rmse']:.4f}, "
367                     f"权重: {row['norm_weight']:.2f}, "
```

```

368                     f"折射率范围: {row['n_min']: .4f}-
369                     {row['n_max']: .4f}\n")
370
371         # 综合结果
372         f.write("\n" + "=" * 40 + "\n")
373         f.write(f"最终综合厚度: {final_thickness:.4f} μm\n" if
374         pd.notna(final_thickness) else "最终综合厚度: N/A\n")
375         f.write("=" * 40 + "\n")
376     return txt_path
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
    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,
markeredgecolor='black', markersize=8, label='极大值点'),
            plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color2,
markeredgecolor='black', 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 (%)')

```

```

416     ax1.grid(True, linestyle='--', alpha=0.5)
417
418     # 子图2-3: 各模型区间权重 (空值防护)
419     for i, (model, result) in enumerate(models_results.items()):
420         ax = fig.add_subplot(gs[1, i])
421         details = result.get('details')
422         if details is not None and not details.empty:
423             details['band_label'] = details.apply(
424                 lambda row: f'{int(row['start_lambda']) / 1000}-
425 {int(row['end_lambda']) / 1000})k", axis=1)
426             bars = ax.bar(details['band_label'], details['norm_weight'],
427 color=colors[model], alpha=0.7)
428             ax.set_title(f'{model.capitalize()} 模型各区间归一化权重',
429             fontsize=13)
430             ax.set_ylabel('归一化权重')
431             ax.set_ylim(0, 1.1)
432             ax.tick_params(axis='x', rotation=45)
433             for bar, weight in zip(bars, details['norm_weight']):
434                 ax.text(bar.get_x() + bar.get_width() / 2,
435 bar.get_height(), f'{weight:.2f}', ha='center', va='bottom',
436                 fontsize=9)
437             else:
438                 ax.text(0.5, 0.5, '无有效区间', ha='center', va='center',
439                 transform=ax.transAxes)
440                 ax.set_title(f'{model.capitalize()} 模型分析结果')
441                 ax.set_xticks([])
442
443     # 子图4-5: 厚度分布 (空值防护)
444     for i, (model, result) in enumerate(models_results.items()):
445         ax = fig.add_subplot(gs[2, i])
446         details = result.get('details')
447         if details is not None and not details.empty:
448             ax.hist(details['avg_thickness'], bins=5, alpha=0.7,
449 color=colors[model])
450             mean_thick = details['avg_thickness'].mean()
451             ax.axvline(mean_thick, color='red', linestyle='--',
452 linewidth=2, label=f'平均: {mean_thick:.4f} μm')
453             ax.set_title(f'{model.capitalize()} 模型厚度分布', fontsize=13)
454             ax.set_xlabel('厚度 (μm)')
455             ax.set_ylabel('区间数量')
456             ax.legend()
457             else:
458                 ax.text(0.5, 0.5, '无有效数据', ha='center', va='center',
459                 transform=ax.transAxes)
460                 ax.set_xticks([])
461
462     # 子图6-7: 残差统计 (空值防护)
463     for i, (model, result) in enumerate(models_results.items()):
464         ax = fig.add_subplot(gs[3, i])
465         details = result.get('details')
466         if details is not None and not details.empty:
467             stats_data = [
468                 details['residual_mean'].mean(),
469                 details['residual_std'].mean(),
470                 details['residual_rmse'].mean()
471             ]
472             bars = ax.bar(['平均残差', '残差标准差', '残差RMSE'], stats_data,
473 color=color4, alpha=0.7)

```

```

465         ax.set_title(f'{model.capitalize()} 模型残差统计', fontsize=13)
466         ax.set_ylabel('值')
467         ax.tick_params(axis='x', rotation=30)
468         for bar, val in zip(bars, stats_data):
469             ax.text(bar.get_x() + bar.get_width() / 2,
470                     bar.get_height(), f'{val:.4f}', ha='center', va='bottom',
471                     fontsize=9)
472         else:
473             ax.text(0.5, 0.5, '无残差数据', ha='center', va='center',
474                     transform=ax.transAxes)
475             ax.set_xticks([])
476
477             # 子图8: 最终结果汇总
478             ax8 = fig.add_subplot(gs[4, :])
479             ax8.axis('off')
480             t_c = models_results.get('cauchy', {}).get('final_thickness', 'N/A')
481             t_s = models_results.get('sellmeier', {}).get('final_thickness', 'N/A')
482             c_details = models_results.get('cauchy', {}).get('details')
483             s_details = models_results.get('sellmeier', {}).get('details')
484             c_count = len(c_details) if (c_details is not None and not
485             c_details.empty) else 0
486             s_count = len(s_details) if (s_details is not None and not
487             s_details.empty) else 0
488             c_rmse = c_details['residual_rmse'].mean() if (c_details is not None
489             and not c_details.empty) else 'N/A'
490             s_rmse = s_details['residual_rmse'].mean() if (s_details is not None
491             and not s_details.empty) else 'N/A'
492
493             summary_text = (
494                 f"最终分析结果汇总 (入射角: {incident_angle}°)\n\n"
495                 f"1. Cauchy 模型:\n"
496                 f"    - 加权平均厚度: {t_c:.4f} μm\n" if pd.notna(t_c) else "    - 加权
497                 平均厚度: N/A\n"
498
499                 f"    - 分析基于 {c_count} 个优良波长区间\n"
500
501                 f"    - 平均残差RMSE: {c_rmse:.4f}\n" if isinstance(
502                     c_rmse, (int, float)) else "    - 平均残差RMSE: N/A\n\n"
503                 f"2. Sellmeier 模型:\n"
504                 f"    - 加权平均厚度: {t_s:.4f} μm\n"
505
506             if pd.notna(t_s) else "    - 加权平均厚度: N/A\n"
507
508                 f"    - 分析基于 {s_count} 个优良波长区间\n"
509
510                 f"    - 平均残差RMSE: {s_rmse:.4f}\n" if isinstance(
511                     s_rmse, (int, float)) else "    - 平均残差RMSE: N/A\n\n"
512
513             f"-----\n"
514             f"-----\n"
515             f"综合两个模型的稳定性加权平均后, \n"
516             f"最终计算得到的薄膜厚度为:\n"
517             f"{final_thickness:.4f} μm" if pd.notna(
518                 final_thickness) else "最终计算得到的薄膜厚度为: N/A"
519             )
520             ax8.text(0.5, 0.5, summary_text, ha='center', va='center', fontsize=12,
521                     bbox=dict(boxstyle="round,pad=0.5", fc='aliceblue',
522                     ec='steelblue', lw=1))
523             plt.tight_layout()

```

```
508     report_path = f"output/images/final_report_angle_{incident_angle}.png"
509     plt.savefig(report_path, dpi=300)
510     plt.close()
511     return report_path
512
513
514 # ----- 8. 主函数: 增加日志输出, 便于调试 -----
515
515 def main(file_path, incident_angle):
516     print("=" * 60)
517     print(f"开始分析文件: {file_path}, 入射角: {incident_angle}°")
518     print("=" * 60)
519     try:
520         # 步骤1: 数据预处理
521         data_valid = preprocess_data(file_path, incident_angle)
522         print("步骤 1/6: 数据预处理完成。")
523
524         # 步骤2: 绘制光谱及极值点图
525         try:
526             extrema = detect_interference_extrema(data_valid)
527             spec_path = f"output/images/反射光谱及干涉极值点 (入射角 {incident_angle}°).png"
528             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)
529             print(f"已保存光谱图: {spec_path}")
530         except Exception as e:
531             print(f"生成光谱图失败: {e}")
532
533         # 步骤3: 模型参数 (碳化硅适配)
534         cauchy_params = {"A": 2.65, "B": 0.015, "C": 1e-7} # Cauchy标准参数
535         sellmeier_params = {"A1": 6.91, "B1": 0.202} # Sellmeier碳化硅参数
536         models_results = {}
537
538         # 步骤4: 各模型计算
539         for model_name, params in [("cauchy", cauchy_params), ("sellmeier", sellmeier_params)]:
540             print(f"\n--- 正在为 [{model_name.capitalize()}] 模型寻找最佳分析区间... ---")
541             bands_df = find_best_bands(data_valid, incident_angle, model_name, params)
542
543             if bands_df.empty:
544                 print(f"警告: {model_name.capitalize()} 模型未找到有效区间, 跳过加权计算")
545                 models_results[model_name] = {'final_thickness': np.nan,
546 'details': None}
547                 continue
548
549             # 计算加权厚度
550             weighted_thickness, good_bands =
551             get_weighted_thickness(bands_df)
552             if pd.isna(weighted_thickness):
553                 print(f"警告: {model_name.capitalize()} 模型无法计算加权厚度")
554                 models_results[model_name] = {'final_thickness': np.nan,
555 'details': None}
555             else:
```

```

555         models_results[model_name] = {'final_thickness':
556             weighted_thickness, 'details': good_bands}
557             print(f"步骤 3/6: {model_name.capitalize()} 模型计算完成")
558
559     # 步骤5: 综合两个模型结果
560     print("\n--- 正在综合两个模型的结果... ---")
561     t_cauchy = models_results.get("cauchy", {}).get('final_thickness')
562     t_sellmeier = models_results.get("sellmeier",
563     {}).get('final_thickness')
564     valid_thickness = [t for t in [t_cauchy, t_sellmeier] if
565     pd.notna(t)]
566
567     if not valid_thickness:
568         final_thickness = np.nan
569         print("错误: 两个模型均无有效厚度")
570     else:
571         # 基于残差RMSE计算模型权重
572         c_details = models_results.get('cauchy', {}).get('details')
573         s_details = models_results.get('sellmeier', {}).get('details')
574         c_rmse = c_details['residual_rmse'].mean() if (c_details is not
575         None and not c_details.empty) else np.inf
576         s_rmse = s_details['residual_rmse'].mean() if (s_details is not
577         None and not s_details.empty) else np.inf
578
579         # 权重 = 1/残差RMSE (残差越小, 权重越大)
580         weight_c = 1 / c_rmse if c_rmse != np.inf else 0
581         weight_s = 1 / s_rmse if s_rmse != np.inf else 0
582         total_weight = weight_c + weight_s
583
584         if total_weight == 0:
585             final_thickness = np.mean(valid_thickness)
586             print(f"模型权重计算失败, 使用简单平均: {final_thickness:.4f}
587         μm")
588         else:
589             norm_c = weight_c / total_weight
590             norm_s = weight_s / total_weight
591             final_thickness = (t_cauchy * norm_c) + (t_sellmeier *
592             norm_s)
593             print(f"最终综合厚度: {final_thickness:.4f} μm")
594             print(f"模型权重 - Cauchy: {norm_c:.2f}, sellmeier:
595             {norm_s:.2f}")
596
597         # 步骤6: 稳定性分析
598         print("\n步骤 4/6: 执行稳定性分析...")
599         # 简化稳定性分析 (避免空值报错)
600         stability_path =
601             f"output/stability/model_consistency_{incident_angle}.png"
602             try:
603                 plot_model_consistency(models_results, incident_angle,
604                 stability_path)
605                 print(f"稳定性分析图已保存: {stability_path}")
606             except Exception as e:
607                 print(f"稳定性分析失败: {e}")
608
609         # 步骤7: 生成报告
610         print("\n步骤 5/6: 生成可视化分析报告...")
611         try:

```

```

602         report_path = generate_final_report(data_valid, models_results,
603                                             final_thickness, incident_angle)
604         print(f"分析报告已保存至: {report_path}")
605     except Exception as e:
606         print(f"生成报告失败: {e}")
607
608     # 步骤8: 保存Excel和文本报告
609     print("\n步骤 6/6: 保存结果数据...")
610     try:
611         excel_path = save_excel_results(models_results,
612                                         final_thickness, incident_angle)
613         print(f"Excel结果已保存至: {excel_path}")
614     except Exception as e:
615         print(f"保存Excel失败: {e}")
616
617     try:
618         txt_path = save_text_report(models_results, final_thickness,
619                                      incident_angle, file_path)
620         print(f"文本报告已保存至: {txt_path}")
621     except Exception as e:
622         print(f"保存文本报告失败: {e}")
623
624
625     print("\n" + "=" * 60)
626     print(f"分析完成! 最终厚度: {final_thickness:.4f} μm" if
627           pd.notna(final_thickness) else "分析完成, 但无有效厚度")
628     print("=" * 60)
629
630 # ----- 9. 其他辅助函数 (保持原逻辑, 增加空值防护) -----
631
632 def plot_thickness_distribution(thickness_data, model_name, title,
633                                 save_path=None):
634     if len(thickness_data) == 0:
635         print("无厚度数据, 跳过绘图")
636         return
637     plt.figure(figsize=(10, 6))
638     plt.hist(thickness_data, bins=10, alpha=0.7, color=color3)
639     plt.axvline(np.mean(thickness_data), color='red', linestyle='dashed',
640                 linewidth=2,
641                 label=f'平均值: {np.mean(thickness_data):.4f} μm')
642     plt.title(f'{model_name} 模型厚度分布 - {title}', fontsize=13)
643     plt.xlabel('厚度 (μm)')
644     plt.ylabel('频率')
645     plt.grid(True, linestyle='--', alpha=0.5)
646     plt.legend()
647     plt.tight_layout()
648     if save_path:
649         plt.savefig(save_path, dpi=300)
650     plt.close()
651
652 def plot_residual_analysis(residual_data, model_name, angle,
653                            save_path=None):
654     if residual_data.empty:

```

```

652     print("无残差数据, 跳过绘图")
653     return
654     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
655     ax1.scatter(residual_data["lambda_nm"], residual_data["residual"],
656                 color=color4, alpha=0.7)
657     ax1.axhline(y=0, color='r', linestyle='--', alpha=0.5)
658     ax1.set_title(f'{model_name} 模型残差随波长变化')
659     ax1.set_xlabel('波长 (nm)')
660     ax1.set_ylabel('残差 (k_i - 预期k值)')
661     ax1.grid(True, linestyle='--', alpha=0.5)
662
662     ax2.hist(residual_data["residual"], bins=10, alpha=0.7, color=color4)
663     ax2.axvline(x=0, color='r', linestyle='--', alpha=0.5)
664     ax2.set_title(f'{model_name} 模型残差分布')
665     ax2.set_xlabel('残差 (k_i - 预期k值)')
666     ax2.set_ylabel('频率')
667     ax2.grid(True, linestyle='--', alpha=0.5)
668     plt.tight_layout()
669     if save_path:
670         plt.savefig(save_path, dpi=300)
671     plt.close()
672
673
674 def plot_qq_residuals(residual_data, model_name, angle, save_path=None):
675     if residual_data.empty:
676         print("无残差数据, 跳过Q-Q图")
677         return
678     plt.figure(figsize=(8, 6))
679     stats.probplot(residual_data["residual"], plot=plt)
680     plt.title(f'{model_name} 模型残差Q-Q图 (入射角 {angle}°)')
681     plt.grid(True, linestyle='--', alpha=0.5)
682     plt.tight_layout()
683     if save_path:
684         plt.savefig(save_path, dpi=300)
685     plt.close()
686
687
688 def plot_model_consistency(models_results, angle, save_path=None):
689     cauchy_data = models_results.get('cauchy', {}).get('details')
690     sellmeier_data = models_results.get('sellmeier', {}).get('details')
691     if cauchy_data is None or sellmeier_data is None or cauchy_data.empty
692     or sellmeier_data.empty:
693         print("无足够数据绘制模型一致性图")
694         return
695
695     plt.figure(figsize=(10, 8))
696     plt.scatter(cauchy_data['avg_thickness'],
697                 sellmeier_data['avg_thickness'], color=color4, alpha=0.7, s=50)
698     min_val = min(cauchy_data['avg_thickness'].min(),
699                   sellmeier_data['avg_thickness'].min())
700     max_val = max(cauchy_data['avg_thickness'].max(),
701                   sellmeier_data['avg_thickness'].max())
702     plt.plot([min_val, max_val], [min_val, max_val], 'r--', alpha=0.7,
703              label='理想一致性线 (y=x)')
704
704     # 计算相关系数
705     corr_coef = np.corrcoef(cauchy_data['avg_thickness'],
706                            sellmeier_data['avg_thickness'])[0, 1]

```

```
703     plt.text(0.05, 0.95, f'相关系数: r = {corr_coef:.4f}',  
704         transform=plt.gca().transAxes,  
705             bbox=dict(facecolor='white', alpha=0.8))  
706  
707     plt.title(f'Cauchy与Sellmeier模型厚度结果一致性 (入射角 {angle}°)')  
708     plt.xlabel('Cauchy模型厚度 (μm)')  
709     plt.ylabel('Sellmeier模型厚度 (μm)')  
710     plt.grid(True, linestyle='--', alpha=0.5)  
711     plt.legend()  
712     plt.tight_layout()  
713     if save_path:  
714         plt.savefig(save_path, dpi=300)  
715     plt.close()  
716  
717 def save_excel_results(models_results, final_thickness, incident_angle):  
718     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")  
719     excel_path =  
f"output/excel/results_angle_{incident_angle}_{timestamp}.xlsx"  
720     with pd.ExcelWriter(excel_path) as writer:  
721         # 汇总结果  
722         cauchy_rmse = models_results.get('cauchy', {}).get('details',  
pd.DataFrame()).get('residual_rmse').mean() if \  
723             (models_results.get('cauchy', {}).get('details') is not None  
and not models_results.get('cauchy', {}).get(  
724                 'details').empty) else 'N/A'  
725         sellmeier_rmse = models_results.get('sellmeier', {}).get('details',  
pd.DataFrame()).get(  
726             'residual_rmse').mean() if \  
727             (models_results.get('sellmeier', {}).get('details') is not None  
and not models_results.get('sellmeier',  
728  
729                 {}).get(  
730                     'details').empty) else 'N/A'  
731  
732         summary_data = {  
733             "模型": ["Cauchy", "Sellmeier", "综合结果"],  
734             "厚度 (μm)": [  
735                 models_results.get('cauchy', {}).get('final_thickness',  
'N/A'),  
736                 models_results.get('sellmeier', {}).get('final_thickness',  
'N/A'),  
737                 final_thickness if pd.notna(final_thickness) else 'N/A'  
738             ],  
739             "有效区间数量": [  
740                 len(models_results.get('cauchy', {}).get('details', [])) if  
741                 (  
742                     models_results.get('cauchy', {}).get('details')  
is not None) else 0,  
743                 len(models_results.get('sellmeier', {}).get('details', []))  
if  
744                     models_results.get('sellmeier',  
745                     {}).get('details') is not None) else 0,  
746                     "-"  
747             ],  
748             "平均残差RMSE": [  
749                 cauchy_rmse if isinstance(cauchy_rmse, (int, float)) else  
'N/A',
```

```

747             sellmeier_rmse if isinstance(sellmeier_rmse, (int, float))
748         else 'N/A',
749         "_"
750     ]
751     pd.DataFrame(summary_data).to_excel(writer, sheet_name="汇总结果",
752     index=False)
753
754     # 各模型详细区间
755     for model, result in models_results.items():
756         details = result.get('details')
757         if details is not None and not details.empty:
758             details.to_excel(writer, sheet_name=f"{model}_区间详情",
759             index=False)
760
761 def perform_stability_analysis(models_results, data_valid, incident_angle):
762     """简化稳定性分析，避免空值报错"""
763     for model_name, result in models_results.items():
764         details = result.get('details')
765         if details is None or details.empty:
766             print(f"{model_name}模型无有效数据，跳过稳定性分析")
767             continue
768
769         try:
770             # 取第一个有效区间计算残差
771             start_lambda = details.iloc[0]['start_lambda']
772             end_lambda = details.iloc[0]['end_lambda']
773             band_data = data_valid[(data_valid["lambda_nm"] >=
774             start_lambda) & (data_valid["lambda_nm"] <= end_lambda)]
775             extrema = detect_interference_extrema(band_data)
776             model_params = {"A": 2.65, "B": 0.015, "C": 1e-7} if model_name
777             == "cauchy" else {"A1": 6.91, "B1": 0.202}
778             _, _, _, _, residual_data = calculate_thickness(extrema,
779             incident_angle, model_name, **model_params)
780
781             if residual_data is not None and not residual_data.empty:
782                 # 保存残差图
783                 res_plot_path =
784                 f"output/stability/{model_name}_residual_analysis_{incident_angle}.png"
785                 plot_residual_analysis(residual_data,
786                 model_name.capitalize(), incident_angle, res_plot_path)
787                 qq_plot_path =
788                 f"output/stability/{model_name}_residual_qq_{incident_angle}.png"
789                 plot_qq_residuals(residual_data, model_name.capitalize(),
790                 incident_angle, qq_plot_path)
791                 print(f"{model_name}模型残差图已保存")
792             except Exception as e:
793                 print(f"{model_name}模型稳定性分析失败: {e}")
794
795             # 模型一致性图
796             consistency_path =
797             f"output/stability/model_consistency_{incident_angle}.png"
798             plot_model_consistency(models_results, incident_angle,
799             consistency_path)
800
801

```

```

793 # ----- 10. 运行入口（确保文件路径正确） -----
794
795 if __name__ == "__main__":
796     # 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）
797     file1_path = "附件1_processed.xlsx"
798     angle1 = 10
799     file2_path = "附件2_processed.xlsx"
800     angle2 = 15
801
802     # 运行分析
803     print("=" * 80)
804     print("开始分析附件1...")
805     main(file_path=file1_path, incident_angle=angle1)
806
807     print("\n" + "=" * 80)
808     print("开始分析附件2...")
809     main(file_path=file2_path, incident_angle=angle2)

```

Problem2_灵敏度单独.py

```

1 import warnings
2 import pandas as pd
3 import numpy as np
4 from scipy.signal import find_peaks
5 import matplotlib.pyplot as plt
6 from math import sin, sqrt, pi
7 import os
8 from datetime import datetime
9
10 # --- 1. 全局设置（沿用solution，修复颜色bug） ---
11 warnings.filterwarnings('ignore')
12 plt.rcParams["font.family"] = ["SimHei", "Dejavu Sans"]
13 plt.rcParams['axes.unicode_minus'] = False
14 plt.rcParams['font.size'] = 12
15 # 颜色配置（避免'midgray'错误，用标准颜色）
16 color_cauchy = "#00BA38" # Cauchy模型（和solution一致）
17 color_sellmeier = "#619cff" # Sellmeier模型（和solution一致）
18 color_avg = "#C8E193" # 平均值（和solution一致）
19 grid_color = 'gray' # 修复颜色错误
20 text_color = 'black'
21 bg_color = 'lightgray'
22
23 # 确保输出目录（用户要求的"灵敏度分析"文件夹）
24 output_root = "output/灵敏度分析"
25 os.makedirs(output_root, exist_ok=True)
26 os.makedirs(f"{output_root}/excel", exist_ok=True)
27 os.makedirs(f"{output_root}/images", exist_ok=True)
28
29
30 # --- 2. 核心物理模型（完全移植solution，确保参数一致） ---
31 def cauchy_refractive_index(lambda_um, A=2.65, B=0.015, C=1e-7):
32     """Cauchy模型：适配碳化硅红外波段（2.5-5μm） - 与solution完全一致"""
33     n = A + B / (lambda_um ** 2) + C / (lambda_um ** 4)
34     return n
35
36
37 def sellmeier_refractive_index(lambda_um, A1=6.91, B1=0.202):

```

```

38     """Sellmeier模型: 碳化硅标准参数 - 与solution完全一致"""
39     if lambda_μm ** 2 <= B1:
40         raise ValueError(f"波长 {lambda_μm:.2f}μm 过小, 需λ² > {B1}μm²")
41     n_squared = 1 + (A1 * lambda_μm ** 2) / (lambda_μm ** 2 - B1)
42     return sqrt(n_squared)
43
44
45 # --- 3. 数据预处理 (完全移植solution, 确保数据范围一致) ---
46 def preprocess_data(file_path, angle):
47     try:
48         data = pd.read_excel(file_path)
49     except Exception:
50         data = pd.read_csv(file_path)
51     # 数据清洗与波长转换 (和solution逻辑一致)
52     data.columns = ["波数 (cm⁻¹)", "反射率 (%)"]
53     data["波数 (cm⁻¹)"] = pd.to_numeric(data["波数 (cm⁻¹)"],
54                                         errors='coerce')
55     data["反射率 (%)"] = pd.to_numeric(data["反射率 (%)"], errors='coerce')
56     data = data.dropna()
57     data["λnm"] = 1e7 / data["波数 (cm⁻¹)"]
58     # 修正波长范围 (2500-5000nm, 和solution一致)
59     valid_mask = (data["λnm"] >= 2500) & (data["λnm"] <= 5000)
60     data_valid = data[valid_mask].sort_values("λnm",
61                                              ascending=False).reset_index(drop=True)
62     if len(data_valid) < 50:
63         raise ValueError(f"有效数据点仅{len(data_valid)}个, 需≥50个")
64     data_valid["incident_angle"] = angle
65     print(
66         f"预处理后[{os.path.basename(file_path)}]: 波长"
67         f"\n{data_valid['λnm'].min():.0f}-{data_valid['λnm'].max():.0f}nm,"
68         f"共{len(data_valid)}个点")
69     return data_valid
70
71
72
73
74 # --- 4. 极值点检测 (完全移植solution, 确保检测精度) ---
75 def detect_interference_extrema(data_valid):
76     R = data_valid["反射率 (%)"].values
77     λnm = data_valid["λnm"].values
78     r_mean = np.mean(R)
79     r_std = np.std(R)
80     # 和solution一致的检测参数
81     min_height_max = r_mean + 0.2 * r_std
82     min_height_min = r_mean - 0.8 * r_std
83     min_distance = max(10, int(len(data_valid) * 0.02))
84     prominence = 0.05
85     # 检测峰谷
86     peak_indices, _ = find_peaks(R, distance=min_distance,
87                                   height=min_height_max, prominence=prominence)
88     valley_indices, _ = find_peaks(-R, distance=min_distance, height=-
89                                   min_height_min, prominence=prominence)
90     # 整合极值点
91     extrema = pd.concat([
92         pd.DataFrame({"type": "max", "λnm": λnm[peak_indices],
93                      "反射率 (%)": R[peak_indices]}),
94         pd.DataFrame({"type": "min", "λnm": λnm[valley_indices],
95                      "反射率 (%)": R[valley_indices]}),
96     ]).sort_values("λnm", ascending=False).reset_index(drop=True)

```

```

87     print(f"检测到极值点: 共{len(extrema)}个 (极大值{len(peak_indices)}个, 极小值
88 {len(valley_indices)}个)")
89     if len(extrema) < 3:
90         raise ValueError(f"极值点不足3个, 需调整检测参数")
91     return extrema
92
93 # --- 5. 厚度计算 (完全移植Solution, 含异常值过滤) ---
94 def calculate_thickness(extrema, incident_angle, ref_model,
95                         **model_params):
96     if len(extrema) < 2:
97         return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
98     ref_index_func = cauchy_refractive_index if ref_model == "cauchy" else
99     sellmeier_refractive_index
100    angle_rad = incident_angle * pi / 180
101    lambda0_nm = extrema.iloc[0]["lambda_nm"]
102    k0_estimates = []
103    # 和Solution一致的k0计算 (波长差>10nm)
104    for i in range(1, len(extrema)):
105        m_i = i * 0.5
106        lambda_i_nm = extrema.iloc[i]["lambda_nm"]
107        lambda_diff = lambda0_nm - lambda_i_nm
108        if lambda_diff > 10:
109            k0_est = (m_i * lambda_i_nm) / lambda_diff
110            k0_estimates.append(k0_est)
111    if not k0_estimates:
112        print("无有效k0估算值, 需调整波长差阈值")
113        return pd.DataFrame(), np.nan, np.nan, np.nan, pd.DataFrame()
114    k0_final = np.median(k0_estimates)
115    results = []
116    residuals = []
117    expected_k = []
118    actual_k = []
119    # 厚度计算与异常值过滤 (0.1-100μm, 和Solution一致)
120    for i, row in extrema.iterrows():
121        m_i = i * 0.5
122        k_i = k0_final + m_i
123        lambda_i_nm = row["lambda_nm"]
124        lambda_i_um = lambda_i_nm / 1000.0
125        try:
126            n_i = ref_index_func(lambda_i_um, **model_params)
127        except Exception as e:
128            print(f"计算折射率失败: {e}, 跳过该点")
129            continue
130        denominator = 2 * sqrt(n_i ** 2 - sin(angle_rad) ** 2)
131        thickness = (k_i * lambda_i_um) / denominator
132        if 0.1 < thickness < 100:
133            results.append({"lambda_nm": lambda_i_nm, "thickness_um":
134 thickness, "k_i": k_i, "n_i": n_i})
135            expected_k_val = (2 * thickness * denominator) / lambda_i_um
136            residuals.append(k_i - expected_k_val)
137            expected_k.append(expected_k_val)
138            actual_k.append(k_i)
139    thickness_df = pd.DataFrame(results)
140    if thickness_df.empty:
141        print("无有效厚度值, 需调整厚度过滤范围")
142        return thickness_df, np.nan, np.nan, np.nan, pd.DataFrame()
143    # 统计量计算 (和Solution一致)

```

```

141     avg_thickness = thickness_df["thickness_µm"].mean()
142     std_thickness = thickness_df["thickness_µm"].std()
143     rsd = (std_thickness / avg_thickness) * 100 if avg_thickness != 0 else
144         np.inf
145     residual_data = pd.DataFrame({
146         "lambda_nm": thickness_df["lambda_nm"], "actual_k": actual_k,
147         "expected_k": expected_k,
148         "residual": residuals, "thickness_µm": thickness_df["thickness_µm"]
149     })
150     print(f"{ref_model}模型: 平均厚度{avg_thickness:.4f}µm, RSD={rsd:.2f}%")
151     return thickness_df, avg_thickness, std_thickness, rsd, residual_data
152
153 # --- 6. 子区间搜索(完全移植solution, 确保区间数量一致) ---
154 def find_best_bands(data_valid, incident_angle, model_name, model_params,
155     window_size=800, step_size=200,
156     min_extrema=3):
157     """和solution完全一致的子区间搜索: window_size=800, step_size=200"""
158     all_bands_results = []
159     min_lambda = data_valid["lambda_nm"].min()
160     max_lambda = data_valid["lambda_nm"].max()
161     ref_index_func = cauchy_refractive_index if model_name == "cauchy" else
162         sellmeier_refractive_index
163     # 生成子区间(和solution一致的范围)
164     for start_lambda in np.arange(min_lambda, max_lambda - window_size +
165         100, step_size):
166         end_lambda = start_lambda + window_size
167         band_data = data_valid[(data_valid["lambda_nm"] >= start_lambda) &
168             (data_valid["lambda_nm"] <= end_lambda)]
169         if len(band_data) < 30:
170             continue
171         try:
172             extrema = detect_interference_extrema(band_data)
173             if len(extrema) < min_extrema:
174                 continue
175             _, avg_thick, _, rsd, residuals = calculate_thickness(extrema,
176             incident_angle, model_name, **model_params)
177             if pd.notna(rsd) and rsd < 50 and residuals is not None and not
178             residuals.empty:
179                 lambda_min_µm = start_lambda / 1000.0
180                 lambda_max_µm = end_lambda / 1000.0
181                 n_min = ref_index_func(lambda_max_µm, **model_params)
182                 n_max = ref_index_func(lambda_min_µm, **model_params)
183                 all_bands_results.append({
184                     "start_lambda": start_lambda, "end_lambda": end_lambda,
185                     "avg_thickness": avg_thick, "rsd": rsd,
186                     "extrema_count": len(extrema), "n_min": n_min, "n_max": n_max,
187                     "residual_std": residuals["residual"].std(),
188                     "residual_mean": residuals["residual"].mean(),
189                     "residual_rmse": np.sqrt(np.mean(residuals["residual"] **
190                         2))
191                 })
192             except Exception as e:
193                 print(f"子区间[{start_lambda:.0f}-{end_lambda:.0f}nm]分析失败:
194 {str(e)[:50]}")
195             continue
196     bands_df = pd.DataFrame(all_bands_results)

```

```

186     print(f"{model_name}模型: 找到{len(bands_df)}个有效子区间")
187     return bands_df
188
189
190 # --- 7. 加权厚度计算 (完全移植Solution, 确保权重逻辑一致) ---
191 def get_weighted_thickness(band_results_df, rsd_threshold=30.0, top_n=3):
192     """和Solution一致的加权逻辑: RSD<30%, 综合得分排序"""
193     if band_results_df.empty:
194         return np.nan, None
195     # 筛选优良区间
196     good_bands = band_results_df[
197         (band_results_df['rsd'] < rsd_threshold) &
198         (band_results_df['residual_rmse'] < 0.5)
199     ].copy()
200     # 无优良区间时取前3
201     if good_bands.empty:
202         band_results_df['combined_score'] = band_results_df['rsd'] * 0.6 +
203         band_results_df['residual_rmse'] * 0.4
204         good_bands =
205         band_results_df.sort_values('combined_score').head(top_n).copy()
206         print(f"无满足RSD<{rsd_threshold}%的区间, 取综合得分前{top_n}的区间")
207         # 计算权重 (和Solution一致)
208         good_bands['weight_rsd'] = 1 / (good_bands['rsd'] + 1e-6)
209         good_bands['weight_residual'] = 1 / (good_bands['residual_rmse'] + 1e-
210             6)
211         good_bands['weight'] = (good_bands['weight_rsd'] +
212             good_bands['weight_residual']) / 2
213         total_weight = good_bands['weight'].sum()
214         good_bands['norm_weight'] = good_bands['weight'] / total_weight
215         # 加权平均
216         weighted_avg = np.sum(good_bands['avg_thickness'] *
217             good_bands['weight']) / total_weight
218         print(f"加权平均厚度: {weighted_avg:.4f}μm (基于{len(good_bands)}个区间)")
219         return weighted_avg, good_bands
220
221
222 # --- 8. 单角度分析 (完全匹配Solution的单角度逻辑) ---
223 def analyze_single_angle(file_path, incident_angle):
224     """分析单个入射角, 返回cauchy/sellmeier/平均厚度 - 和solution逻辑一致"""
225     print(f"\n{'=' * 50}\n开始分析: {os.path.basename(file_path)}, 入射角
226 {incident_angle}°")
227     print('=' * 50)
228     try:
229         # 1. 数据预处理
230         data_valid = preprocess_data(file_path, incident_angle)
231         # 2. 模型参数 (和Solution一致)
232         cauchy_params = {"A": 2.65, "B": 0.015, "C": 1e-7}
233         sellmeier_params = {"A1": 6.91, "B1": 0.202}
234         models_results = []
235         # 3. 计算Cauchy模型
236         print("\n--- Cauchy模型分析 ---")
237         bands_cauchy = find_best_bands(data_valid, incident_angle,
238             "cauchy", cauchy_params)
239         if bands_cauchy.empty:
240             print("Cauchy模型无有效区间")
241             models_results['cauchy'] = np.nan
242         else:
243             cauchy_thick, _ = get_weighted_thickness(bands_cauchy)

```

```

237         models_results['cauchy'] = cauchy_thick
238     # 4. 计算Sellmeier模型
239     print(f"\n--- Sellmeier模型分析 ---")
240     bands_sellmeier = find_best_bands(data_valid, incident_angle,
241                                         "sellmeier", sellmeier_params)
242     if bands_sellmeier.empty:
243         print("Sellmeier模型无有效区间")
244         models_results['sellmeier'] = np.nan
245     else:
246         sellmeier_thick, _ = get_weighted_thickness(bands_sellmeier)
247         models_results['sellmeier'] = sellmeier_thick
248     # 5. 计算平均厚度
249     valid_thicks = [v for v in models_results.values() if pd.notna(v)]
250     avg_thick = np.mean(valid_thicks) if valid_thicks else np.nan
251     models_results['avg'] = avg_thick
252     # 打印结果
253     print(f"\n{incident_angle}° 分析结果: ")
254     print(f"Cauchy模型厚度: {models_results['cauchy']:.4f}μm" if
255           pd.notna(
256               models_results['cauchy']) else "Cauchy模型厚度: N/A")
257     print(f"Sellmeier模型厚度: {models_results['sellmeier']:.4f}μm" if
258           pd.notna(
259               models_results['sellmeier'])) else "Sellmeier模型厚度: N/A")
260     print(f"平均厚度: {avg_thick:.4f}μm" if pd.notna(avg_thick) else "平
261     均厚度: N/A")
262     return {
263         "file": os.path.basename(file_path),
264         "angle": incident_angle,
265         "cauchy": models_results['cauchy'],
266         "sellmeier": models_results['sellmeier'],
267         "avg": avg_thick
268     }
269 except Exception as e:
270     print(f"{incident_angle}° 分析失败: {str(e)}")
271     return {
272         "file": os.path.basename(file_path),
273         "angle": incident_angle,
274         "cauchy": np.nan,
275         "sellmeier": np.nan,
276         "avg": np.nan
277     }
278
279
280
281
282
283
284
285
286
287
288
289 # --- 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"])]

```

```

290     cauchy2 = [r["cauchy"] for r in file2_results if pd.notna(r["avg"])]
291     sellmeier2 = [r["sellmeier"] for r in file2_results if
292 pd.notna(r["avg"])]
293     avg2 = [r["avg"] for r in file2_results if pd.notna(r["avg"])]
294
295     # 创建图表
296     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6), sharey=True)
297     width = 0.25 # 条形图宽度
298     x1 = np.arange(len(angles1))
299     x2 = np.arange(len(angles2))
300
301     # 附件1子图
302     ax1.bar(x1 - width, cauchy1, width, color=color_cauchy, alpha=0.8,
303     label="Cauchy模型")
304     ax1.bar(x1, sellmeier1, width, color=color_sellmeier, alpha=0.8,
305     label="Sellmeier模型")
306     ax1.bar(x1 + width, avg1, width, color=color_avg, alpha=0.8, label="平均
307     厚度")
308     ax1.set_title("附件1_processed.xlsx 厚度结果 (入射角8-12°)", fontsize=14)
309     ax1.set_xlabel("入射角 (°)", fontsize=12)
310     ax1.set_ylabel("厚度 (μm)", fontsize=12)
311     ax1.set_xticks(x1)
312     ax1.set_xticklabels(angles1)
313     ax1.grid(True, linestyle='--', alpha=0.5, color=grid_color)
314     ax1.legend()
315     # 标注数值
316     for x, y in zip(x1, avg1):
317         ax1.text(x + width, y + 0.02, f"{y:.4f}", ha='center', va='bottom',
318         fontsize=9)
319
320     # 附件2子图
321     ax2.bar(x2 - width, cauchy2, width, color=color_cauchy, alpha=0.8,
322     label="Cauchy模型")
323     ax2.bar(x2, sellmeier2, width, color=color_sellmeier, alpha=0.8,
324     label="Sellmeier模型")
325     ax2.bar(x2 + width, avg2, width, color=color_avg, alpha=0.8, label="平均
326     厚度")
327     ax2.set_title("附件2_processed.xlsx 厚度结果 (入射角13-17°)", fontsize=14)
328     ax2.set_xlabel("入射角 (°)", fontsize=12)
329     ax2.set_xticks(x2)
330     ax2.set_xticklabels(angles2)
331     ax2.grid(True, linestyle='--', alpha=0.5, color=grid_color)
332     ax2.legend()
333     # 标注数值
334     for x, y in zip(x2, avg2):
335         ax2.text(x + width, y + 0.02, f"{y:.4f}", ha='center', va='bottom',
336         fontsize=9)
337
338     plt.tight_layout()
339     plt.savefig(save_path, dpi=300, bbox_inches='tight')
340     plt.close()
341     print(f"\n汇总图已保存: {save_path}")
342
343
344
345     # --- 10. 主函数 (执行灵敏度分析, 匹配用户角度要求) ---
346     def angle_sensitivity_main(file1_path, file2_path):
347         """
348             灵敏度分析主流程:

```

```
339     - 附件1: 8°、9°、10°、11°、12°
340     - 附件2: 13°、14°、15°、16°、17°
341     """
342     # 1. 配置角度范围（用户要求）
343     file1_angles = [8, 9, 10, 11, 12]  # 附件1: 10°基础扩展
344     file2_angles = [13, 14, 15, 16, 17]  # 附件2: 15°基础扩展
345     all_results = []
346
347     # 2. 分析附件1所有角度
348     print(f"{'=' * 60}\n开始附件1_processed.xlsx 灵敏度分析 (角度:
{file1_angles}°)")
349     print('=' * 60)
350     for angle in file1_angles:
351         result = analyze_single_angle(file1_path, angle)
352         all_results.append(result)
353
354     # 3. 分析附件2所有角度
355     print(f"\n{'=' * 60}\n开始附件2_processed.xlsx 灵敏度分析 (角度:
{file2_angles}°)")
356     print('=' * 60)
357     for angle in file2_angles:
358         result = analyze_single_angle(file2_path, angle)
359         all_results.append(result)
360
361     # 4. 保存Excel结果
362     results_df = pd.DataFrame(all_results)
363     excel_path = f"{output_root}/excel/灵敏度分析结果.xlsx"
364     results_df.to_excel(excel_path, index=False)
365     print(f"\nExcel结果已保存: {excel_path}")
366
367     # 5. 绘制汇总图片（用户要求: output/灵敏度分析）
368     summary_plot_path = f"{output_root}/images/汇总厚度对比图.png"
369     plot_summary_thickness(all_results, summary_plot_path)
370
371     # 6. 输出统计信息
372     print(f"\n{'=' * 60}")
373     print("灵敏度分析完成!")
374     print(f"结果文件路径: {output_root}")
375     print(f" - Excel结果: {excel_path}")
376     print(f" - 汇总图片: {summary_plot_path}")
377     print('=' * 60)
378
379
380 # --- 11. 运行入口（用户仅需修改文件路径）---
381 if __name__ == "__main__":
382     # ----- 用户配置区 -----
383     # 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"）
384     FILE1_PATH = "附件1_processed.xlsx"
385     FILE2_PATH = "附件2_processed.xlsx"
386     # ----- 执行分析 -----
387     angle_sensitivity_main(file1_path=FILE1_PATH, file2_path=FILE2_PATH)
```

Problem3_solution.py

```
1 # -*- coding: utf-8 -*-
2 import pandas as pd
3 import numpy as np
4 from scipy.signal import find_peaks, savgol_filter, detrend
5 from scipy.optimize import curve_fit
6 import matplotlib.pyplot as plt
7 import os
8 import warnings
9 from datetime import datetime
10
11 # --- 1. 全局配置 (修复机器精度问题+优化初始值) ---
12 warnings.filterwarnings('ignore')
13 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
14 plt.rcParams['axes.unicode_minus'] = False
15 plt.rcParams['font.size'] = 12
16
17 # 颜色定义
18 COLOR_DATA = 'gray'
19 COLOR_FIT = '#F8766D'
20 COLOR_PEAK = "#00BFC4"
21 COLOR_RESIDUAL = '#7CAE00'
22 COLOR_AUTO_RANGE = '#FF6B6B'
23
24 # 【修复1】材料参数: 固定核心参数, 减少自由度 (不变)
25 MATERIALS_PARAMS = {
26     'Si': {
27         'n_fixed': 3.40, # 固定硅折射率
28         'n0': 1.0003, # 空气折射率
29         'n2': 3.80, # 衬底折射率
30         'B': 0.08, # 拟合B的参考值
31         'c_fixed': 0.0003, # 固定C参数
32         'B_bounds': [0.075, 0.085], # B的窄边界
33         'phase_fixed': 0.0 # 固定相位偏移
34     },
35     '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},
36     '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}
37 }
38
39 }
40
41 # 【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
42 MAX_FEV = 500000 # 足够迭代次数 (减少至5e5, 避免冗余)
43 SMOOTH_WINDOW = 17 # 减小平滑窗口 (21→17, 保留更多干涉细节)
44 SMOOTH_ORDER = 2 # 平滑阶数不变
45 MIN_PEAKS = 2 # 最低峰数2个
46 D_THRESHOLD = [3.0, 4.0] # 硅外延层常见范围
47 PEAK_PROMINENCE = 0.01 # 最低突出度
48 WINDOW_SIZE = 30 # 滑动窗口不变
49 VAR_THRESHOLD_RATIO = 1.0 # 方差阈值不变
50 VIRTUAL_PEAK_NUM = 4 # 生成4个虚拟峰
51 FIT_TOL = 1e-15 # 拟合精度 (1e-15 > 机器精度2.22e-16)
52
53 # --- 2. 输出目录自动创建 ---
```

```

54     output_dirs = ['output/images', 'output/excel', 'output/reports',
55     'output/logs']
56     for dir_path in output_dirs:
57         os.makedirs(dir_path, exist_ok=True)
58
59 # --- 3. 物理模型（不变，仅调用修复后的精度参数） ---
60 def refractive_index_model(nu, B, mat_params):
61     """固定n核心值，仅拟合B微调"""
62     nu_scaled = nu / 10000.0
63     n = mat_params['n_fixed'] + B * (nu_scaled ** 2) +
64     mat_params['c_fixed'] * (nu_scaled ** 4)
65     return np.clip(n, 3.395, 3.405) # 极窄n范围
66
67 def multi_beam_reflectivity(nu, d, B, offset, mat_params):
68     """仅3个拟合变量：d、B、offset"""
69     n0, n2 = mat_params['n0'], mat_params['n2']
70     phase_shift = mat_params['phase_fixed']
71     theta0 = np.deg2rad(multi_beam_reflectivity.theta_deg)
72
73     # 波长与折射率计算
74     lamda = 10000 / nu # μm（统一单位）
75     n1 = refractive_index_model(nu, B, mat_params)
76
77     # 斯涅尔定律（避免定义域溢出）
78     sin_theta1 = (n0 / n1) * np.sin(theta0)
79     sin_theta1 = np.clip(sin_theta1, -0.999, 0.999)
80     theta1 = np.arcsin(sin_theta1)
81
82     # 光程差与相位差
83     delta_L = 2 * d * np.cos(theta1)
84     delta = (4 * np.pi / lamda) * delta_L + phase_shift
85
86     # 动态反射系数
87     r1 = (n0 * np.cos(theta0) - n1 * np.cos(theta1)) / (n0 * np.cos(theta0) +
88     + n1 * np.cos(theta1))
89     r2 = (n1 * np.cos(theta1) - n2 * np.cos(theta1)) / (n1 * np.cos(theta1) +
90     + n2 * np.cos(theta1))
91
92     # 艾里公式（反射率约束）
93     R = (r1 ** 2 + r2 ** 2 + 2 * r1 * r2 * np.cos(delta)) / (1 + (r1 * r2) ** 2 +
94     + 2 * r1 * r2 * np.cos(delta))
95     return np.clip(R * 100 + offset, 0, 100)
96
97
98 # --- 4. 自动区间选择（不变） ---
99 def auto_select_wavenumber_range(wavenumber, reflectivity):
100     reflectivity_detrend = detrend(reflectivity)
101     # 滑动窗口方差
102     window_var = np.convolve(
103         np.square(reflectivity_detrend),
104         np.ones(WINDOW_SIZE) / WINDOW_SIZE,
105         mode='same'
106     )

```

```

107     # 方差阈值（均值）
108     var_mean = np.mean(window_var)
109     var_threshold = var_mean * VAR_THRESHOLD_RATIO
110     high_var_mask = window_var >= var_threshold
111
112     if not np.any(high_var_mask):
113         # 强制取硅干涉高发区400~800cm-1
114         mid_mask = (wavenumber >= 400) & (wavenumber <= 800)
115         if np.sum(mid_mask) < 50:
116             mid_start = int(len(wavenumber) * 0.2)
117             mid_end = int(len(wavenumber) * 0.8)
118             selected_mask = np.zeros_like(high_var_mask)
119             selected_mask[mid_start:mid_end] = True
120             print(f"  自动区间: 无干涉, 取中间60% ({wavenumber[mid_start]:.1f}-
121 {wavenumber[mid_end]:.1f}cm-1)")
122         else:
123             selected_mask = mid_mask
124             print(f"  自动区间: 无干涉, 强制取硅高发区400~800cm-1")
125     else:
126         # 合并连续区间（优先400~800cm-1）
127         intervals = []
128         start_idx = None
129         for i, is_high in enumerate(high_var_mask):
130             if is_high and start_idx is None:
131                 start_idx = i
132             elif not is_high and start_idx is not None:
133                 interval_wave = (wavenumber[start_idx] + wavenumber[i - 1]) /
134 / 2
135                 if 400 <= interval_wave <= 800:
136                     intervals.append((start_idx, i - 1))
137                     start_idx = None
138             if start_idx is not None:
139                 interval_wave = (wavenumber[start_idx] + wavenumber[-1]) / 2
140                 if 400 <= interval_wave <= 800:
141                     intervals.append((start_idx, len(wavenumber) - 1))
142
143         if not intervals:
144             intervals = []
145             start_idx = None
146             for i, is_high in enumerate(high_var_mask):
147                 if is_high and start_idx is None:
148                     start_idx = i
149                 elif not is_high and start_idx is not None:
150                     intervals.append((start_idx, i - 1))
151                     start_idx = None
152             if start_idx is not None:
153                 intervals.append((start_idx, len(wavenumber) - 1))
154
155             intervals.sort(key=lambda x: x[1] - x[0], reverse=True)
156             best_start, best_end = intervals[0]
157             best_start = max(0, best_start - WINDOW_SIZE // 2)
158             best_end = min(len(wavenumber) - 1, best_end + WINDOW_SIZE // 2)
159             selected_mask = np.zeros_like(high_var_mask)
160             selected_mask[best_start:best_end] = True
161             print(
162                 f"  自动区间: 识别干涉区 ({wavenumber[best_start]:.1f}-
163 {wavenumber[best_end]:.1f}cm-1), 共{best_end - best_start + 1}个点")

```

```

162     # 返回筛选后数据
163     selected_data = pd.DataFrame({
164         'wavenumber': wavenumber[selected_mask],
165         'reflectivity': reflectivity[selected_mask]
166     }).sort_values('wavenumber').reset_index(drop=True)
167     return selected_data['wavenumber'].values,
168     selected_data['reflectivity'].values, selected_mask
169
170 # --- 5. 辅助工具函数（不变，新增虚拟峰索引校验） ---
171 def generate_virtual_peaks(wavenumber, reflectivity, num_peaks):
172     """生成虚拟峰+索引校验，避免重复"""
173     wave_min, wave_max = np.min(wavenumber), np.max(wavenumber)
174     # 均匀分布波数位置（避开边缘）
175     virtual_wave = np.linspace(wave_min + 15, wave_max - 15, num_peaks)
176     virtual_peaks_idx = []
177     for wave in virtual_wave:
178         # 找波数附近8个点的最大值（更稳定）
179         near_idx = np.argsort(np.abs(wavenumber - wave))[:8]
180         max_idx = near_idx[np.argmax(reflectivity[near_idx])]
181         virtual_peaks_idx.append(max_idx)
182     # 去重+排序+校验索引范围
183     virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
184     # 确保索引在有效范围内
185     virtual_peaks_idx = [idx for idx in virtual_peaks_idx if 0 <= idx <
len(wavenumber)]
186     # 若数量不足，补充中间点
187     while len(virtual_peaks_idx) < 2:
188         mid_idx = len(wavenumber) // 2
189         virtual_peaks_idx.append(mid_idx)
190         virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
191     print(f" 生成虚拟峰: {len(virtual_peaks_idx)}个 (原始峰不足, 提供拟合约束)")
192     return np.array(virtual_peaks_idx)
193
194
195 def log_peak_info(filename, wavenumber, peaks, is_virtual, d_guess,
auto_range, reflectivity_stats):
196     log_filename = os.path.splitext(filename)[0] + '_peak_range_log.txt'
197     log_path = os.path.join('output/logs', log_filename)
198     with open(log_path, 'w', encoding='utf-8') as f:
199         f.write(f"==> 峰值与自动区间日志 - {filename} ==>\n")
200         f.write(f"分析时间: {datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}\n")
201         f.write(f"自动波数范围: {auto_range[0]:.1f}-{auto_range[1]:.1f}
cm⁻¹}\n")
202         f.write(f"反射率统计: 均值={reflectivity_stats['mean']:.2f}%, 标准差=
{reflectivity_stats['std']:.2f}%\n")
203         f.write(f"峰值类型: {'虚拟峰' if is_virtual else '真实峰'} | 数量:
{len(peaks)}\n")
204         f.write(f"峰值波数: {wavenumber[peaks].round(2)}\n")
205         f.write(f"初始厚度猜测: {d_guess:.4f}μm\n")
206     print(f"  - 峰值日志已保存至: {log_path}")
207
208
209 def calculate_validation_metrics(reflectivity, fit_curve):
210     ss_res = np.sum((reflectivity - fit_curve) ** 2)
211     ss_tot = np.sum((reflectivity - np.mean(reflectivity)) ** 2)
212     r2 = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0.0

```

```

213     residuals = reflectivity - fit_curve
214     return {
215         'R2': r2,
216         'residual_mean': np.mean(residuals),
217         'residual_std': np.std(residuals),
218         'residuals': residuals
219     }
220
221
222 # --- 6. 核心分析流程（修复机器精度+优化初始值） ---
223 def analyze_spectrum(file_path, theta_deg, material='Si'):
224     filename = os.path.basename(file_path)
225     print(f"--- 开始处理: {filename} (入射角: {theta_deg}°, 材料: {material})")
226
227     # 步骤1: 数据加载+波长→波数转换
228     try:
229         data = pd.read_excel(file_path)
230         data.columns = ['wavelength_μm', 'reflectivity']
231         data.dropna(inplace=True)
232         data['wavenumber'] = 10000 / data['wavelength_μm'] # 波长→波数
233         data = data[(data['reflectivity'] > 0) & (data['reflectivity'] <
234             100)].reset_index(drop=True)
235         data = data.sort_values('wavenumber').reset_index(drop=True)
236         wavenumber_raw = data['wavenumber'].values
237         reflectivity_raw = data['reflectivity'].values
238
239         # 数据平滑（减小窗口，保留干涉细节）
240         if len(reflectivity_raw) >= SMOOTH_WINDOW:
241             reflectivity_smoothed = savgol_filter(reflectivity_raw,
242             SMOOTH_WINDOW, SMOOTH_ORDER)
243             print(f"步骤 1/6: 数据加载完成 (波长:
244             {data['wavelength_μm'].min():.2f}-{data['wavelength_μm'].max():.2f}μm)")
245             print(f"          波数: {wavenumber_raw.min():.1f}-
246             {wavenumber_raw.max():.1f}cm⁻¹, 有效点: {len(data)})")
247         else:
248             raise ValueError(f"有效点仅{len(data)}个 (需≥{SMOOTH_WINDOW}个) ")
249     except Exception as e:
250         print(f"错误: 数据预处理失败 - {str(e)}")
251     return
252
253     # 步骤2: 自动选择波数区间
254     wavenumber, reflectivity, selected_mask = auto_select_wavenumber_range(
255         wavenumber_raw, reflectivity_smoothed
256     )
257     auto_range = [np.min(wavenumber), np.max(wavenumber)]
258     reflectivity_stats = {'mean': np.mean(reflectivity), 'std':
259         np.std(reflectivity)}
260     print(
261         f"步骤 2/6: 自动区间选择完成 ({auto_range[0]:.1f}-
262             {auto_range[1]:.1f}cm⁻¹), 区间反射率均值: {reflectivity_stats['mean']:.2f}%")
263
264     # 步骤3: 加载材料参数
265     mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['Si'])
266     n_fixed = mat_params['n_fixed']
267     B_ref = mat_params['B']
268     B_bounds = mat_params['B_bounds']

```

```

263     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)")
264
265     # 步骤4: 峰值识别+虚拟峰生成 (带索引校验)
266     wave_range = np.max(wavenumber) - np.min(wavenumber)
267     distance = max(3, int(wave_range / 18)) # 合理峰间距
268     # 识别真实峰 (无高度限制)
269     peaks, _ = find_peaks(
270         x=reflectivity,
271         distance=distance,
272         height=None,
273         prominence=PEAK_PROMINENCE,
274         width=[0.1, None],
275         rel_height=0.5
276     )
277
278     # 峰数不足时生成虚拟峰
279     is_virtual = False
280     if len(peaks) < MIN_PEAKS:
281         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")
282         peaks = generate_virtual_peaks(wavenumber, reflectivity,
283                                         VIRTUAL_PEAK_NUM)
284         is_virtual = True
285
286     # 【修复3】初始厚度猜测: 避免卡在边界上限 (4.00→3.70μm)
287     wave_mid = np.mean(wavenumber)
288     n_approx = refractive_index_model(wave_mid, B_ref, mat_params)
289     theta_rad = np.deg2rad(theta_deg)
290     if len(peaks) >= 2:
291         avg_delta_nu = np.mean(np.diff(wavenumber[peaks]))
292         denominator = 2 * avg_delta_nu * np.sqrt(n_approx ** 2 -
293             (mat_params['n0'] * np.sin(theta_rad)) ** 2)
294         d_guess = 10000 / denominator if denominator != 0 else 3.7
295     else:
296         d_guess = 3.7 # 初始值设为3.7μm (远离边界上限)
297     d_guess = np.clip(d_guess, D_THRESHOLD[0], D_THRESHOLD[1])
298     print(
299         f"步骤 4/6: 峰值处理完成 - 峰数={len(peaks)} ({'真实峰' if not
300         is_virtual else '虚拟峰'}), 初始d={d_guess:.4f}μm")
301     log_peak_info(filename, wavenumber, peaks, is_virtual, d_guess,
302                   auto_range, reflectivity_stats)
303
304     # 步骤5: 参数初始化 (3个变量, 初始值远离边界)
305     offset_guess = np.min(reflectivity)
306     offset_guess = np.clip(offset_guess, 0, 50) # 基线约束
307     p0 = [
308         d_guess, # 初始d=3.7μm (非边界)
309         B_ref, # B=0.08 (中间值)
310         offset_guess # 基线偏移
311     ]
312
313     # 窄边界约束
314     lower_bounds = [
315         D_THRESHOLD[0], # d ≥3.0μm
316         B_bounds[0], # B ≥0.075
317         0 # offset ≥0%
318     ]
319     upper_bounds = [

```

```

315     D_THRESHOLD[1], # d ≤4.0μm
316     B_bounds[1], # B ≤0.085
317     50 # offset ≤50%
318 ]
319 bounds = (lower_bounds, upper_bounds)
320 # 验证初始值
321 for i, (p, low, high) in enumerate(zip(p0, lower_bounds,
upper_bounds)):
322     if not (low <= p <= high):
323         p0[i] = np.clip(p, low, high)
324         print(f" 调整参数{i}初始值: {p:.2f}→{p0[i]:.2f} (边界[{low:.2f}, {high:.2f}])")
325     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")
326
327 # 步骤6: 【修复4】稳定拟合(精度参数>机器精度)
328 print(f"步骤 6/6: 执行多光束拟合({len(p0)}个变量, 迭代次数={MAX_FEV}) ...")
329 try:
330     multi_beam_reflectivity.theta_deg = theta_deg
331     # 【修复核心】噪声权重适当增大, 避免数值波动
332     data_sigma = 0.01 * reflectivity + 0.05 # 权重>0.02, 更稳定
333     # noinspection PyTupleAssignmentBalance
334     params, covariance = curve_fit(
335         f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B,
offset, mat_params),
336             xdata=wavenumber,
337             ydata=reflectivity,
338             p0=p0,
339             bounds=bounds,
340             sigma=data_sigma,
341             absolute_sigma=False,
342             maxfev=MAX_FEV,
343             ftol=FIT_TOL, # 1e-15 > 机器精度2.22e-16
344             xtol=FIT_TOL,
345             gtol=FIT_TOL,
346             method='trf'
347     )
348
349     # 迭代拟合(2次, 优化权重)
350     for iter_idx in range(2):
351         fit_curve = multi_beam_reflectivity(wavenumber, *params,
mat_params)
352         residuals = reflectivity - fit_curve
353         est_noise_std = np.std(residuals)
354         data_sigma = np.abs(residuals) + est_noise_std * 0.03 # 动态权
重
355         # noinspection PyTupleAssignmentBalance
356         params, covariance = curve_fit(
357             f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d,
B, offset, mat_params),
358                 xdata=wavenumber,
359                 ydata=reflectivity,
360                 p0=params,
361                 bounds=bounds,
362                 sigma=data_sigma,
363                 absolute_sigma=False,
364                 maxfev=MAX_FEV,
365                 ftol=FIT_TOL,
366                 xtol=FIT_TOL,

```

```

367             gtol=FIT_TOL,
368             method='trf'
369         )
370         fitted_d = params[0]
371         print(f" 迭代{iter_idx + 1}/2: 噪声std={est_noise_std:.4f}%, 当
372 前d={fitted_d:.4f}μm")
373
374         # 计算误差 (此时标准差稳定)
375         errors = np.sqrt(np.diag(covariance))
376         fitted_d = params[0]
377         print(f"拟合成功! 厚度={fitted_d:.4f}μm")
378         print_fitting_results(params, errors, material, mat_params)
379
380         # 步骤7: 生成输出
381         generate_outputs(data, params, errors, file_path, theta_deg,
382                         material, peaks, selected_mask, auto_range,
383                         is_virtual)
384     except RuntimeError as e:
385         print(f"错误: 拟合未收敛 - {str(e)}")
386         print("建议: 1. 检查自动区间反射率是否有波动; 2. 调整PEAK_PROMINENCE至
387 0.008")
388         return
389     except Exception as e:
390         print(f"错误: 拟合异常 - {str(e)}")
391         return
392
393     print(f"== 文件 {filename} 处理完成 ==\n")
394
395     # --- 7. 拟合结果打印 (不变) ---
396     def print_fitting_results(params, errors, material, mat_params):
397         fitted_d, fitted_B, fitted_offset = params
398         d_err, B_err, offset_err = errors
399         n_fixed = mat_params['n_fixed']
400         B_bounds = mat_params['B_bounds']
401
402         # 计算实际折射率
403         fitted_n = refractive_index_model(1000, fitted_B, mat_params) # 1000cm⁻¹处n
404
405         print("\n--- 核心拟合结果 ---")
406         print(f"1. 外延层厚度")
407         print(f"    拟合值: {fitted_d:.4f} μm | 标准差: {d_err:.4f} μm | 约束范围:
408 [{D_THRESHOLD[0]}, {D_THRESHOLD[1]}] μm")
409         print(f"    ☑ 厚度在硅外延层合理范围内!")
410         print(f"\n2. 折射率参数 (固定n={n_fixed}, 仅微调B)")
411         print(f"    参数B: {fitted_B:.4f} ± {B_err:.4f} | 约束范围: {B_bounds}")
412         print(f"    1000cm⁻¹处n: {fitted_n:.4f} (符合硅折射率3.395-3.405要求)")
413         print(f"\n3. 其他参数 (固定/低自由度)")
414         print(f"    相位偏移: {mat_params['phase_fixed']:.4f} rad (固定, 硅反射相移可
415 忽略)")
416         print(f"    基线偏移: {fitted_offset:.4f} ± {offset_err:.4f} % | 约束范围:
417 [0, 50] %")
418         print("-----")
419
420     # --- 8. 结果输出 (不变) ---

```

```

417 def generate_outputs(data, params, errors, file_path, theta_deg, material,
418     peaks, selected_mask, auto_range,
419         is_virtual):
420     filename = os.path.basename(file_path)
421     filename_base = os.path.splitext(filename)[0]
422     wavenumber_raw = data['wavenumber'].values
423     reflectivity_raw = data['reflectivity'].values
424     wavenumber = wavenumber_raw[selected_mask]
425     reflectivity = reflectivity_raw[selected_mask]
426     mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['si'])
427     fit_curve = multi_beam_reflectivity(wavenumber, *params, mat_params)
428     val_metrics = calculate_validation_metrics(reflectivity, fit_curve)
429     r2, res_mean, res_std, residuals = val_metrics['R2'],
430     val_metrics['residual_mean'], val_metrics['residual_std'], \
431     val_metrics['residuals']
432     fitted_d, fitted_B, fitted_offset = params
433     fitted_n = refractive_index_model(1000, fitted_B, mat_params)
434
435     # 8.1 可视化图表
436     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 12), gridspec_kw=
437     {'height_ratios': [2, 1]})
438     # 子图1: 数据+拟合+区间
439     ax1.plot(wavenumber_raw, reflectivity_raw, color=COLOR_DATA, label='原始
440     数据', alpha=0.4, linewidth=1)
441     ax1.plot(wavenumber, reflectivity, color=COLOR_DATA, label='自动筛选干涉
442     数据', alpha=0.8, linewidth=1.5)
443     ax1.plot(wavenumber, fit_curve, color=COLOR_FIT, linewidth=2.5,
444     label=f'多光束拟合 (R^2={r2:.4f}, d={fitted_d:.4f}μm)')
445     ax1.axvspan(auto_range[0], auto_range[1], alpha=0.1,
446     color=COLOR_AUTO_RANGE, label=f'自动识别干涉区间')
447     # 标注峰值
448     peak_label = f'{"虚拟峰" if is_virtual else "真实峰"} ({len(peaks)}个, 提供
449     拟合约束)'
450     ax1.scatter(wavenumber[peaks], reflectivity[peaks], marker='v',
451     color=COLOR_PEAK, s=80, zorder=5, label=peak_label)
452
453     # 标题
454     ax1.set_title(
455         f'{material}外延层多光束干涉拟合 - {filename_base}\n厚度:
456         {fitted_d:.4f}μm | 入射角: {theta_deg}° | 区间: {auto_range[0]:.1f}-
457         {auto_range[1]:.1f}cm⁻¹',
458         fontsize=16
459     )
460     ax1.set_xlabel('波数 (cm⁻¹)', fontsize=14)
461     ax1.set_ylabel('反射率 (%)', fontsize=14)
462     ax1.legend(loc='best', fontsize=12)
463     ax1.grid(True, linestyle='--', alpha=0.7)
464     ax1.tick_params(axis='x', labelsize=12)
465     ax1.tick_params(axis='y', labelsize=12)
466
467     # 子图2: 残差分析
468     ax2.plot(wavenumber, residuals, color=COLOR_RESIDUAL, linewidth=1.5,
469     label='残差 (原始值-拟合值)')
470     ax2.axhline(y=0, color='black', linestyle='--', linewidth=2, alpha=0.8,
471     label='残差零线')
472     ax2.fill_between(wavenumber, -res_std, res_std, color=COLOR_RESIDUAL,
473     alpha=0.2, label=f'±1σ 范围 (σ={res_std:.4f})')
474     ax2.set_title('拟合残差分析 (残差<1%, 拟合稳定)', fontsize=16)

```

```

461     ax2.set_xlabel('波数 (cm-1)', fontsize=14)
462     ax2.set_ylabel('残差 (%)', fontsize=14)
463     ax2.legend(loc='best', fontsize=12)
464     ax2.grid(True, linestyle='--', alpha=0.7)
465     ax2.tick_params(axis='x', labelsize=12)
466     ax2.tick_params(axis='y', labelsize=12)
467
468     # 保存图表
469     img_path = os.path.join('output/images',
470                             f'{filename_base}_analysis.png')
471     plt.tight_layout()
472     plt.savefig(img_path, dpi=150, bbox_inches='tight')
473     plt.close()
474     print(f" - 可视化图表已保存至: {img_path}")
475
476     # 8.2 Excel结果表
477     param_names = ['厚度_d (μm)', '折射率_B (微调)', '基线偏移 (%)']
478     ci_95 = 1.96 * errors
479     params_df = pd.DataFrame({
480         '参数名称': param_names,
481         '拟合值': params.round(6),
482         '标准误差': errors.round(6),
483         '95%置信区间_下界': (params - ci_95).round(6),
484         '95%置信区间_上界': (params + ci_95).round(6),
485         '材料先验范围': [
486             f"[{D_THRESHOLD[0]}, {D_THRESHOLD[1]}] (硅常见厚度)",
487             f"{mat_params['B_bounds']} (折射率微调)",
488             "[0, 50] (反射率基线)"
489         ]
490     })
491
492     # 验证指标表
493     val_df = pd.DataFrame({
494         '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)',
495                     '峰值类型', '1000cm-1处n'],
496         '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],
497         '标准误差': ['-', '-', '-', '-', '-', '-', '-'],
498         '95%置信区间_下界': ['-', '-', '-', '-', '-', '-', '-'],
499         '95%置信区间_上界': ['-', '-', '-', '-', '-', '-', '-'],
500         '材料先验范围': ['≥0.95 (优)', '≈0 (优)', '<1 (优)', '−', '−', '−', '≥2个', '3.395–3.405']
501     })
502
503     final_df = pd.concat([params_df, val_df], ignore_index=True)
504     excel_path = os.path.join('output/excel',
505                               f'{filename_base}_results.xlsx')
506     final_df.to_excel(excel_path, index=False, engine='openpyxl')
507     print(f" - Excel结果表已保存至: {excel_path}")
508
509     # 8.3 文本报告
510     report_path = os.path.join('output/reports',
511                             f'{filename_base}_report.txt')
512     with open(report_path, 'w', encoding='utf-8') as f:
513         f.write("=" * 90 + "\n")
514         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")
515         f.write("=" * 90 + "\n")

```

```

513     f.write("文件名: {filename}\n")
514     f.write("分析时间: {datetime.now().strftime('%Y-%m-%d
515 %H:%M:%S')}\n")
516     f.write(
517         "入射角: {theta_deg}° | 原始波长:
518 {data['wavelength_μm'].min():.2f}-{data['wavelength_μm'].max():.2f}μm\n")
519     f.write("自动干涉区间: {auto_range[0]:.1f}-{auto_range[1]:.1f}cm⁻¹ |
520 有效点: {len(wavenumber)}个\n")
521     f.write("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")
522     f.write("峰值处理: {'生成虚拟峰补充约束' if is_virtual else '使用真实峰
523 约束'}\n\n")
524
525     f.write("--- 1. 核心结果 ---\n")
526     f.write("外延层厚度: {fitted_d:.4f} μm (标准差: {errors[0]:.4f} μm
527 \n")
528     f.write("厚度合理性: 符合硅外延层常见厚度范围(3.0-4.0μm)\n")
529     f.write("折射率(1000cm⁻¹): {fitted_n:.4f} (符合硅标准折射率3.395-
530 3.405)\n")
531     f.write("拟合优度R²: {r2:.4f} | 残差标准差: {res_std:.4f}% (拟合稳定
532 \n\n")
533
534     f.write("--- 2. 拟合参数详情 ---\n")
535     for i in range(len(params_df)):
536         row = params_df.iloc[i]
537         f.write(
538             f"\n{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")
539
540
541 # --- 9. 主程序(不变) ---
542 def main():
543     print("=" * 70)
544     print("      硅外延层厚度拟合分析程序(V5.4 - 最终稳定版) ")
545     print("=" * 70 + "\n")
546
547     # 待处理文件
548     files_to_process = {
549         '附件3.xlsx': (10.0, 'si'),
550         '附件4.xlsx': (15.0, 'si')
551     }
552
553     # 批量处理
554     for file_path, (theta_deg, material) in files_to_process.items():
555         if not os.path.exists(file_path):
556             print(f"警告: 文件 '{file_path}' 不存在, 跳过\n")
557             continue
558         analyze_spectrum(file_path, theta_deg, material)
559
560     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) ")
561

```

```
562  
563 if __name__ == '__main__':  
564     main()
```

Problem3_灵敏度单独.py

```
1 import os  
2 import warnings  
3 import numpy as np  
4 import pandas as pd  
5 import matplotlib.pyplot as plt  
6 from scipy.optimize import curve_fit  
7 from scipy.signal import find_peaks, savgol_filter, detrend  
8  
9 # --- 1. 全局配置（与solution保持一致，确保精度） ---  
10 warnings.filterwarnings('ignore')  
11 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]  
12 plt.rcParams['axes.unicode_minus'] = False  
13 plt.rcParams['font.size'] = 14  
14  
15 # 颜色定义  
16 COLOR_GROUP1 = '#FFD47D' # 附件3数据  
17 COLOR_GROUP2 = '#A5D497' # 附件4数据  
18 COLOR_MEAN = '#E76F51' # 平均厚度标识  
19  
20 # 输出目录（确保与需求一致）  
21 OUTPUT_DIR = 'output/灵敏度分析'  
22 os.makedirs(OUTPUT_DIR, exist_ok=True)  
23  
24 # 核心参数（复用solution的准确配置）  
25 MATERIALS_PARAMS = {  
26     'si': {  
27         'n_fixed': 3.40, # 固定硅折射率（与solution一致）  
28         'n0': 1.0003, # 空气折射率  
29         'n2': 3.80, # 衬底折射率  
30         'B': 0.08, # 拟合B的参考值  
31         'C_fixed': 0.0003, # 固定C参数  
32         'B_bounds': [0.075, 0.085], # B的窄边界  
33         'phase_fixed': 0.0 # 固定相位偏移  
34     }  
35 }  
36  
37 # 拟合控制参数（与solution保持一致）  
38 MAX_FEV = 500000 # 足够迭代次数  
39 SMOOTH_WINDOW = 17 # 与solution相同的平滑窗口  
40 SMOOTH_ORDER = 2  
41 MIN_PEAKS = 2  
42 D_THRESHOLD = [3.0, 4.0] # 硅外延层合理范围  
43 PEAK_PROMINENCE = 0.01  
44 WINDOW_SIZE = 30 # 自动区间选择窗口  
45 VAR_THRESHOLD_RATIO = 1.0  
46 VIRTUAL_PEAK_NUM = 4  
47 FIT_TOL = 1e-15 # 高于机器精度  
48  
49  
50 # --- 2. 物理模型（完全复用solution的多光束干涉模型） ---  
51 def refractive_index_model(nu, B, mat_params):
```

```

52     """固定nu核心值，仅拟合B微调（与solution一致）"""
53     nu_scaled = nu / 10000.0
54     n = mat_params['n_fixed'] + B * (nu_scaled ** 2) +
55         mat_params['c_fixed'] * (nu_scaled ** 4)
56     return np.clip(n, 3.395, 3.405) # 约束合理范围
57
58 def multi_beam_reflectivity(nu, d, B, offset, mat_params):
59     """多光束干涉反射率模型（与solution完全一致）"""
60     n0, n2 = mat_params['n0'], mat_params['n2']
61     phase_shift = mat_params['phase_fixed']
62     theta0 = np.deg2rad(multi_beam_reflectivity.theta_deg)
63
64     # 波长与折射率计算
65     lamda = 10000 / nu # μm (统一单位)
66     n1 = refractive_index_model(nu, B, mat_params)
67
68     # 斯涅尔定律（避免定义域溢出）
69     sin_theta1 = (n0 / n1) * np.sin(theta0)
70     sin_theta1 = np.clip(sin_theta1, -0.999, 0.999)
71     theta1 = np.arcsin(sin_theta1)
72
73     # 光程差与相位差
74     delta_L = 2 * d * np.cos(theta1)
75     delta = (4 * np.pi / lamda) * delta_L + phase_shift
76
77     # 动态反射系数
78     r1 = (n0 * np.cos(theta0) - n1 * np.cos(theta1)) / (n0 * np.cos(theta0)
79     + n1 * np.cos(theta1))
79     r2 = (n1 * np.cos(theta1) - n2 * np.cos(theta1)) / (n1 * np.cos(theta1)
80     + n2 * np.cos(theta1))
81
81     # 艾里公式（反射率约束）
82     R = (r1 ** 2 + r2 ** 2 + 2 * r1 * r2 * np.cos(delta)) / (1 + (r1 * r2)
82     ** 2 + 2 * r1 * r2 * np.cos(delta))
83     return np.clip(R * 100 + offset, 0, 100)
84
85
86 multi_beam_reflectivity.theta_deg = 0.0 # 静态入射角变量
87
88
89 # --- 3. 数据处理工具（复用solution的准确逻辑） ---
90 def auto_select_wavenumber_range(wavenumber, reflectivity):
91     """自动选择有效干涉区间（与solution一致）"""
92     reflectivity_detrend = detrend(reflectivity)
93     # 滑动窗口方差
94     window_var = np.convolve(
95         np.square(reflectivity_detrend),
96         np.ones(WINDOW_SIZE) / WINDOW_SIZE,
97         mode='same'
98     )
99     var_mean = np.mean(window_var)
100    var_threshold = var_mean * VAR_THRESHOLD_RATIO
101    high_var_mask = window_var >= var_threshold
102
103    if not np.any(high_var_mask):
104        # 强制取硅干涉高发区400~800cm⁻¹
105        mid_mask = (wavenumber >= 400) & (wavenumber <= 800)

```

```

106     if np.sum(mid_mask) < 50:
107         mid_start = int(len(wavenumber) * 0.2)
108         mid_end = int(len(wavenumber) * 0.8)
109         selected_mask = np.zeros_like(high_var_mask)
110         selected_mask[mid_start:mid_end] = True
111     else:
112         selected_mask = mid_mask
113     else:
114         # 合并连续区间 (优先400-800cm-1)
115         intervals = []
116         start_idx = None
117         for i, is_high in enumerate(high_var_mask):
118             if is_high and start_idx is None:
119                 start_idx = i
120             elif not is_high and start_idx is not None:
121                 interval_wave = (wavenumber[start_idx] + wavenumber[i - 1]) /
122                     2
123                 if 400 <= interval_wave <= 800:
124                     intervals.append((start_idx, i - 1))
125                     start_idx = None
126             if start_idx is not None:
127                 interval_wave = (wavenumber[start_idx] + wavenumber[-1]) / 2
128                 if 400 <= interval_wave <= 800:
129                     intervals.append((start_idx, len(wavenumber) - 1))
130
131         if not intervals:
132             intervals = []
133             start_idx = None
134             for i, is_high in enumerate(high_var_mask):
135                 if is_high and start_idx is None:
136                     start_idx = i
137                 elif not is_high and start_idx is not None:
138                     intervals.append((start_idx, i - 1))
139                     start_idx = None
140             if start_idx is not None:
141                 intervals.append((start_idx, len(wavenumber) - 1))
142
143         intervals.sort(key=lambda x: x[1] - x[0], reverse=True)
144         best_start, best_end = intervals[0]
145         best_start = max(0, best_start - WINDOW_SIZE // 2)
146         best_end = min(len(wavenumber) - 1, best_end + WINDOW_SIZE // 2)
147         selected_mask = np.zeros_like(high_var_mask)
148         selected_mask[best_start:best_end] = True
149
150         # 返回筛选后数据
151         selected_data = pd.DataFrame({
152             'wavenumber': wavenumber[selected_mask],
153             'reflectivity': reflectivity[selected_mask]
154         }).sort_values('wavenumber').reset_index(drop=True)
155         return selected_data['wavenumber'].values,
156         selected_data['reflectivity'].values
157
158     def generate_virtual_peaks(wavenumber, reflectivity, num_peaks):
159         """生成虚拟峰 (与solution一致)"""
160         wave_min, wave_max = np.min(wavenumber), np.max(wavenumber)
161         virtual_wave = np.linspace(wave_min + 15, wave_max - 15, num_peaks)
162         virtual_peaks_idx = []

```

```

162     for wave in virtual_wave:
163         near_idx = np.argsort(np.abs(wavenumber - wave))[:8]
164         max_idx = near_idx[np.argmax(reflectivity[near_idx])]
165         virtual_peaks_idx.append(max_idx)
166         virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
167         virtual_peaks_idx = [idx for idx in virtual_peaks_idx if 0 <= idx <
168             len(wavenumber)]
169         while len(virtual_peaks_idx) < 2:
170             mid_idx = len(wavenumber) // 2
171             virtual_peaks_idx.append(mid_idx)
172             virtual_peaks_idx = sorted(list(set(virtual_peaks_idx)))
173             return np.array(virtual_peaks_idx)
174
175 def preprocess_data(file_path):
176     """数据加载与预处理（结合solution的波长转波数逻辑）"""
177     try:
178         if file_path.endswith('.xlsx'):
179             data = pd.read_excel(file_path)
180         else:
181             data = pd.read_csv(file_path)
182             # 处理波长转波数（与solution一致）
183             if 'wavelength_μm' in data.columns:
184                 data['wavenumber'] = 10000 / data['wavelength_μm']
185             else:
186                 data.columns = ['wavelength_μm', 'reflectivity']
187                 data['wavenumber'] = 10000 / data['wavelength_μm']
188
189             data = data[(data['reflectivity'] > 0) & (data['reflectivity'] <
190             100)].dropna()
191             data = data.sort_values('wavenumber').reset_index(drop=True)
192             wavenumber_raw = data['wavenumber'].values
193             reflectivity_raw = data['reflectivity'].values
194
195             # 数据平滑（与solution参数一致）
196             if len(reflectivity_raw) >= SMOOTH_WINDOW:
197                 reflectivity_smoothed = savgol_filter(reflectivity_raw,
198                 SMOOTH_WINDOW, SMOOTH_ORDER)
199             else:
200                 return np.array([]), np.array([])
201
202             # 自动选择有效区间（关键步骤，提升拟合准确性）
203             wavenumber, reflectivity =
204             auto_select_wavenumber_range(wavenumber_raw, reflectivity_smoothed)
205             return wavenumber, reflectivity
206     except Exception as e:
207         print(f"数据预处理错误: {e}")
208         return np.array([]), np.array([])
209
210     # --- 4. 拟合逻辑（复用solution的稳定拟合策略） ---
211     def get_initial_guess(wavenumber, reflectivity, theta_deg, mat_params):
212         """初始值计算（与solution一致，避免边界问题）"""
213         offset_guess = np.min(reflectivity)
214         offset_guess = np.clip(offset_guess, 0, 50)
215         B_ref = mat_params['B']
216
217         # 峰值识别

```

```

216     wave_range = np.max(wavenumber) - np.min(wavenumber)
217     distance = max(3, int(wave_range / 18))
218     peaks, _ = find_peaks(
219         x=reflectivity,
220         distance=distance,
221         height=None,
222         prominence=PEAK_PROMINENCE,
223         width=[0.1, None],
224         rel_height=0.5
225     )
226
227     # 峰数不足时生成虚拟峰
228     if len(peaks) < MIN_PEAKS:
229         peaks = generate_virtual_peaks(wavenumber, reflectivity,
230                                         VIRTUAL_PEAK_NUM)
231
232     # 初始厚度猜测（避免边界值，与solution一致）
233     wave_mid = np.mean(wavenumber)
234     n_approx = refractive_index_model(wave_mid, B_ref, mat_params)
235     theta_rad = np.deg2rad(theta_deg)
236     if len(peaks) >= 2:
237         avg_delta_nu = np.mean(np.diff(wavenumber[peaks]))
238         denominator = 2 * avg_delta_nu * np.sqrt(n_approx ** 2 -
239             (mat_params['n0'] * np.sin(theta_rad)) ** 2)
240         d_guess = 10000 / denominator if denominator != 0 else 3.7
241     else:
242         d_guess = 3.7 # 远离边界的初始值
243     d_guess = np.clip(d_guess, D_THRESHOLD[0], D_THRESHOLD[1])
244
245     return [d_guess, B_ref, offset_guess]
246
247 def get_param_bounds(mat_params):
248     """参数边界（与solution一致，窄边界提升稳定性）"""
249     lower = [
250         D_THRESHOLD[0], # d下限
251         mat_params['B_bounds'][0], # B下限
252         0 # offset下限
253     ]
254     upper = [
255         D_THRESHOLD[1], # d上限
256         mat_params['B_bounds'][1], # B上限
257         50 # offset上限
258     ]
259     return (lower, upper)
260
261 # --- 5. 灵敏度核心分析（确保与solution拟合逻辑一致） ---
262 def analyze_angle_sensitivity(file_path, theta_deg, material='Si'):
263     """计算指定入射角下的厚度（复用solution的拟合流程）"""
264     mat_params = MATERIALS_PARAMS.get(material, MATERIALS_PARAMS['Si'])
265     wavenumber, reflectivity = preprocess_data(file_path)
266
267     if len(wavenumber) < 50: # 确保有足够的数据点
268         return np.nan
269
270     # 初始值与边界
271     p0 = get_initial_guess(wavenumber, reflectivity, theta_deg, mat_params)

```

```

272     bounds = get_param_bounds(mat_params)
273
274     try:
275         multi_beam_reflectivity.theta_deg = theta_deg
276         # 初始拟合（与solution一致的噪声权重）
277         data_sigma = 0.01 * reflectivity + 0.05
278         params, _ = curve_fit(
279             f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d, B,
280             offset, mat_params),
281             xdata=wavenumber,
282             ydata=reflectivity,
283             p0=p0,
284             bounds=bounds,
285             sigma=data_sigma,
286             absolute_sigma=False,
287             maxfev=MAX_FEV,
288             ftol=FIT_TOL,
289             xtol=FIT_TOL,
290             gtol=FIT_TOL,
291             method='trf'
292         )
293
294         # 多轮迭代优化（与solution一致）
295         for _ in range(2):
296             fit_curve = multi_beam_reflectivity(wavenumber, *params,
297             mat_params)
298             residuals = reflectivity - fit_curve
299             est_noise_std = np.std(residuals)
300             data_sigma = np.abs(residuals) + est_noise_std * 0.03
301             params, _ = curve_fit(
302                 f=lambda nu, d, B, offset: multi_beam_reflectivity(nu, d,
303                 B, offset, mat_params),
304                 xdata=wavenumber,
305                 ydata=reflectivity,
306                 p0=params,
307                 bounds=bounds,
308                 sigma=data_sigma,
309                 absolute_sigma=False,
310                 maxfev=MAX_FEV,
311                 method='trf'
312             )
313             return params[0] # 返回厚度值
314     except Exception as e:
315         print(f"拟合失败（入射角{theta_deg}°）: {e}")
316         return np.nan
317
318     # --- 6. 灵敏度图表（展示各角度厚度及平均值） ---
319     def plot_sensitivity_chart(results_dict):
320         """绘制入射角-厚度关系图，包含平均厚度标注"""
321         plt.figure(figsize=(14, 8))
322         file_paths = list(results_dict.keys())
323         if len(file_paths) < 2:
324             return
325
326         # 处理附件3数据
327         angles1, thicknesses1 = results_dict[file_paths[0]]
328         valid1 = ~np.isnan(thicknesses1)
329         angles1_valid = np.array(angles1)[valid1]

```

```
327     thicknesses1_valid = np.array(thicknesses1)[valid1]
328     # 计算平均厚度
329     mean1 = np.mean(thicknesses1_valid) if len(thicknesses1_valid) > 0 else
330     np.nan
331
332     # 处理附件4数据
333     angles2, thicknesses2 = results_dict[file_paths[1]]
334     valid2 = ~np.isnan(thicknesses2)
335     angles2_valid = np.array(angles2)[valid2]
336     thicknesses2_valid = np.array(thicknesses2)[valid2]
337     # 计算平均厚度
338     mean2 = np.mean(thicknesses2_valid) if len(thicknesses2_valid) > 0 else
339     np.nan
340
341     # 绘制各角度厚度
342     bars1 = plt.bar(angles1_valid - 0.2, thicknesses1_valid, width=0.4,
343                      color=COLOR_GROUP1, label=os.path.basename(file_paths[0]))
344     bars2 = plt.bar(angles2_valid + 0.2, thicknesses2_valid, width=0.4,
345                      color=COLOR_GROUP2, label=os.path.basename(file_paths[1]))
346
347     # 为附件3数据添加数值标签
348     for bar in bars1:
349         height = bar.get_height()
350         plt.text(bar.get_x() + bar.get_width()/2., height,
351                  f'{height:.4f}μm',
352                  ha='center', va='bottom', fontsize=12)
353
354     # 为附件4数据添加数值标签
355     for bar in bars2:
356         height = bar.get_height()
357         plt.text(bar.get_x() + bar.get_width()/2., height,
358                  f'{height:.4f}μm',
359                  ha='center', va='bottom', fontsize=12)
360
361     # 标注平均厚度
362     if not np.isnan(mean1):
363         plt.axhline(y=mean1, color=COLOR_GROUP1, linestyle='--',
364                      linewidth=2,
365                      label=f'{os.path.basename(file_paths[0])} 平均厚度:
366 {mean1:.4f}μm')
367     if not np.isnan(mean2):
368         plt.axhline(y=mean2, color=COLOR_GROUP2, linestyle='--',
369                      linewidth=2,
370                      label=f'{os.path.basename(file_paths[1])} 平均厚度:
371 {mean2:.4f}μm')
372
373     # 图表配置
374     plt.xlabel('入射角 (°)', fontsize=20)
375     plt.ylabel('拟合厚度 (μm)', fontsize=20)
376     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)
377     plt.grid(True, linestyle='--', alpha=0.7, axis='y')
378     plt.legend(loc='best', fontsize=14)
379     plt.xticks(range(8, 18)) # 入射角范围8-17°
380     plt.tight_layout()
381
382     # 保存图表
383     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
384     析.png')
```

```

378     plt.savefig(save_path, dpi=300, bbox_inches='tight')
379     plt.close()
380     print(f"灵敏度分析图表已保存至: {save_path}")
381
382
383 # --- 7. 主程序（按需求配置角度范围） ---
384 def main():
385     # 配置文件与对应的入射角范围（附件3:8-12°，附件4:13-17°）
386     files_config = {
387         '附件3.xlsx': [8, 9, 10, 11, 12], # 包含10°及新增角度
388         '附件4.xlsx': [13, 14, 15, 16, 17] # 包含15°及新增角度
389     }
390
391     results = {}
392     for file_path, angles in files_config.items():
393         if not os.path.exists(file_path):
394             print(f"警告: 文件 '{file_path}' 不存在, 跳过")
395             continue
396         # 计算每个入射角对应的厚度
397         thicknesses = [analyze_angle_sensitivity(file_path, angle) for
398 angle in angles]
399         results[file_path] = (angles, thicknesses)
400
401     # 生成包含平均厚度的灵敏度图表
402     plot_sensitivity_chart(results)
403
404
405 if __name__ == "__main__":
406     main()

```

Problem4_solution.py

```

1 # sic多波束干涉厚度计算代码
2 import pandas as pd
3 import numpy as np
4 from scipy.signal import find_peaks
5 import matplotlib.pyplot as plt
6 import os
7 from datetime import datetime
8 from math import sin, sqrt, pi
9
10 # ----- 1. 全局配置（依据标准与B题要求） -----
11 SIC_REFRACTIVE_INDEX = 2.55
12 # 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
13 VALID_WAVELENGTH_NM = (2500, 5000)
14 # 输出目录
15 OUTPUT_DIR = "output"
16 os.makedirs(OUTPUT_DIR, exist_ok=True)
17 os.makedirs(f"{OUTPUT_DIR}/data", exist_ok=True)
18 os.makedirs(f"{OUTPUT_DIR}/plots", exist_ok=True)
19
20 # 绘图配置
21 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
22 plt.rcParams['axes.unicode_minus'] = False
23

```

```

24
25 # ----- 2. 数据预处理（依据B题附件数据格式） -----
26
27 def preprocess_spectral_data(file_path, incident_angle):
28     """
29     处理B题附件的光谱数据（波数 $\text{cm}^{-1}$  → 波长nm，筛选有效区间）
30     :param file_path: 附件路径（附件1_processed.xlsx/附件2_processed.xlsx）
31     :param incident_angle: 入射角（B题附件1为10°，附件2为15°）
32     :return: 预处理后的数据（DataFrame）
33     """
34
35     # 读取数据（适配Excel格式）
36     try:
37         data = pd.read_excel(file_path, header=None, names=["波数( $\text{cm}^{-1}$ )",
38                               "反射率(%)"])
39         except Exception as e:
40             raise ValueError(f"读取文件失败: {str(e)}")
41
42         # 数据清洗: 去除非数值、缺失值
43         data = data.apply(pd.to_numeric, errors="coerce").dropna()
44         # 波数→波长:  $\lambda(\text{nm}) = 1e7 / \text{波数}(\text{cm}^{-1})$  (B题原理推导基础)
45         data["波长(nm)"] = 1e7 / data["波数( $\text{cm}^{-1}$ )"]
46         # 筛选有效波长区间
47         data = data[(data["波长(nm)"] >= VALID_WAVELENGTH_NM[0]) & (data["波长(nm)"] <= VALID_WAVELENGTH_NM[1])]
48         # 按波长降序排列（便于干涉级次计算）
49         data = data.sort_values("波长(nm)",
50                               ascending=False).reset_index(drop=True)
51         # 添加入射角信息
52         data["入射角(°)"] = incident_angle
53
54         # 数据量校验（B题多波束需足够点捕捉干涉条纹）
55         if len(data) < 50:
56             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
57 检测")
58
59         return data
60
61
62 # ----- 3. 多波束干涉极值点检测（B题图2多波束特征） -----
63
64 def detect_multibeam_extrema(data):
65     """
66     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）
67     :param data: 预处理后的光谱数据
68     :return: 极值点数据（DataFrame）、极大值索引、极小值索引
69     """
70
71     reflectance = data["反射率(%)"].values
72     wavelength = data["波长(nm)"].values
73
74     # 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）
75     # 极大值检测
76     peak_indices, _ = find_peaks(
77         reflectance,
78         distance=5, # 峰间距（多波束峰更密集）
79         prominence=0.05, # 峰显著性（多波束峰特征更明显）
80         height=np.mean(reflectance) + 0.1 * np.std(reflectance) # 峰高门槛
81     )
82
83     # 极小值检测（取负反射率的极大值）
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175

```

```

76     valley_indices, _ = find_peaks(
77         -reflectance,
78         distance=5,
79         prominence=0.05,
80         height=-(np.mean(reflectance) - 0.1 * np.std(reflectance)) # 谷深门槛
81     )
82
83     # 整合极值点
84     extrema_data = pd.DataFrame({
85         "波长(㎚)": np.concatenate([wavelength[peak_indices],
86         wavelength[valley_indices]]),
87         "反射率(%)": np.concatenate([reflectance[peak_indices],
88         reflectance[valley_indices]]),
89         "极值类型": ["极大值"] * len(peak_indices) + ["极小值"] *
90         len(valley_indices)
91     }).sort_values("波长(㎚)", ascending=False).reset_index(drop=True)
92
93     # 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）
94     if len(extrema_data) < 3:
95         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")
96
97
98     print(f"多波束极值检测结果：极大值{len(peak_indices)}个，极小值
99     {len(valley_indices)}个，共{len(extrema_data)}个")
100    return extrema_data, peak_indices, valley_indices
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
# ----- 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(㎚)"] # 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(㎚)"]
        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）
```

```

127
128     # 步骤2：计算每个极值点对应的厚度
129     for idx, row in extrema_data.iterrows():
130         lambda_nm = row["波长(nm)"]
131         lambda_um = lambda_nm / 1000 # 转换为μm（标准厚度单位）
132         m_i = idx * 0.5 # 当前极值点与参考点的级次差
133         k_i = k0 + m_i # 当前极值点的干涉级次
134
135         # GB/T 42905-2023公式(10.8)：附加相移影响可忽略（小数点后第三位）
136         #  $T = (k_i - 0.5) * \lambda_{\mu m} / (2 * \sqrt{n^2 - \sin^2\theta})$ 
137         denominator = 2 * sqrt(n ** 2 - sin(angle_rad) ** 2)
138         thickness_um = (k_i - 0.5) * lambda_um / denominator
139
140         # 筛选合理厚度（标准测试范围3-200μm）
141         if 3 <= thickness_um <= 200:
142             thickness_results.append({
143                 "波长(nm)": lambda_nm,
144                 "反射率(%)": row["反射率(%)"],
145                 "极值类型": row["极值类型"],
146                 "干涉级次k_i": round(k_i, 4),
147                 "厚度(μm)": round(thickness_um, 4)
148             })
149
150     # 结果整理
151     result_df = pd.DataFrame(thickness_results)
152     if result_df.empty:
153         raise ValueError("无有效厚度计算结果，需调整极值点筛选条件")
154
155     # 统计指标（符合标准精密度要求：单个实验室RSD≤1%）
156     avg_thickness = result_df["厚度(μm)"].mean()
157     std_thickness = result_df["厚度(μm)"].std()
158     thickness_rsd = (std_thickness / avg_thickness) * 100 if avg_thickness
159     != 0 else np.inf
160
161     print(f"厚度统计：平均厚度={avg_thickness:.4f}μm, 标准差=
162 {std_thickness:.4f}μm, RSD={thickness_rsd:.2f}%")
163
164     return result_df, avg_thickness, thickness_rsd
165
166 # ----- 5. 结果可视化（展示多波束光谱与极值点） -----
167
168 def plot_multibeam_spectrum(data, extrema_data, peak_indices,
169                             valley_indices, incident_angle, save_path):
170     """
171     绘制多波束干涉光谱图（标注极值点，符合B题图2特征）
172     :param data: 预处理后的数据
173     :param extrema_data: 极值点数据
174     :param peak_indices: 极大值索引
175     :param valley_indices: 极小值索引
176     :param incident_angle: 入射角（°）
177     :param save_path: 图像保存路径
178     """
179
180     fig, ax = plt.subplots(figsize=(12, 6))
181
182     # 绘制原始光谱
183     ax.plot(data["波长(nm)"], data["反射率(%)"], color="#619cff", alpha=0.8,
184             linewidth=1.2, label="多波束反射光谱")
185
186     # 标注极大值点
187     for index, value in peak_indices.items():
188         ax.vlines(index, 0, value, colors='red', linestyles='dashed')
189
190     # 标注极小值点
191     for index, value in valley_indices.items():
192         ax.vlines(index, 0, value, colors='blue', linestyles='dashed')
193
194     # 标注极值点
195     for index, value in extrema_data["反射率(%)"].items():
196         ax.vlines(index, 0, value, colors='black', linestyles='solid')
197
198     # 入射角标注
199     ax.text(180, 95, f'入射角: {incident_angle}°', transform=ax.transData)
200
201     # 图例
202     ax.legend()
203
204     # 保存图像
205     plt.savefig(save_path)
206
207     # 显示光谱图
208     plt.show()
209
210     # 清理
211     plt.close()
212
213     return fig

```

```

180     ax.scatter(
181         data.iloc[peak_indices]["波长(nm)"],
182         data.iloc[peak_indices]["反射率(%)"],
183         color="#00BA38", s=60, marker="^", label="多波束极大值点", zorder=5
184     )
185     # 标注极小值点
186     ax.scatter(
187         data.iloc[valley_indices]["波长(nm)"],
188         data.iloc[valley_indices]["反射率(%)"],
189         color="#F8766D", s=60, marker="v", label="多波束极小值点", zorder=5
190     )
191
192     # 图表配置
193     ax.set_xlabel("波长 (nm)", fontsize=12)
194     ax.set_ylabel("反射率 (%)", fontsize=12)
195     ax.set_title(f"sic多波束干涉光谱 (入射角{incident_angle}°)", fontsize=14,
196     fontweight="bold")
196     ax.grid(True, linestyle="--", alpha=0.5, color="gray")
197     ax.legend(fontsize=10)
198     ax.set_xlim(VALID_WAVELENGTH_NM[0], VALID_WAVELENGTH_NM[1])
199
200     # 保存图像
201     plt.tight_layout()
202     plt.savefig(save_path, dpi=300, bbox_inches="tight")
203     plt.close()
204     print(f"光谱图已保存至: {save_path}")
205
206
207 # ----- 6. 主函数 (整合流程: 数据→极值→厚度→输出) -----
208
209 def main(file_path, incident_angle, file_label):
210     """
211         主流程: 处理单个附件的多波束厚度计算
212         :param file_path: 附件路径
213         :param incident_angle: 入射角 (°)
214         :param file_label: 附件标签 (如"附件1_10°")
215         :return: 最终平均厚度 (μm)
216     """
217     print(f"\n{'=' * 60}\n开始处理{file_label}...\n{'=' * 60}")
218
219     # 1. 数据预处理
220     try:
221         data = preprocess_spectral_data(file_path, incident_angle)
222         print(
223             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"
224         )
225     except Exception as e:
226         print(f"数据预处理失败: {str(e)}")
227         return None
228
229     # 2. 多波束极值检测
230     try:
231         extrema_data, peak_indices, valley_indices =
232         detect_multibeam_extrema(data)
233     except Exception as e:
234         print(f"极值检测失败: {str(e)}")
235         return None

```

```
234     # 3. 厚度计算
235     try:
236         thickness_df, avg_thickness, thickness_rsd =
237             calculate_multibeam_thickness(extrema_data, incident_angle)
238     except Exception as e:
239         print(f"厚度计算失败: {str(e)}")
240         return None
241
242     # 4. 结果可视化
243     plot_path = f"{OUTPUT_DIR}/plots/{file_label}_多波束光谱图.png"
244     plot_multibeam_spectrum(data, extrema_data, peak_indices,
245     valley_indices, incident_angle, plot_path)
246
247     # 5. 结果保存 (Excel)
248     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
249     excel_path = f"{OUTPUT_DIR}/data/{file_label}_多波束厚度结果
250     _{timestamp}.xlsx"
251     with pd.ExcelWriter(excel_path, engine="openpyxl") as writer:
252         # 原始预处理数据
253         data.to_excel(writer, sheet_name="预处理光谱数据", index=False)
254         # 极值点数据
255         extrema_data.to_excel(writer, sheet_name="多波束极值点", index=False)
256         # 厚度计算结果
257         thickness_df.to_excel(writer, sheet_name="厚度计算结果", index=False)
258         # 统计汇总
259         stats_df = pd.DataFrame({
260             "统计项": ["平均厚度(μm)", "厚度标准差(μm)", "厚度RSD(%)", "入射角
261             (°)", "有效极值点数"],
262             "数值": [avg_thickness, thickness_df["厚度(μm)"].std(),
263             thickness_rsd, incident_angle, len(extrema_data)]
264         })
265         stats_df.to_excel(writer, sheet_name="结果统计", index=False)
266
267     print(f"结果数据已保存至: {excel_path}")
268     print(f"{file_label} 处理完成, 最终平均厚度: {avg_thickness:.4f}μm\n")
269     return avg_thickness
270
271
272
273     # ----- 7. 执行入口 (适配B题附件1、附件2) -----
274
275 if __name__ == "__main__":
276     # 配置B题附件路径与入射角 (需根据实际文件位置调整)
277     附件1路径 = "附件1_processed.xlsx" # B题附件1: 入射角10°
278     附件2路径 = "附件2_processed.xlsx" # B题附件2: 入射角15°
279
280     # 执行附件1处理
281     附件1平均厚度 = main(
282         file_path=附件1路径,
283         incident_angle=10,
284         file_label="附件1_入射角10°"
285     )
286
287     # 执行附件2处理
288     附件2平均厚度 = main(
289         file_path=附件2路径,
290         incident_angle=15,
291         file_label="附件2_入射角15°"
292     )
```

```

286
287     # 最终结果汇总
288     print(f"\n{'=' * 80}")
289     print("B题SIC多波束干涉厚度计算最终结果汇总")
290     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")
291     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")
292     print(f"{'=' * 80}")

```

Problem_灵敏度分析

```

1 # 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）
2 import pandas as pd
3 import numpy as np
4 from scipy.signal import find_peaks
5 import matplotlib.pyplot as plt
6 import os
7 from math import sin, sqrt, pi
8
9 # ----- 1. 核心参数（源自B题与GB/T 42905-2023标准） -----
10 SIC_REFRACTIVE_INDEX = 2.55 # 标准明确sic折射率
11 VALID_WAVELENGTH_NM = (2500, 5000) # 有效波长区间（对应标准3-200μm测试范围）
12 SENSITIVITY_ANGLE_GROUPS = {
13     "附件1_processed.xlsx": [8, 9, 10, 11, 12], # 附件1基础10°+新增角度
14     "附件2_processed.xlsx": [13, 14, 15, 16, 17] # 附件2基础15°+新增角度
15 }
16 OUTPUT_DIR = "output/灵敏度分析"
17 os.makedirs(OUTPUT_DIR, exist_ok=True) # 自动创建灵敏度分析输出文件夹
18
19 # 绘图基础配置（确保图表清晰）
20 plt.rcParams["font.family"] = ["SimHei", "DejaVu Sans"]
21 plt.rcParams['axes.unicode_minus'] = False
22 plt.rcParams['font.size'] = 11
23 plt.rcParams['figure.dpi'] = 150
24
25
26 # ----- 2. 数据预处理（适配B题附件格式） -----
27 def preprocess_data(file_path, incident_angle):
28     """简化版数据处理：波数转波长+有效区间筛选+数据清洗"""
29     # 读取附件数据（B题附件为Excel格式，无表头）
30     data = pd.read_excel(file_path, header=None, names=["波数(cm⁻¹)", "反射率(%)"])
31     # 清洗非数值/缺失值
32     data = data.apply(pd.to_numeric, errors="coerce").dropna()
33     # 波数-波长（B题原理：λ( nm ) = 1e7 / 波数( cm⁻¹ )）
34     data["波长(nm)"] = 1e7 / data["波数(cm⁻¹)"]
35     # 筛选有效波长区间（排除干扰）
36     data = data[(data["波长(nm)"] >= VALID_WAVELENGTH_NM[0]) & (data["波长(nm)"] <= VALID_WAVELENGTH_NM[1])]
37     # 按波长降序（便于干涉级次计算）
38     data = data.sort_values("波长(nm)", ascending=False).reset_index(drop=True)
39     data["入射角(°)"] = incident_angle
40     return data

```

```

41
42
43 # ----- 3. 多波束极值检测 (B题图2多波束特征适配) -----
44 -----
45 def detect_extrema(data):
46     """简化版极值检测: 仅保留多波束峰谷核心逻辑"""
47     reflectance = data["反射率(%)"].values
48     wavelength = data["波长(nm)"].values
49
50     # 多波束峰检测 (峰更尖锐, 降低显著性阈值)
51     peaks, _ = find_peaks(reflectance, distance=5, prominence=0.05,
52                           height=np.mean(reflectance) + 0.1 *
53                           np.std(reflectance))
54
55     # 多波束谷检测 (取负反射率的峰)
56     valleys, _ = find_peaks(-reflectance, distance=5, prominence=0.05,
57                             height=-np.mean(reflectance) - 0.1 *
58                             np.std(reflectance)))
59
60     # 整合极值点 (按波长排序)
61     extrema = pd.DataFrame({
62         "波长(nm)": np.concatenate([wavelength[peaks],
63                                     wavelength[valleys]]),
64         "反射率(%)": np.concatenate([reflectance[peaks],
65                                     reflectance[valleys]]),
66         "类型": ["峰"] * len(peaks) + ["谷"] * len(valleys)
67     }).sort_values("波长(nm)", ascending=False).reset_index(drop=True)
68
69     return extrema if len(extrema) >= 3 else None # 多波束需至少3个极值点
70
71
72 # ----- 4. 厚度计算 (严格遵循GB/T 42905-2023公式) -----
73 -----
74 def calc_thickness(extrema, incident_angle):
75     """简化版厚度计算: 基于标准10.8公式 (忽略微小相移影响)"""
76     n = SIC_REFRACTIVE_INDEX
77     angle_rad = incident_angle * pi / 180 # 入射角转弧度
78     lambda_ref = extrema.iloc[0]["波长(nm)"] # 参考波长 (最长波长)
79     k0_est = []
80
81     # 估算干涉级次基数k0 (多波束级次差为0.5)
82     for i in range(1, len(extrema)):
83         lambda_i = extrema.iloc[i]["波长(nm)"]
84         delta_lambda = lambda_ref - lambda_i
85         if delta_lambda < 10:
86             continue
87         k0 = (i * 0.5 * lambda_i) / delta_lambda # 级次差m=i*0.5
88         k0_est.append(k0)
89
90     k0 = np.median(k0_est) # 中位数抗异常值 (多波束稳定性更优)
91
92     # 计算每个极值点厚度并取平均
93     thickness_list = []
94     for idx, row in extrema.iterrows():
95         lambda_um = row["波长(nm)"] / 1000 # 转μm (标准厚度单位)
96         k_i = k0 + idx * 0.5 # 当前级次
97         # 标准公式: T=(k_i-0.5)*λ/(2*sqrt(n²-sin²θ))
98         denom = 2 * sqrt(n ** 2 - sin(angle_rad) ** 2)
99         thick = (k_i - 0.5) * lambda_um / denom
100        if 3 <= thick <= 200: # 标准测试范围 (3-200μm)
101            thickness_list.append(thick)
102
103    return np.mean(thickness_list)

```

```

93         thickness_list.append(thick)
94
95     return np.mean(thickness_list) if thickness_list else None # 返回平均厚度
96
97
98 # ----- 5. 灵敏度分析核心逻辑（入射角→厚度响应） -----
99
100 def sensitivity_analysis(file_path, angle_group):
101     """针对单个附件的灵敏度分析：计算不同入射角的平均厚度"""
102     thickness_results = []
103     for angle in angle_group:
104         # 流程：数据处理→极值检测→厚度计算
105         data = preprocess_data(file_path, angle)
106         extrema = detect_extrema(data)
107         if extrema is None:
108             continue
109         avg_thick = calc_thickness(extrema, angle)
110         if avg_thick is not None:
111             thickness_results.append({"入射角(°)": angle, "平均厚度(μm)": round(avg_thick, 4)})
112
113
114 # ----- 6. 结果可视化（汇总所有入射角-厚度数据） -----
115
116 def plot_sensitivity_summary(results_dict):
117     """绘制灵敏度分析汇总图：所有附件的入射角-厚度关系"""
118     fig, ax = plt.subplots(figsize=(10, 6))
119
120     # 区分附件1和附件2数据（用不同样式）
121     colors = ["#2E86AB", "#A23B72"]
122     markers = ["o", "s"]
123     labels = ["附件1_processed (基础10°)", "附件2_processed (基础15°)"]
124
125     for (file, df), color, marker, label in zip(results_dict.items(), colors, markers, labels):
126         if not df.empty:
127             ax.plot(
128                 df["入射角(°)", df["平均厚度(μm)"],
129                 color=color, marker=marker, markersize=8, linewidth=2,
130                 label=label
131             )
132             # 标注数据点数值
133             for _, row in df.iterrows():
134                 ax.text(
135                     row["入射角(°)", row["平均厚度(μm)"],
136                     f'{row["平均厚度(μm)"]:.4f}',
137                     ha="center", va="bottom", fontsize=9, color=color
138                 )
139             # 图表配置（突出灵敏度分析主题）
140             ax.set_xlabel("入射角 (°)", fontsize=12, fontweight="bold")
141             ax.set_ylabel("SIC外延层平均厚度 (μm)", fontsize=12, fontweight="bold")
142             ax.set_title("SIC多波束干涉厚度-入射角灵敏度分析", fontsize=14,
143                         fontweight="bold", pad=20)
144             ax.grid(True, linestyle="--", alpha=0.5, color="gray")
145             ax.legend(loc="best", framealpha=0.9)

```

```
144     ax.set_xlim(min(SENSITIVITY_ANGLE_GROUPS["附件1_processed.xlsx"]) - 0.5,
145                 max(SENSITIVITY_ANGLE_GROUPS["附件2_processed.xlsx"]) + 0.5)
146
147     # 保存图表到灵敏度分析文件夹
148     plot_path = os.path.join(OUTPUT_DIR, "sic厚度-入射角灵敏度分析图.png")
149     plt.tight_layout()
150     plt.savefig(plot_path, dpi=300, bbox_inches="tight")
151     plt.close()
152     print(f"灵敏度分析图已保存至: {plot_path}")
153
154
155 # ----- 7. 执行入口（一键运行灵敏度分析） -----
156 -----
156 if __name__ == "__main__":
157     # 存储所有附件的灵敏度分析结果
158     all_sensitivity_results = []
159
160     # 遍历两个附件，执行灵敏度分析
161     for file_name, angle_group in SENSITIVITY_ANGLE_GROUPS.items():
162         print(f"正在处理{file_name}的灵敏度分析（入射角: {angle_group}°）...")
163         result_df = sensitivity_analysis(file_name, angle_group)
164         all_sensitivity_results[file_name] = result_df
165         print(f"{file_name}分析完成，有效数据点: {len(result_df)}个\n")
166
167     # 绘制并保存汇总图
168     plot_sensitivity_summary(all_sensitivity_results)
169     print("灵敏度分析全部完成！结果图表位于: output/灵敏度分析")
```