碳化硅外延层厚度确定的设计及效果分析

摘 要

红外干涉法是一种外延层厚度测量的无损伤测量方法。通过运用红外干涉原理和利用红外光谱仪，我们能够无损、简便、快速地测量碳化硅外延层的厚度。本文基于对红外干涉法测量外延层过程中波数与反射率的分析，建立了相应的数学模型，并设计了合理的厚度测量方案，对提高外延层厚度测量的精确性具有一定作用。

针对问题一：本文针对红外干涉法测量碳化硅外延层厚度问题，建立了基于空气—外延层—内衬体系两束反射光干涉的数学模型。通过**斯涅尔定律**计算折射角 ϕ，结合几何光程差与等效真空光程差，将外延层厚度 d 与相邻干涉极值的波数间隔 $\Delta\tilde{v}$ 联系起来，推导出解析公式

$$d = \frac{1}{2n_1 \cos\phi \, \Delta\tilde{v}}.$$

在算法上，采用逐对极值法，计算每一组相邻波数所对应的厚度值。

针对问题二：本文针对不同入射角下的红外干涉光谱，建立了基于干涉条纹分析的数学模型，用以精确求取碳化硅外延层厚度。首先，我们对实验反射率数据进行多项预处理，包括中值滤波、Savitzky-Golay 平滑和基线校正，以降低噪声干扰并保留信号趋势。随后，采用 **AMPD 算法**检测光谱中的波峰位置，结合波峰间波数差及厚度计算公式，得到每组相邻波峰对应的厚度值。通过**箱线图法**剔除异常值后，计算剩余厚度的平均值，并利用**非线性曲线拟合**和**交叉验证**评估模型的可靠性。最后，根据附件一和附件二数据计算出来的厚度结果分别为 $\mathbf{8.0074}\,\mu$m 和 $\mathbf{8.0396}\,\mu$m，最终算得的结果为 $\mathbf{8.0233}\,\mu$m 。同时，拟合 R^2 接近 $\mathbf{0.98}$，RMSE 与 MAE 均较小，表明模型对实验数据解释力强，厚度估计稳定可靠。

针对问题三：在假定多光束干涉的情况下，反射光的振幅分布符合几何级数，因此我们可以建立反射率与波数之间的等式关系，得到数学模型。与双光束干涉的数学模型进行对比，从数学模型的关系式中我们发现，多光束干涉的必要条件为外延层厚度与波长具有可比性。以及外延层–衬底界面具有足够的反射率。在此基础上，我们推导出厚度与干涉条纹间隔的定量关系，并利用非线性最小二乘方法对实验数据进行拟合。计算结果显示，当入射角为 10° 时厚度为 $\mathbf{3.92}\,\mu$m，15° 时为 $\mathbf{3.77}\,\mu$m，平均厚度约 $\mathbf{3.845}\,\mu$m。

对比附件一、二的实验数据发现，多光束干涉模型拟合效果较差，说明碳化硅晶圆片中主要表现为双光束干涉，其厚度计算结果仍为 $\mathbf{8.0233}\,\mu$m。

关键词： AMPD 算法　斯涅尔定律　交叉验证　多光束干涉　非线性拟合算法

1 问题重述

1.1 问题背景

碳化硅是一种新兴的第三代半导体材料，因其高击穿电场强度、高热导率和宽禁带特性，在高功率电子器件、电动汽车以及射频通信等领域展现出广泛应用前景。在碳化硅半导体器件中，外延层的厚度直接影响器件的电阻、电容及击穿电压控制，从而决定器件的性能和可靠性。因此，对外延层厚度进行精确、无损的测量具有重要意义。

现有多种测量方法中，红外干涉法凭借其简便、快速和无损的特点，成为测量碳化硅外延层厚度的有效手段。该方法基于空气—外延层—衬底体系中反射光的干涉条纹，通过分析光的干涉极值可反演外延层厚度。然而，实验数据中仍存在多种影响因素，包括光谱分辨率限制、噪声干扰、入射角变化以及折射率随波长的色散效应等，这些因素在不同测量条件下会导致厚度反演误差。

因此，在建立数学模型和求解方法时，需要首先在忽略色散条件下建立基础干涉光程与厚度的关系（问题一），随后考虑折射率色散对测量结果的影响并进行校正（问题二），并在此基础上进一步考虑多光束干涉对碳化硅外延层厚度计算的精度的影响，推导测量过程中多光束干涉产生的必要条件，评估干涉对测量精度的影响，进而设计稳健的实验数据处理和优化算法，以提高测量精度和可靠性（问题三）。

1.2 需要解决的问题

问题一：在只考虑外延层和衬底界面界面只有一次反射、透射，忽略多光束干涉的情况下，建立与红外光谱相关的碳化硅外延层折射率的模型，并建立起根据红外光谱的波长、外延层的折射率和红外光的入射角等参数确定外延层的厚度的数学模型。

问题二：根据问题一的数学模型，确定外延层厚度的算法，使用附件 1 和附件 2 中提供的实测数据，计算最终厚度，并分析可靠性。

问题三：考虑外延层界面和衬底界面产生多次反射和透射所产生的多光束干涉，推导产生多光束干涉的必要条件，以及多光束干涉对外延层厚度计算精度可能产生的影响。根据多光束干涉的必要条件，分析附件 3 和附件 4 提供的硅晶圆片的测试结果是否出现多光束干涉，给出确定硅外延层厚度计算的数学模型和算法，以及相应的计算结果。在考虑多光束干涉的情况下，重新计算碳化硅晶圆片的厚度。

2 问题分析

2.1 问题一分析

本题要求通过红外干涉法测量碳化硅外延层厚度。根据题意，空气—外延层—内衬三层体系的红外光谱可视为两束光干涉模型：一束光在空气—外延层界面反射，另一束光透射进入外延层并在外延层—内衬界面反射后返回，两束光叠加形成干涉条纹。

在物理分析上，入射角 θ 与折射角 ϕ 满足斯涅尔定律，光在外延层中的往返几何光程差为 $2d\cos\phi$，折算为真空等效光程差 $\Delta L = 2n_1 d\cos\phi$。干涉极值条件（波峰或波谷）将相位差公式转化为 ΔL 与波数间隔 $\Delta\tilde{v}$ 的关系，消去界面相位项 δ，得到厚度解析公式：

$$d = \frac{1}{2n_1\cos\phi\,\Delta\tilde{v}}.$$

算法上，采用逐对极值法，通过测量任意相邻波峰或波谷的波数差直接反演厚度 d。该方法公式简洁、计算直接，无需拟合或额外参数，适合忽略色散条件下快速、可靠地求解外延层厚度。

2.2 问题二分析

本题要求根据问题 1 的数学模型，我们需要先对附件 1 和附件 2 提供的碳化硅晶圆片的光谱实测数据进行数据清洗，再通过 AMPD 波峰检测算法，求出所有波峰对应的波数值。接着用 Python 实现上述数学模型的计算，对每一组相邻波峰对应的波数值都计算出一组相应厚度值。再用箱线法剔除异常厚度值，最终对剩下的厚度值进行平均处理，得到最终厚度值。

2.3 问题三分析

本题需要考虑外延层界面和衬底界面产生的多次反射和透射。可以通过外延层-衬底界面的振幅反射系数和空气-外延层界面的振幅透射系数表示经过多次反射和折射的光强。发现得到的光强属于几何级数，以此建立干涉光谱的反射率关于波数的方程。经转化后可以得到厚度的表达式。结合所给数据，基于条纹间距计算厚度或进行全谱拟合。

3 模型假设

(1) 空气的折射率等于真空的折射率等于 1；

(2) 入射角足够小，可以忽略菲涅尔效应在 s 方向和 p 方向上的差距；

(3) 单色光近似：单次入射红外光近似为单色或窄带光，否则干涉条纹不清晰；

(4) 相干条件：反射光 1 和反射光 2 保持相干；

(5) 碳化硅种类为 4H-Si

(6) 界面光滑平整：空气-外延层、外延层-内衬界面是理想平面，没有明显散射。

4 符号说明

符号	含义
d	碳化硅外延层厚度
n_0	空气折射率
n_1	碳化硅外延层折射率
n_2	硅晶圆片外延层折射率
λ	红外光在真空中的波长
θ	入射角（空气—外延层界面）
ϕ	折射角（空气—外延层界面）
ΔL	等效光程差（对应在真空中的光程差）
$\Delta\varphi$	两束反射光的相位差
δ	合计半波损失
R	接收到的光强
E_1	反射光 1 对应的光强
E_2	反射光 2 对应的光强
$\tilde{v}$	波数
$\Delta\tilde{v}$	相邻干涉条纹的波数间隔
$A(\tilde{v}), B(\tilde{v})$	以波数 $\tilde{v}$ 为自变量的多项式
Z	整数集
k	干涉极值的整数序号（$k \in \mathbb{Z}$）
r_1	空气-外延层界面振幅反射系数
r_2	外延层-衬底界面振幅反射系数
t_1	空气-外延层界面（空气入射）振幅透射系数
t_2	空气-外延层界面（外延层入射）振幅透射系数

5 模型的建立与求解

5.1 问题一模型的建立与求解

物理分析

考虑空气—外延层—内衬的三层体系。入射红外光在空气—外延层界面部分反射形成反射光 1，其余透射进入外延层并在外延层—内衬界面反射后返回，形成反射光 2。两束光在探测器处叠加，产生干涉条纹。

入射角 θ 与外延层内折射角 ϕ 满足斯涅尔定律：

$$n_0 \sin\theta = n_1 \sin\phi. \tag{1}$$

反射光 2 相对于反射光 1 多经过一次外延层往返，其几何光程差为

$$\Delta L \ = 2d\cos\phi.$$

折算为真空等效光程差为

$$\Delta L = n_1(\tilde{v})\Delta L \ = 2n_1(\tilde{v})d\cos\phi. \tag{2}$$

数学建模

记真空波长为 λ，波数为 $\tilde{v} = 1/\lambda$。两束光的相位差为

$$\Delta\varphi = 2\pi\tilde{v}\,\Delta L + \delta, \tag{3}$$

其中 δ 为半波损失产生的相位差（常数）。

干涉光强为

$$R = E_1^2 + E_2^2 + 2E_1E_2\cos(\Delta\varphi), \tag{4}$$

亦可写为

$$R(\tilde{v}) = A(\tilde{v}) + B(\tilde{v})\cos\big(2\pi\tilde{v}\,\Delta L + \delta\big), \tag{5}$$

其中 $A(\tilde{v}), B(\tilde{v})$ 为缓慢变化函数。

干涉极值满足：

$$\text{波峰（极大）}: \quad \Delta\varphi = 2k\pi, \quad k \in \mathbb{Z}, \tag{6}$$

$$\text{波谷（极小）}: \quad \Delta\varphi = (2k+1)\pi, \quad k \in \mathbb{Z}. \tag{7}$$

设相邻两个波峰对应波数为 $\tilde{v}(k)$ 与 $\tilde{v}(k+1)$，则有

$$2\pi\big(\tilde{v}(k+1) - \tilde{v}(k)\big)\Delta L = 2\pi,$$

即

$$\Delta L = \frac{1}{\tilde{v}(k+1) - \tilde{v}(k)} = \frac{1}{\Delta\tilde{v}}. \tag{8}$$

为了确定碳化硅外延层的折射率，我们参考 Wang 等人在《4H-SiC: A new nonlinear material for midinfrared lasers》[1] 中提供的数据，折射率 n_1 随波长变化，可通过以下公式表示：

$$n_1^2 - 1 = \frac{0.20075\,\lambda^2}{\lambda^2 + 12.07224} + \frac{5.54861\,\lambda^2}{\lambda^2 - 0.02641} + \frac{35.65066\,\lambda^2}{\lambda^2 - 1268.24708}$$

其中 λ 为红外光在真空中的波长。

结合 (2)，假定小范围的波数变化不会造成折射率改变，得到碳化硅外延层厚度解析公式：

$$d = \frac{1}{2n_1\cos\phi\,\Delta\tilde{v}}. \tag{9}$$

模型结果

模型表明，通过测量红外干涉谱中任意相邻极值的波数间隔 $\Delta\tilde{v}$，结合已知折射率 n_1 与折射角 ϕ，即可直接反演碳化硅外延层厚度 d。

5.2 问题二模型的建立与求解

5.2.1 数据预处理

为了减少噪声干扰并保证反射率数据的可靠性，我们在分析前对原始数据进行了三步预处理。首先，使用窗口大小为 5 的中值滤波器（`median_filter`）去除尖峰噪声，该方法能够消除局部异常点而不影响整体趋势。随后，在滤波后的数据上应用 Savitzky–Golay 平滑处理，窗口长度设为 11，多项式阶数为 3，以降低高频波动并保持光谱的局

部结构特征。最后，对平滑后的光谱进行基线校正，将数据序列减去其最小值，使最低点对应于零，从而消除因仪器漂移等因素导致的基线偏移。[1]

5.2.2 波峰检测

为了从光谱数据中提取干涉条纹信息，我们需要检测光谱中的波峰。题目给出了两个附件，分别是入射角为 10° 和 15° 时针对同一块碳化硅晶圆片的测试结果，其中第 1 列为波数（单位：cm^{-1}），第 2 列为干涉光谱的反射率（单位：%）。

反射光在外延层与衬底界面之间的干涉作用，形成了明暗相间的干涉条纹。在反射率光谱图中，横坐标为波数（单位：cm^{-1}），纵坐标为反射率（单位：%）。我们可以看到，在特定的波数位置，光谱中呈现出明显的反射率波峰，这些波峰反映了干涉增强的现象。因此我们需要提取反射率波峰。

为了准确地提取波峰位置，我们使用了 AMPD 算法（Amplitude Modulation Peak Detection）。该算法通过扫描光谱数据，检测局部最大值，从而找出波峰位置。

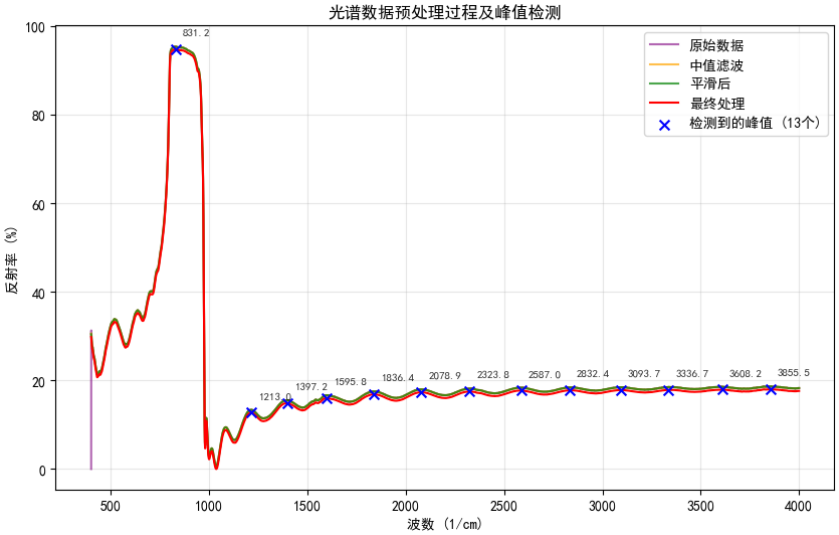

图 1: 入射角 10° 时的峰值检测图

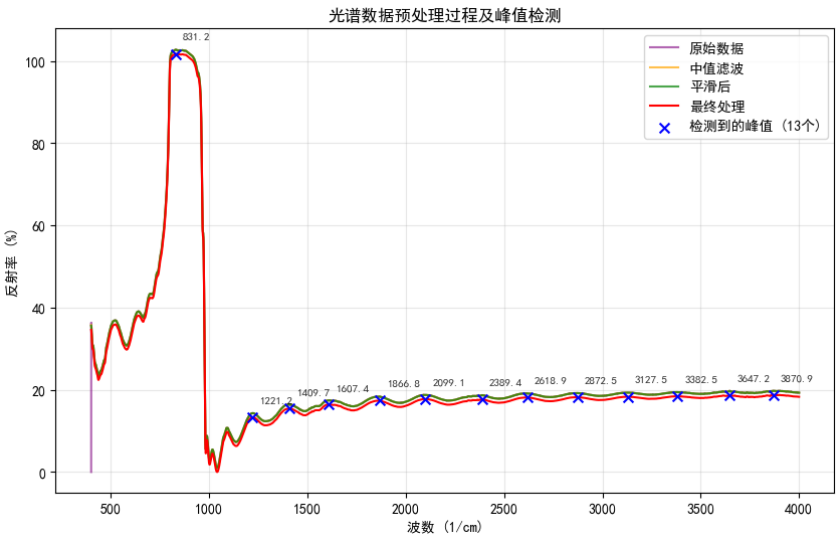

图 2: 入射角 15° 时的峰值检测图

[1]AI 辅助完成

5.2.3 对每一组相邻波数进行外延层厚度计算

在检测出所有波峰位置后，对所有波峰进行排查。发现 test1.xlsx 和 test2.xlsx 文件算出来的第一个波峰都远高于其他波峰，故排除。我们记录剩下波峰对应的波数值。根据问题一建立的厚度计算公式，结合波峰间的波数差，计算出外延层的厚度。对于每一组相邻的波峰的波数值，都可以计算出一组相应的外延层厚度。如下图所示：

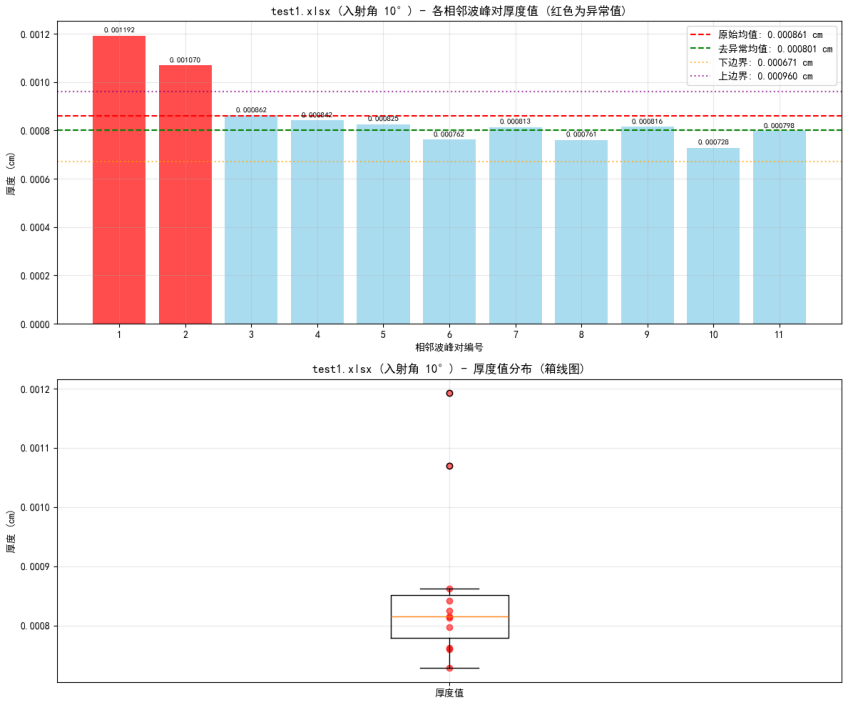

图 3: 入射角 10° 时的厚度值

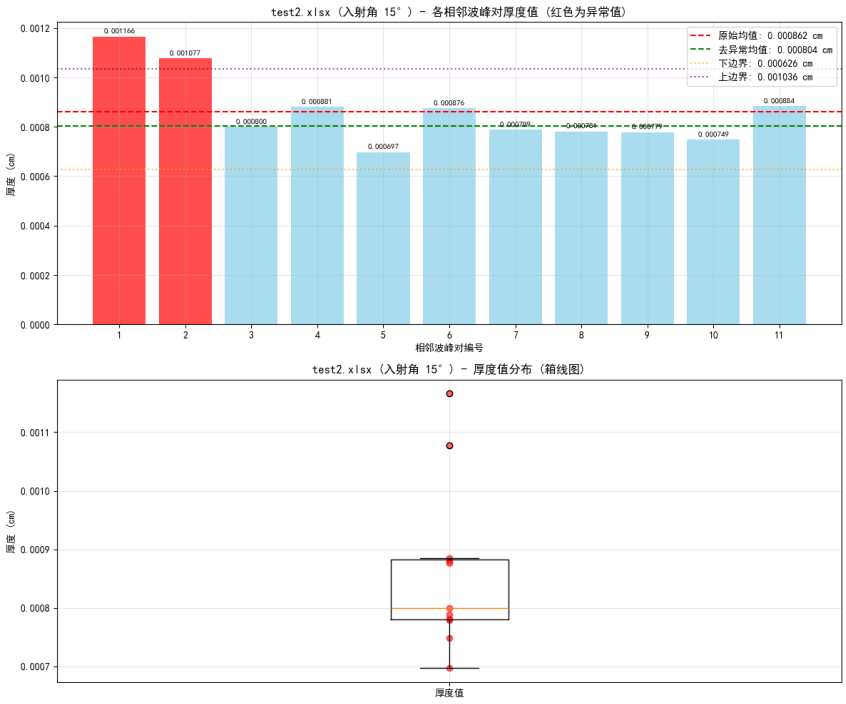

图 4: 入射角 15° 时的厚度值

5.2.4 箱线图法去除异常值

在计算出所有相邻波数所对应的厚度值后，由于实验误差或数据质量问题，我们发现部分数据远高于整体平均值。故需要对异常值进行排查。因此，我们采用了箱线图法来剔除这些异常值。箱线图法通过计算数据的四分位数和四分位差, 确定数据的上下边界值，并剔除位于边界之外的异常值。

5.2.5 求取厚度平均值

在剔除异常值之后，我们对每个文件的剩下的厚度值进行平均处理。对于附件一得到平均厚度值为：8.0074 µm。对于附件二得到平均厚度值为：8.0396 µm。

5.2.6 结果可靠性分析

检验的公式和原理：

$$R(\tilde{\nu}) \;=\; A(\tilde{\nu}) \;+\; B(\tilde{\nu})\,\cos\Big(4\pi\, n_1(\tilde{\nu})\, d\,\cos\big(\theta_i(\tilde{\nu})\big)\,\tilde{\nu} + \delta\Big). \tag{10}$$

这个方程包含波数和干涉率，故我们使用附件一和附件二的数据以及算出来的厚度 d 可以将该函数拟合出来。为了检验 d 的可靠性：对于任意一个附件，我们都先将附件中的数据随机分成两组，其中 80% 的数据作为拟合数据，20% 的数据作为检验数据，以检验拟合的可靠性。

在模型拟合的过程中，我们运用 `curve_fit`，这是 SciPy 库中的一个函数。该函数用于最小化非线性模型和观测数据之间的误差，并找到最佳的模型参数。其核心原理是最小二乘法，即最小化模型预测值与实际观测值之间的差异。

具体而言，对于模型中的函数 $A(\tilde{\nu})$、$B(\tilde{\nu})$ 和 δ，我们都对其作关于波数的二次多项式近似。因此，整个模型一共包含 9 个待拟合的参数。我们对该函数使用 SciPy 库中的 `curve_fit` 函数进行拟合。该函数运用最小二乘法的原理，最小化模型预测值与实际观测值之间的差距，从而实现函数的拟合。

拟合完成之后，我们用包含 20% 数据的测试集进行评估。通过比较测试集的预测值与真实值，我们计算了 R^2 分数、均方根误差（RMSE）和平均绝对误差（MAE），从而对模型进行评估，拟合结果如下：

表 1: 拟合结果评估指标

附件	厚度 d (cm)	入射角 θ	R^2 分数	RMSE	MAE
附件一	0.00080074	10°	0.983456	0.106848	0.087104
附件二	0.00080396	15°	0.983023	0.115724	0.085607

从结果可以看出，两组实验数据的拟合 R^2 均接近于 0.98，且 RMSE 与 MAE 数值较小。这说明模型对观测数据的解释程度较高，拟合效果理想。由此可见，根据我们建立的数学模型计算出的厚度具有较强的可靠性。

5.3 问题三模型的建立与求解

5.3.1 多光束干涉的必要条件

在本题中，我们需要分析外延层光谱中可能出现的多光束干涉现象，并基于干涉规律建立厚度计算模型。为此，首先需要明确多光束干涉产生的必要条件，这将直接决定

实验数据中是否能够观察到干涉条纹以及条纹是否足够清晰，从而影响厚度计算的可靠性。

多光束干涉的产生依赖于以下两个条件：

1. **外延层厚度与波长具有可比性。**当外延层厚度 d 与红外光在真空中的波长 λ 同量级时，两束相邻反射光的等效光程差

$$\Delta L = 2n_1 d\cos\phi$$

所对应的相位差为

$$\Delta\varphi = \frac{2\pi}{\lambda}\Delta L + \delta,$$

其数值足够大时，能够形成明显的干涉条纹。若 d 过小，则 $\Delta\varphi$ 太小，干涉效应不明显；若 d 过大，则相邻条纹的波数间隔 $\Delta\tilde{v}$ 过小，容易受到实验噪声和测量误差的干扰。

2. **外延层–衬底界面反射率足够大。**外延层–衬底界面的振幅反射系数 r_2 决定了多次反射光的强度。如果 r_2 较大，则多次反射的光强不可忽略，条纹对比度较高；反之，若反射率过低，则高阶反射光强度衰减过快，导致干涉条纹模糊甚至消失。

5.3.2 多光束干涉对厚度计算精度的影响

在满足必要条件的情况下，多光束干涉能够在实验光谱中形成明显的条纹，从而为外延层厚度 d 的测量提供依据。需要注意到，多光束干涉在提升厚度计算灵敏度的同时，也可能引入额外的不确定性，需要从以下几个方面进行分析。

首先，高反射率条件下的强多光束干涉可显著提高测量灵敏度。由于相位差

$$\Delta\varphi = \frac{2\pi}{\lambda}\cdot 2n_1 d\cos\phi + \delta$$

与厚度 d 呈线性关系，当界面反射率较大时，反射谱对 d 的微小变化极为敏感，即小的厚度变化即可引起明显的条纹偏移。这一特性有利于提高厚度测量的分辨率。

然而，高灵敏度的同时也意味着系统稳定性要求更高。实验中存在温度波动、入射角微小扰动以及光谱标定误差等因素，这些都会影响折射率 n_1、入射角 ϕ 或波长 λ 的精确取值。由于干涉条纹对这些参数变化高度敏感，其不确定性将被放大，表现为厚度计算中的系统性偏差。

此外，有限的光源相干长度和入射角分布还会进一步降低条纹对比度，使得条纹峰值位置的判定存在随机误差，从而增加了厚度反演的不确定性。所以综合来看，多光束干涉一方面提升了测量分辨率，另一方面也可能放大系统误差和实验噪声。

因此，在利用多光束干涉进行厚度测量时，需要在灵敏度和稳定性之间进行权衡。其中，保证厚度计算结果可靠性的关键，就在于合理控制界面反射率、优化实验条件并对环境扰动进行修正。

5.3.3 测试结果中多光束干涉的出现情况判断

查阅 Large-scale high aspect ratio Al-doped ZnO nanopillars arrays as anisotropic metamaterial.[2] 得知在所给波数范围，硅晶圆片折射率在 3.469 上下轻微波动，与空气差异明显，可认为反射率较大，高阶反射光光强不可忽视。同时，根据模型拟合得到的反射率数据，在所给波数范围中近似于 0.22>0.1, 故认为高阶反射光光强不可忽视。

5.3.4 多光束干涉条件下的厚度计算数学模型

假定外延层样品在测试过程中确实发生了多光束干涉。其物理过程可描述为：入射红外光进入外延层后，在空气–外延层界面以及外延层–衬底界面之间多次反射和透射，产生一系列相干的反射光束。所有这些反射光束在探测端叠加，从而形成干涉条纹。为了建立数学模型，我们对干涉过程进行如下分析与推导。

首先考虑光束的传播与相位累积。设空气折射率为 n_0，外延层折射率为 n_1，衬底折射率为 n_2，入射角为 θ，折射角为 ϕ。由于实验中入射角仅为 10° 与 15°，其余弦值接近 1，故可近似认为

$$\cos\phi \approx 1.$$

在这种近似下，光在外延层内往返一次的等效光程差为

$$\Delta L = 2n_1 d\cos\phi \approx 2n_1 d,$$

因此两束相邻反射光的相位差可表示为

$$\Delta\varphi = \frac{2\pi}{\lambda}\Delta L + \delta = \frac{4\pi n_1 d}{\lambda} + \delta,$$

其中 δ 表示可能的半波损失。

接下来分析反射光的振幅变化规律。设空气–外延层界面的振幅反射系数为 r_1，外延层–衬底界面的振幅反射系数为 r_2，空气–外延层界面的透射系数分别为 t_1（空气入射时）和 t_2（外延层入射时）。则各级反射光的振幅表达式如下：

第一束反射光（由空气–外延层界面反射）：

$$E_1 = r_1.$$

第二束反射光（入射光经透射进入外延层，在外延层–衬底界面反射一次，再透射出界面）：

$$E_2 = t_1 r_2 t_2 e^{i\Delta\varphi}.$$

第三束反射光（在外延层–衬底界面反射两次）：

$$E_3 = t_1 r_2 t_2 \cdot (r_1 r_2 e^{i\Delta\varphi}) \cdot e^{i\Delta\varphi}.$$

由此可见，除第一束反射光外，其余各级反射光之间的关系均满足公比为 $r_1 r_2 e^{i\Delta\varphi}$ 的几何级数。于是，总反射振幅为

$$E = r_1 + \frac{t_1 r_2 t_2 e^{i\Delta\varphi}}{1 - r_1 r_2 e^{i\Delta\varphi}}.$$

实验中可测得的量为反射率 $R(\tilde{v})$，即反射光强与入射光强之比。由于反射光强与振幅平方成正比，因此有

$$R(\tilde{v}) = |E|^2.$$

经过代数化简，可得

$$R(\tilde{v}) = \frac{R_1 + R_2 + 2\sqrt{R_1 R_2}\cos\Delta\varphi}{1 + R_1 R_2 + 2\sqrt{R_1 R_2}\cos\Delta\varphi},$$

其中 $R_1 = r_1^2$，$R_2 = r_2^2$。从该表达式可以看出，$R(\tilde{v})$ 随 $\Delta\varphi$ 变化呈现周期性振荡，其周期为 2π。

由干涉条纹的极值条件可知，任意相邻两个干涉极值所对应的相位差满足

$$\Delta\varphi_{k+1} - \Delta\varphi_k = 2\pi,$$

即

$$\frac{4\pi n_1 d}{\lambda_{k+1}} - \frac{4\pi n_1 d}{\lambda_k} = 2\pi.$$

换算为波数 $\tilde{v} = 1/\lambda$，则相邻条纹的波数间隔 $\Delta\tilde{v}$ 满足

$$2n_1 d\Delta\tilde{v} = 1,$$

由此可解得外延层厚度

$$d = \frac{1}{2n_1\Delta\tilde{v}}.$$

5.3.5 多光束干涉条件下的厚度计算算法

由数学模型部分可知，在多光束干涉条件下，反射率函数可表示为

$$R(\tilde{v}) = \frac{R_1 + R_2 + 2\sqrt{R_1 R_2}\cos\Delta\varphi}{1 + R_1 R_2 + 2\sqrt{R_1 R_2}\cos\Delta\varphi},$$

由于该函数同时依赖于波数与干涉相位，因此可以利用附件三与附件四的数据对外延层厚度 d 进行拟合。

为此，我们假设 $R_1(\tilde{v})$、$R_2(\tilde{v})$ 以及 $\Delta\varphi(\tilde{v})$ 分别满足关于波数的二次多项式近似，每个多项式包含三个待拟合参数。结合厚度参数 d，整个模型共涉及 10 个待拟合参数。

在拟合过程中，我们将每个附件中的数据随机划分为两部分，其中 80% 的数据用于参数拟合，20% 的数据用于检验模型的可靠性。具体拟合过程与问题二中的检验方法保持一致，采用 SciPy 库中的 `curve_fit` 函数实现最小二乘意义下的非线性拟合，从而获得最优参数。

计算结果表明：当入射角为 10° 时，外延层厚度为 **3.92** μm；当入射角为 15° 时，外延层厚度为 **3.77** μm。最终取其平均值，得到外延层厚度约为 **3.845** μm。

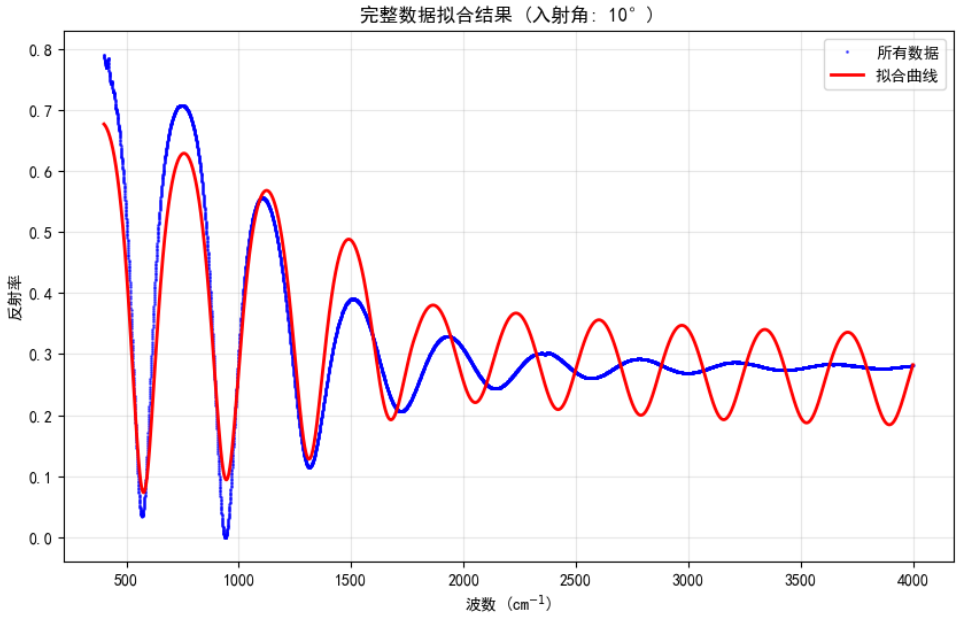

图 5: 附件 3 拟合结果

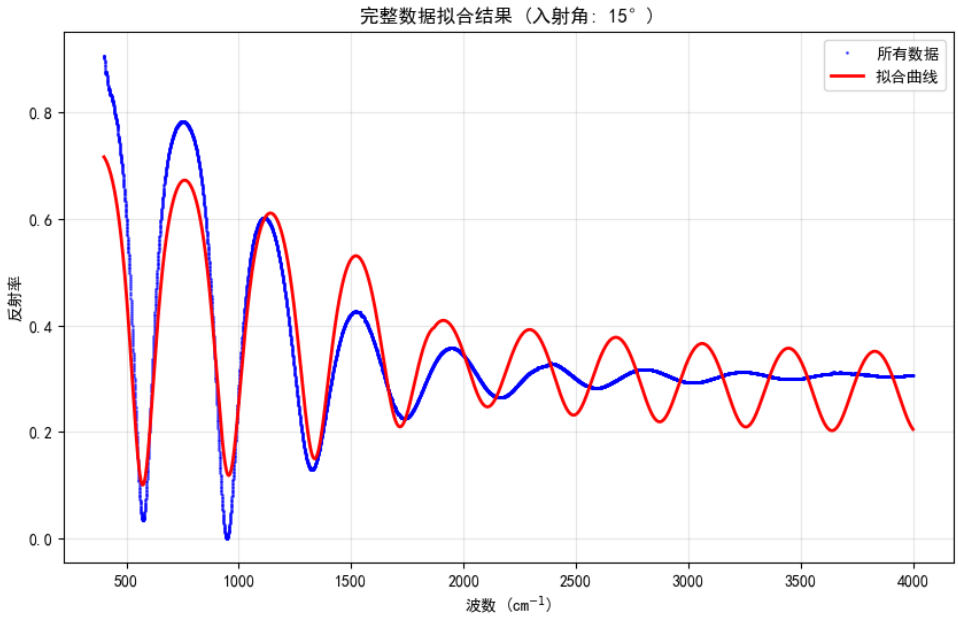

图 6: 附件 4 拟合结果

5.3.6 多光束干涉的影响评估

本文认为，多光束干涉也不会出现在碳化硅晶圆片的测试结果。理由如下：将附件一和附件二的数据带入多光束干涉模型的算法计算，发现拟合结果极差 (如下图)。因此在碳化硅晶原片中只考虑双光束干涉, 厚度依然为先前算的厚度平均值 d=8.0233 m。

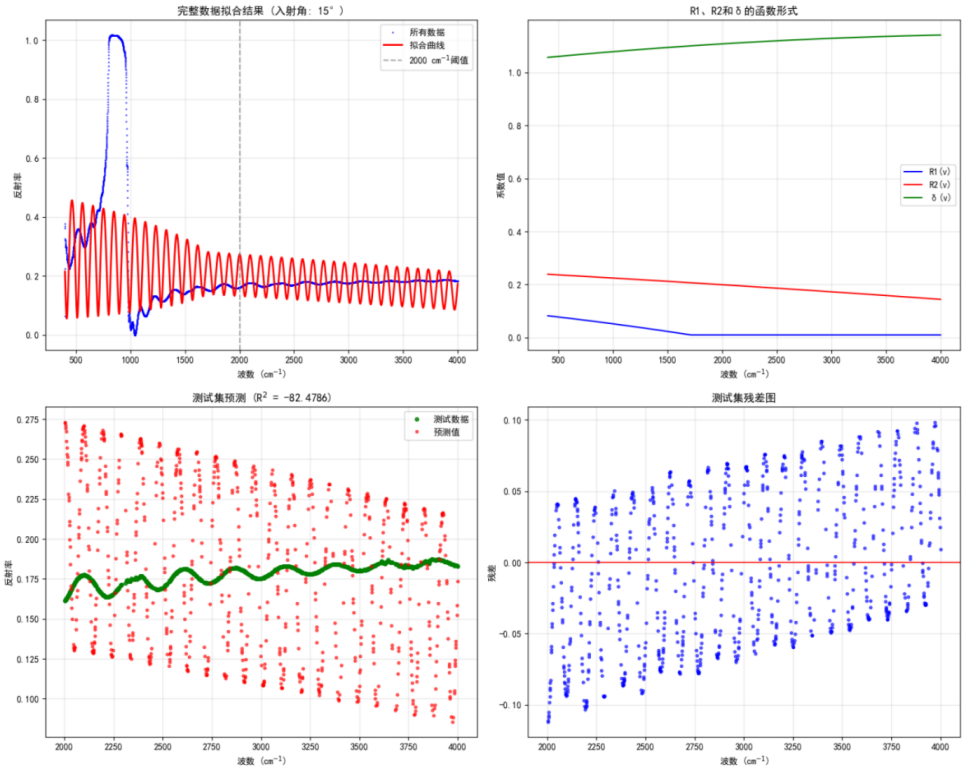

图 7: 拟合结果示意图

6 模型评价

6.1 问题一模型评价

本题所建立的干涉条纹厚度反演模型的核心公式

$$d = \frac{1}{2n_1 \cos\phi \, \Delta\tilde{v}}$$

能够直接将实验光谱中可观测的条纹间隔 $\Delta\tilde{v}$ 与外延层厚度建立一一对应关系。由于推导过程依赖于基础的光学干涉原理，模型的理论依据充分，复杂度低，便于在实际测试数据中快速应用。此外，模型仅需测量相邻条纹间隔即可完成厚度反演，对实验环境中的随机噪声具有一定的鲁棒性，这使得其在实际检测中具有较强的实用价值。

模型在推导过程中假定折射率 n_1 在小范围波数内为常数，并近似取 $\cos\phi \approx 1$，这在入射角较小的情况下是合理的，但在更大角度或更宽光谱范围内可能引入系统误差。若需进一步提高模型的适用性，可以在后续研究中考虑引入波长依赖的色散模型和更精细的入射角修正，以增强模型在不同实验条件下的稳定性。

6.2 问题二模型评价

针对问题二所建立的干涉光学厚度计算与拟合模型，总体而言具有以下优点：首先，在数据预处理环节引入中值滤波、Savitzky-Golay 平滑与基线校正，使得原始光谱数据的噪声得以有效抑制，为后续波峰检测提供了较为准确的基础。其次，采用 AMPD 算法进行峰值检测，能够稳健地识别出干涉条纹的位置，避免了传统方法中容易受局部噪声影响的缺陷。最终在结果可靠性分析中，我们引入了非线性拟合与交叉验证方法。通过运用以上建立起的数学模型，我们在最终结果中得出的 R^2 接近 0.98 且 RMSE 与 MAE 较小，充分表明模型的解释力与可靠性较高，体现出较强的鲁棒性。

同时也需要指出，模型仍存在一定局限性：例如，厚度计算依赖于相邻波峰差分，若局部数据点出现异常，仍可能对部分结果产生偏差。

总体来看，该模型在数据处理、峰值检测与厚度估计等方面较好地结合了物理机制与数值方法，能够较为稳定地完成问题二所要求的建模与求解任务。

6.3 问题三模型评价

该模型由干涉机理出发，结合相位差与光程差的关系建立了反射率与波数之间的函数关系式，并通过非线性拟合获得厚度，具有较好的物理解释性和数学可操作性，能够在多光束干涉显著存在时给出合理的计算结果。附件 3 和附件 4 的拟合结果显示, 当入射角分别为 10° 和 15° 时，得到的厚度值分别为 $3.92\,\mu$m 与 $3.77\,\mu$m, 平均值约为 $3.845\,\mu$m, 验证了方法的有效性。

另一方面，将碳化硅晶圆片的实验数据代入该模型后，拟合效果明显下降，说明该类样品中主要表现为双光束干涉，而非多光束干涉。由此可见，该模型适用于满足厚度与波长同量级、外延层-衬底界面反射率较大的情况，但在反射率不足或条纹对比度较低时，适用性受到限制。

从统计学的角度来看，该 R^2 值表明所选自变量集对因变量具备了较强的解释能力。仅余约 20% 的变异可由模型之外的其他随机因素所解释。这一结果初步证实本文多波束模型的合理性。

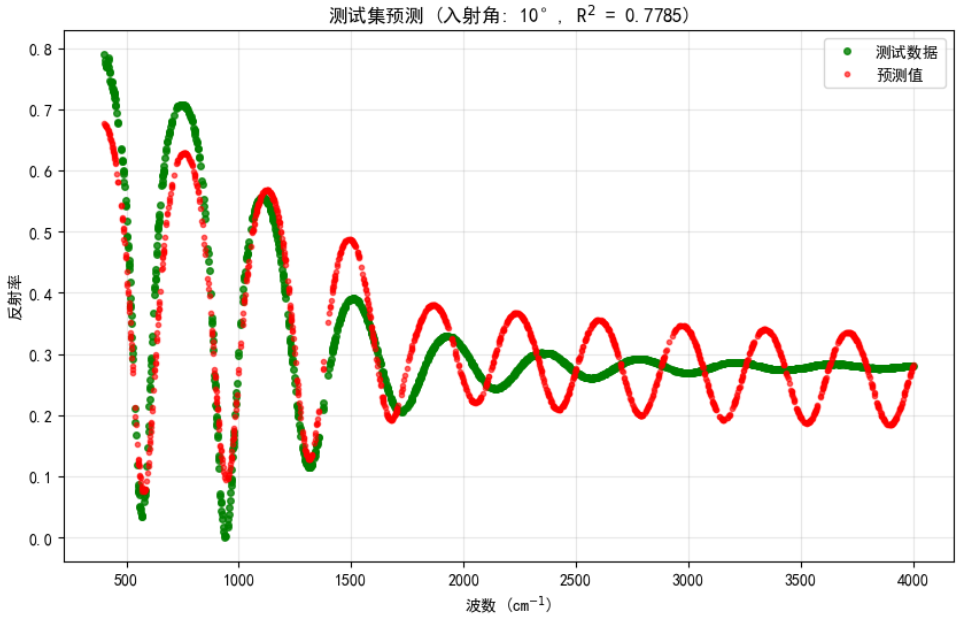

图 8: 附件 3 R^2 分析

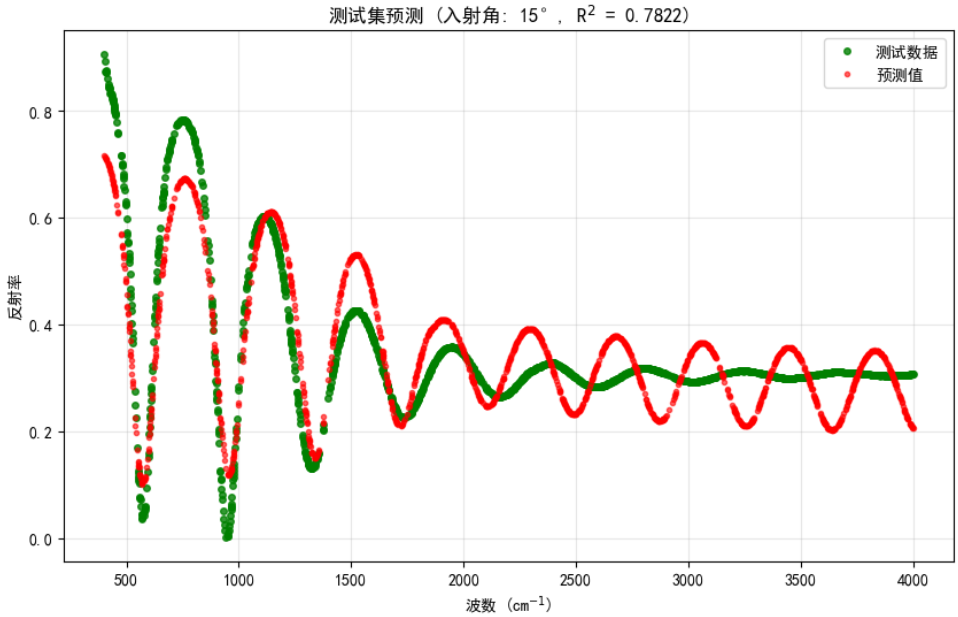

图 9: 附件 2 R^2 分析

参考文献

[1] S. Wang, M. Zhan, G. Wang, H. Xuan, W. Zhang, C. Liu, C. Xu, Y. Liu, Z. Wei, X. Chen. 4H-SiC: A new nonlinear material for midinfrared lasers. Laser Photonics Rev. 7, 831–838 (2013)

[2] E. Shkondin, O. Takayama, M. E. Aryaee Panah, P. Liu, P. V. Larsen, M. D. Mar, F. Jensen, A. V. Lavrinenko. Large-scale high aspect ratio Al-doped ZnO nanopillars arrays as anisotropic metamaterials. Opt. Mater. Express 7, 1606-1627 (2017) (Numerical data kindly provided by Osamu Takayama)

[3] 吴昌敏, 阮丽浓. 用红外干涉法测量薄膜厚度 [J]. 光学技术, 1985, (02): 28-29.

[4] 用红外干涉法测量砷化镓外延层的厚度 [J]. 仪器仪表通讯, 1973, (02): 7-10+6.

A 附件清单

A.1 附录 A: 双光束干涉确定外延层厚度算法.py

A.2 附录 B: 双光束干涉可靠性检验.py

A.3 附录 C: 多光束干涉确定外延层厚度算法.py

Listing 1: 双光束干涉确定外延层厚度算法完整代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import sympy as sp
from scipy.ndimage import median_filter
from scipy.signal import savgol_filter
import os
from pathlib import Path


plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 读取数据
def load_data(file_path):
    df = pd.read_excel(file_path)
    wavenumber = df['波数 (cm-1)'].values
    reflectance = df['反射率 (%)'].values
    return wavenumber, reflectance

def preprocess_reflectance(wavenumber, reflectance, visualize=False):
    """
    对反射率数据进行清洗和预处理
    """
    # 1. 中值滤波去除尖峰噪声
    reflectance_clean = median_filter(reflectance, size=5)

    # 2. Savitzky-Golay平滑
    reflectance_smooth = savgol_filter(reflectance_clean,
        window_length=11, polyorder=3)

    # 3. 简单的基线校正
    reflectance_baseline = reflectance_smooth - np.min(
        reflectance_smooth)

    if visualize:
        plt.figure(figsize=(10, 6))
        plt.plot(wavenumber, reflectance, 'purple', alpha=0.6, label=
            '原始数据')
        plt.plot(wavenumber, reflectance_clean, 'orange', alpha=0.7,
            label='中值滤波')
```

```
        plt.plot(wavenumber, reflectance_smooth, 'green', alpha=0.7,
            label='平滑后')
        plt.plot(wavenumber, reflectance_baseline, 'red', linewidth
            =1.5, label='最终处理')

        # AMPD算法检测峰值
        peaks = AMPD(reflectance_baseline)

        # 在图像上标注峰值点
        plt.scatter(wavenumber[peaks], reflectance_baseline[peaks],
                    color='blue', marker='x', s=50, zorder=5,
                    label=f'检测到的峰值 ({len(peaks)}个)')

        # 为每个峰值添加数值标注
        for i, peak in enumerate(peaks):
            plt.annotate(f'{wavenumber[peak]:.1f}',
                         xy=(wavenumber[peak], reflectance_baseline[
                             peak]),
                         xytext=(5, 10), textcoords='offset points',
                         fontsize=8, alpha=0.8)

        plt.xlabel('波数 (1/cm)')
        plt.ylabel('反射率 (%)')
        plt.title('光谱数据预处理过程及峰值检测')
        plt.legend()
        plt.grid(True, alpha=0.3)
        plt.show()

    return reflectance_baseline

# 斯涅尔定律计算折射角
def calculate_refraction_angle(n0, n1, theta):
    theta_rad = math.radians(theta)
    phi_rad = math.asin(n0 * math.sin(theta_rad) / n1)
    return phi_rad

# 定义Sellmeier方程的折射率计算
def calculate_refractive_index(wavelength):
      = sp.symbols('lambda')
    term1 = (0.20075 *  **2) / ( **2 + 12.07224)
    term2 = (5.54861 *  **2) / ( **2 - 0.02641)
    term3 = (35.65066 *  **2) / ( **2 - 1268.24708)
    n_squared_minus_1 = term1 + term2 + term3
    n_squared = n_squared_minus_1 + 1
    n_expression = sp.sqrt(n_squared)
    n_value = n_expression.subs( , wavelength*10000)
    return float(n_value)

# AMPD波峰检测函数
def AMPD(data):
    p_data = np.zeros_like(data, dtype=np.int32)
```

```
    count = data.shape[0]
    arr_rowsum = []
    for k in range(1, count // 2 + 1):
        row_sum = 0
        for i in range(k, count - k):
            if data[i] > data[i - k] and data[i] > data[i + k]:
                row_sum -= 1
        arr_rowsum.append(row_sum)
    min_index = np.argmin(arr_rowsum)
    max_window_length = min_index
    for k in range(1, max_window_length + 1):
        for i in range(k, count - k):
            if data[i] > data[i - k] and data[i] > data[i + k]:
                p_data[i] += 1
    return np.where(p_data == max_window_length)[0]

# 计算厚度（剔除异常值的平均厚度）
def calculate_thickness(wavenumbers, reflectance, n0=1, theta=10):
    # 数据预处理
    reflectance_processed = preprocess_reflectance(wavenumbers,
       reflectance, visualize=True)
    peaks = AMPD(reflectance_processed)
    peak_wavenumbers = wavenumbers[peaks]
    print("波峰位置的波数: ", peak_wavenumbers)
    peak_wavenumbers = peak_wavenumbers[1:]

    thicknesses = []
    thickness_details = []
    for i in range(1, len(peak_wavenumbers)):
        wavelength = 1 / peak_wavenumbers[i]
        n1 = calculate_refractive_index(wavelength)
        print(f"波长 {wavelength:.6f} cm 处的折射率: {n1:.6f}")
        phi = calculate_refraction_angle(n0, n1, theta)
        delta_v = peak_wavenumbers[i] - peak_wavenumbers[i - 1]
        thickness = 1 / (2 * n1 * math.cos(phi) * delta_v)
        thicknesses.append(thickness)
        thickness_details.append({
            'peak_pair': (i-1, i),
            'wavenumber1': peak_wavenumbers[i-1],
            'wavenumber2': peak_wavenumbers[i],
            'delta_v': delta_v,
            'wavelength': wavelength,
            'refractive_index': n1,
            'refraction_angle': math.degrees(phi),
            'thickness': thickness
        })

    if thicknesses:
        avg_thickness = np.mean(thicknesses)#箱线图法剔除异常值
        Q1 = np.percentile(thicknesses, 25)
        Q3 = np.percentile(thicknesses, 75)
```

```
        IQR = Q3 - Q1
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        filtered = [t for t in thicknesses if lower <= t <= upper]
        if filtered:
            avg_thickness_filtered = np.mean(filtered)
        else:
            avg_thickness_filtered = avg_thickness
        return avg_thickness, avg_thickness_filtered,
           thickness_details, lower, upper  # 新增返回边界值
    else:
        return None, None, None, None, None

#绘制原始光谱和检测到的波峰
def plot_spectrum_with_peaks(wavenumber, reflectance, peaks,
   file_name, theta):
    """
    绘制原始光谱和检测到的波峰位置
    """
    plt.figure(figsize=(12, 6))
    plt.plot(wavenumber, reflectance, 'b-', linewidth=1, label='反射
       率光谱')

    # 标记波峰位置
    peak_wavenumbers = wavenumber[peaks]
    peak_reflectance = reflectance[peaks]
    plt.plot(peak_wavenumbers, peak_reflectance, 'ro', markersize=8,
             label=f'检测到的波峰 (共{len(peaks)}个)')

    # 在波峰位置添加波数值标注
    for i, (w, r) in enumerate(zip(peak_wavenumbers, peak_reflectance
       )):
        plt.text(w, r + 0.02, f'{w:.1f}', ha='center', va='bottom',
           fontsize=8,
                 bbox=dict(boxstyle="round,pad=0.3", facecolor="yellow
                    ", alpha=0.7))

    plt.xlabel('波数 (1/cm)')
    plt.ylabel('反射率 (%)')
    plt.title(f'{file_name} - 入射角 {theta}° 光谱波峰检测结果')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# 可视化厚度结果（异常值显示红色,其他显示蓝色）
def plot_thickness_results(thickness_details, file_name, avg_raw,
   avg_filtered, lower, upper):
    if not thickness_details:
        return

```

```
    peak_pairs = [f"{i+1}" for i in range(len(thickness_details))]
    thickness_values = [detail['thickness'] for detail in
       thickness_details]

    # 判断哪些是异常值（红色），哪些是正常值（天蓝色）
    colors = ['red' if not (lower <= t <= upper) else 'skyblue' for t
        in thickness_values]

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    bars = ax1.bar(peak_pairs, thickness_values, alpha=0.7, color=
       colors)  # 使用不同颜色
    ax1.axhline(y=avg_raw, color='red', linestyle='--', label=f'原始
       均值: {avg_raw:.6f} cm')
    ax1.axhline(y=avg_filtered, color='green', linestyle='--', label=
       f'去异常均值: {avg_filtered:.6f} cm')
    ax1.axhline(y=lower, color='orange', linestyle=':', alpha=0.7,
       label=f'下边界: {lower:.6f} cm')
    ax1.axhline(y=upper, color='purple', linestyle=':', alpha=0.7,
       label=f'上边界: {upper:.6f} cm')

    ax1.set_xlabel('相邻波峰对编号')
    ax1.set_ylabel('厚度 (cm)')
    ax1.set_title(f'{file_name} - 各相邻波峰对厚度值 (红色为异常值)')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    for bar, value in zip(bars, thickness_values):
        ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
           max(thickness_values)*0.01,
                f'{value:.6f}', ha='center', va='bottom', fontsize=8)

    ax2.boxplot(thickness_values, vert=True)
    ax2.scatter([1] * len(thickness_values), thickness_values, alpha
       =0.6, color='red')
    ax2.set_ylabel('厚度 (cm)')
    ax2.set_title(f'{file_name} - 厚度值分布 (箱线图)')
    ax2.set_xticks([1])
    ax2.set_xticklabels(['厚度值'])
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# 主函数
def main():
     # 获取当前脚本所在目录
    current_dir = Path(__file__).parent

    # 构建文件路径
    file_path1 = current_dir / 'test1.xlsx'
```

```
    file_path2 = current_dir / 'test2.xlsx'

    wavenumber1, reflectance1 = load_data(file_path1)
    wavenumber2, reflectance2 = load_data(file_path2)

    print("\n文件1的厚度计算 (入射角 10°):")
    reflectance1_processed = preprocess_reflectance(wavenumber1,
        reflectance1, visualize=False)
    peaks1 = AMPD(reflectance1_processed)
    # 绘制原始光谱和波峰
    plot_spectrum_with_peaks(wavenumber1, reflectance1, peaks1, "
        test1.xlsx", 10)
    avg1_raw, avg1_filtered, details1, lower1, upper1 =
        calculate_thickness(wavenumber1, reflectance1, theta=10)
    print(f"文件1原始平均厚度 = {avg1_raw:.8f} cm, 去异常平均厚度 = {
        avg1_filtered:.8f} cm")
    plot_thickness_results(details1, "test1.xlsx (入射角 10°)",
        avg1_raw, avg1_filtered, lower1, upper1)

    print("\n文件2的厚度计算 (入射角 15°):")
    reflectance2_processed = preprocess_reflectance(wavenumber2,
        reflectance2, visualize=False)
    peaks2 = AMPD(reflectance2_processed)
    # 绘制原始光谱和波峰
    plot_spectrum_with_peaks(wavenumber2, reflectance2, peaks2, "
        test2.xlsx", 15)


    avg2_raw, avg2_filtered, details2, lower2, upper2 =
        calculate_thickness(wavenumber2, reflectance2, theta=15)
    print(f"文件2原始平均厚度 = {avg2_raw:.8f} cm, 去异常平均厚度 = {
        avg2_filtered:.8f} cm")
    plot_thickness_results(details2, "test2.xlsx (入射角 15°)",
        avg2_raw, avg2_filtered, lower2, upper2)

    if details1 and details2:
        print("\n两个入射角结果的比较:")
        print("=" * 50)
        print(f"入射角 10° 的原始平均厚度: {avg1_raw:.8f} cm, 去异常:
            {avg1_filtered:.8f} cm")
        print(f"入射角 15° 的原始平均厚度: {avg2_raw:.8f} cm, 去异常:
            {avg2_filtered:.8f} cm")

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

Listing 2: 双光束干涉可靠性检验完整代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import math
import sympy as sp
from scipy.ndimage import median_filter
from scipy.signal import savgol_filter
import os
from pathlib import Path

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 读取数据
def load_data(file_path):
    df = pd.read_excel(file_path)
    wavenumber = df['波数 (cm-1)'].values
    reflectance = df['反射率 (%)'].values
    return wavenumber, reflectance

# 数据预处理函数
def preprocess_reflectance(wavenumber, reflectance, visualize=False):
    """
    对反射率数据进行清洗和预处理
    """
    # 1. 中值滤波去除尖峰噪声
    reflectance_clean = median_filter(reflectance, size=5)

    # 2. Savitzky-Golay平滑
    reflectance_smooth = savgol_filter(reflectance_clean,
       window_length=11, polyorder=3)

    # 3. 简单的基线校正（减去最小值）
    reflectance_baseline = reflectance_smooth - np.min(
       reflectance_smooth)

    if visualize:
        plt.figure(figsize=(10, 6))
        plt.plot(wavenumber, reflectance, 'purple', alpha=0.6, label=
           '原始数据')
        plt.plot(wavenumber, reflectance_clean, 'orange', alpha=0.7,
           label='中值滤波')
        plt.plot(wavenumber, reflectance_smooth, 'green', alpha=0.7,
           label='平滑后')
        plt.plot(wavenumber, reflectance_baseline, 'red', linewidth
           =1.5, label='最终处理')

        # 使用AMPD算法检测峰值
        peaks = AMPD(reflectance_baseline)

        # 在图像上标注峰值点
```

```
        plt.scatter(wavenumber[peaks], reflectance_baseline[peaks],
                    color='blue', marker='x', s=50, zorder=5,
                    label=f'检测到的峰值 ({len(peaks)}个)')

        # 为每个峰值添加数值标注
        for i, peak in enumerate(peaks):
            plt.annotate(f'{wavenumber[peak]:.1f}',
                         xy=(wavenumber[peak], reflectance_baseline[
                            peak]),
                         xytext=(5, 10), textcoords='offset points',
                         fontsize=8, alpha=0.8)

        plt.xlabel('波数 (1/cm)')
        plt.ylabel('反射率 (%)')
        plt.title('光谱数据预处理过程及峰值检测')
        plt.legend()
        plt.grid(True, alpha=0.3)
        plt.show()

    return reflectance_baseline

# AMPD波峰检测函数
def AMPD(data):
    p_data = np.zeros_like(data, dtype=np.int32)
    count = data.shape[0]
    arr_rowsum = []
    for k in range(1, count // 2 + 1):
        row_sum = 0
        for i in range(k, count - k):
            if data[i] > data[i - k] and data[i] > data[i + k]:
                row_sum -= 1
        arr_rowsum.append(row_sum)
    min_index = np.argmin(arr_rowsum)
    max_window_length = min_index
    for k in range(1, max_window_length + 1):
        for i in range(k, count - k):
            if data[i] > data[i - k] and data[i] > data[i + k]:
                p_data[i] += 1
    return np.where(p_data == max_window_length)[0]

# Sellmeier方程计算折射率
def calculate_refractive_index(wavelength):
    """
    计算碳化硅在特定波长下的折射率
    wavelength: 波长 (cm)
    """
      = sp.symbols('lambda')
    term1 = (0.20075 *  **2) / ( **2 + 12.07224)
    term2 = (5.54861 *  **2) / ( **2 - 0.02641)
    term3 = (35.65066 *  **2) / ( **2 - 1268.24708)
    n_squared_minus_1 = term1 + term2 + term3
```

```
    n_squared = n_squared_minus_1 + 1
    n_expression = sp.sqrt(n_squared)

    # 将波长从cm转换为 m
    wavelength_um = wavelength * 10000
    n_value = n_expression.subs( , wavelength_um)
    return float(n_value)

# 提高计算效率，创建折射率计算的向量化版本
calculate_refractive_index_vec = np.vectorize(
    calculate_refractive_index)

# 干涉模型函数
def interference_model(v, a0, a1, a2, b0, b1, b2, p0, p1, p2, d,
    theta_deg):
    """
    干涉反射率模型
    v: 波数 (cm ¹)
    d: 薄膜厚度 (cm)
    theta_deg: 入射角 (度)
    """
    # 将角度转换为弧度
    theta_rad = np.deg2rad(theta_deg)

    # 缓变函数 A(v), B(v), Psi(v)
    A_v = a0 + a1 * v + a2 * (v**2)
    B_v = b0 + b1 * v + b2 * (v**2)
    Psi_v = p0 + p1 * v + p2 * (v**2)

    wavelength = 1 / v

    # 计算折射率 n(v)
    n_v = calculate_refractive_index_vec(wavelength)

    # 计算主相位差 Δ(v)
    Delta_v = 4 * np.pi * n_v * d * np.cos(theta_rad) * v

    # 计算总相位并返回反射率 R(v)
    total_phase = Delta_v + Psi_v
    R_v = B_v + A_v * np.cos(total_phase)

    return R_v

# 包装函数用于curve_fit
def model_wrapper(v, a0, a1, a2, b0, b1, b2, p0, p1, p2):
    return interference_model(v, a0, a1, a2, b0, b1, b2, p0, p1, p2,
        d_fixed, theta_fixed)

def main():
    global d_fixed, theta_fixed

```

```
148     # 获取当前脚本所在目录
149     current_dir = Path(__file__).parent
150 
151     # 构建文件路径
152     file_path1 = current_dir / 'test1.xlsx'
153     file_path2 = current_dir / 'test2.xlsx'
154 
155     # 已知厚度和入射角
156     d1 = 0.00080074  # cm (文件1的厚度)
157     d2 = 0.00080396  # cm (文件2的厚度)
158     theta1 = 10      # 度 (文件1的入射角)
159     theta2 = 15      # 度 (文件2的入射角)
160 
161     # 处理文件1
162     print("处理文件1 (入射角 10°)...")
163     wavenumber1, reflectance1 = load_data(file_path1)
164     process_file(wavenumber1, reflectance1, d1, theta1, "文件1")
165 
166     # 处理文件2
167     print("\n处理文件2 (入射角 15°)...")
168     wavenumber2, reflectance2 = load_data(file_path2)
169     process_file(wavenumber2, reflectance2, d2, theta2, "文件2")
170 
171 def process_file(wavenumber, reflectance, d, theta, file_name):
172     global d_fixed, theta_fixed
173     d_fixed = d
174     theta_fixed = theta
175     reflectance_processed = preprocess_reflectance(wavenumber,
            reflectance, visualize=True)
176     peaks = AMPD(reflectance_processed)
177 
178     # 绘制原始光谱和检测到的波峰
179     plot_spectrum_with_peaks(wavenumber, reflectance_processed, peaks
            , file_name, theta)
180 
181     if len(peaks) >= 4:
182         fourth_peak_idx = peaks[3]
183         start_idx = fourth_peak_idx
184         print(f"找到第4个波峰在索引 {fourth_peak_idx}, 波数 {
                wavenumber[fourth_peak_idx]:.1f} cm ⁻¹")
185     else:
186         start_idx = int(len(wavenumber) * 0.3)  # 如果没有找到足够波
                峰, 使用30%作为起始点
187         print(f"未找到足够波峰, 从索引 {start_idx} 开始")
188 
189     # 截取数据 (从第四个波峰开始)
190     wavenumber_cut = wavenumber[start_idx:]
191     reflectance_cut = reflectance_processed[start_idx:]  # 使用预处理
            后的数据
192 
193     #随机划分训练集(80%)和测试集(20%)
```

```
    wavenumber_train, wavenumber_test, reflectance_train,
        reflectance_test = train_test_split(
        wavenumber_cut, reflectance_cut, test_size=0.2, random_state
            =42
    )

    print(f"训练集大小: {len(wavenumber_train)}, 测试集大小: {len(
        wavenumber_test)}")

    # 初始参数猜测
    initial_guess = [
        (np.max(reflectance_train) - np.min(reflectance_train)) / 2,
            0, 0,  # A(v)系数: a0, a1, a2
        np.mean(reflectance_train), 0, 0,  # B(v)系数: b0, b1, b2
        0, 0, 0  # Psi(v)系数: p0, p1, p2
    ]

    print("开始拟合模型...")
    try:
        # 执行拟合
        popt, pcov = curve_fit(model_wrapper, wavenumber_train,
            reflectance_train,
                               p0=initial_guess, maxfev=5000)

        # 提取拟合参数
        a0, a1, a2, b0, b1, b2, p0, p1, p2 = popt
        print("拟合成功!")
        print(f"A(v)参数: a0={a0:.4f}, a1={a1:.6f}, a2={a2:.9f}")
        print(f"B(v)参数: b0={b0:.4f}, b1={b1:.6f}, b2={b2:.9f}")
        print(f"Psi(v)参数: p0={p0:.4f}, p1={p1:.6f}, p2={p2:.9f}")

        #使用测试集验证模型
        reflectance_pred = model_wrapper(wavenumber_test, *popt)

        # 计算评估指标
        r2 = r2_score(reflectance_test, reflectance_pred)
        rmse = np.sqrt(mean_squared_error(reflectance_test,
            reflectance_pred))
        mae = np.mean(np.abs(reflectance_test - reflectance_pred))

        print(f"\n模型验证结果:")
        print(f"R²分数: {r2:.6f}")
        print(f"RMSE: {rmse:.6f}")
        print(f"MAE: {mae:.6f}")

        #可视化结果
        plot_results(wavenumber, reflectance_processed,
            wavenumber_train, reflectance_train,
                     wavenumber_test, reflectance_test, popt, d, theta
                         , file_name,
                     start_idx, r2, rmse)
```

```
237
238     except Exception as e:
239         print(f"拟合失败: {e}")
240         import traceback
241         traceback.print_exc()
242
243 def plot_spectrum_with_peaks(wavenumber, reflectance, peaks,
        file_name, theta):
244     """
245     绘制原始光谱和检测到的波峰位置
246     """
247     plt.figure(figsize=(12, 6))
248     plt.plot(wavenumber, reflectance, 'b-', linewidth=1, label='反射
            率光谱')
249
250     # 标记波峰位置
251     peak_wavenumbers = wavenumber[peaks]
252     peak_reflectance = reflectance[peaks]
253     plt.plot(peak_wavenumbers, peak_reflectance, 'ro', markersize=8,
254              label=f'检测到的波峰 (共{len(peaks)}个)')
255
256     # 在波峰位置添加波数值标注
257     for i, (w, r) in enumerate(zip(peak_wavenumbers, peak_reflectance
            )):
258         plt.text(w, r + 0.02, f'{w:.1f}', ha='center', va='bottom',
                fontsize=8,
259                  bbox=dict(boxstyle="round,pad=0.3", facecolor="yellow
                        ", alpha=0.7))
260
261     plt.xlabel('波数 (1/cm)')
262     plt.ylabel('反射率 (%)')
263     plt.title(f'{file_name} - 入射角 {theta}° 光谱波峰检测结果')
264     plt.legend()
265     plt.grid(True, alpha=0.3)
266     plt.tight_layout()
267     plt.show()
268
269 def plot_results(wavenumber_full, reflectance_full, wavenumber_train,
        reflectance_train,
270                  wavenumber_test, reflectance_test, popt, d, theta,
                        file_name,
271                  start_idx, r2, rmse):
272     """
273     绘制拟合和验证结果
274     """
275     fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,
            12))
276
277     # 图1: 完整光谱和截取位置
278     ax1.plot(wavenumber_full, reflectance_full, 'b-', alpha=0.7,
            label='完整光谱')
```

```
    ax1.axvline(x=wavenumber_full[start_idx], color='r', linestyle='--',
                label=f'截取起始点: {wavenumber_full[start_idx]:.1f} cm⁻¹')
    ax1.set_xlabel('波数 (1/cm)')
    ax1.set_ylabel('反射率 (%)')
    ax1.set_title(f'{file_name} - 完整光谱和截取位置\n厚度: {d:.6f} cm, 入射角: {theta}°')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # 图2: 训练集拟合结果
    wavenumber_sorted = np.sort(wavenumber_train)
    reflectance_train_sorted = reflectance_train[np.argsort(wavenumber_train)]
    reflectance_pred_train = model_wrapper(wavenumber_sorted, *popt)

    ax2.plot(wavenumber_sorted, reflectance_train_sorted, 'bo', alpha=0.6,
             markersize=3, label='训练数据')
    ax2.plot(wavenumber_sorted, reflectance_pred_train, 'r-', linewidth=1.5,
             label='拟合曲线')
    ax2.set_xlabel('波数 (1/cm)')
    ax2.set_ylabel('反射率 (%)')
    ax2.set_title('训练集拟合结果')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # 图3: 测试集验证结果
    wavenumber_test_sorted = np.sort(wavenumber_test)
    reflectance_test_sorted = reflectance_test[np.argsort(wavenumber_test)]
    reflectance_pred_test = model_wrapper(wavenumber_test_sorted, *popt)

    ax3.plot(wavenumber_test_sorted, reflectance_test_sorted, 'go', alpha=0.6,
             markersize=4, label='测试数据')
    ax3.plot(wavenumber_test_sorted, reflectance_pred_test, 'r-', linewidth=1.5,
             label='预测曲线')
    ax3.set_xlabel('波数 (1/cm)')
    ax3.set_ylabel('反射率 (%)')
    ax3.set_title(f'测试集验证结果\nR*R = {r2:.4f}, RMSE = {rmse:.4f}')
    ax3.legend()
    ax3.grid(True, alpha=0.3)

    # 图4: 残差分析
    residuals = reflectance_test - model_wrapper(wavenumber_test, *
```

```
        popt)
    ax4.hist(residuals, bins=30, alpha=0.7, color='purple')
    ax4.axvline(x=0, color='r', linestyle='--')
    ax4.set_xlabel('残差')
    ax4.set_ylabel('频数')
    ax4.set_title('测试集残差分布')
    ax4.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 打印拟合的参数函数
    a0, a1, a2, b0, b1, b2, p0, p1, p2 = popt
    print(f"\n拟合的函数表达式:")
    print(f"A( ) = {a0:.6f} + {a1:.9f}•  + {a2:.12f}• ²")
    print(f"B( ) = {b0:.6f} + {b1:.9f}•  + {b2:.12f}• ²")
    print(f" ( ) = {p0:.6f} + {p1:.9f}•  + {p2:.12f}• ²")

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

Listing 3: 多光束干涉确定外延层厚度算法完整代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import medfilt, savgol_filter
from scipy.optimize import curve_fit
import math
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import os
from pathlib import Path

plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei', '
    DejaVu Sans']  # 支持中文的字体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 数据预处理函数
def preprocess_reflectance(wavenumber, reflectance, visualize=False):
    """
    对反射率数据进行清洗和预处理
    """
    # 1. 中值滤波去除尖峰噪声
    reflectance_clean = medfilt(reflectance, kernel_size=5)

    # 2. Savitzky-Golay平滑
    reflectance_smooth = savgol_filter(reflectance_clean,
        window_length=11, polyorder=3)

    # 3. 简单的基线校正（减去最小值）
    reflectance_baseline = reflectance_smooth - np.min(
```

```
        reflectance_smooth)

    if visualize:
        plt.figure(figsize=(10, 6))
        plt.plot(wavenumber, reflectance, 'b-', alpha=0.5, label='原始数据', linewidth=1)
        plt.plot(wavenumber, reflectance_smooth, 'r-', linewidth=2, label='预处理后数据')
        plt.xlabel('波数 (cm$^{-1}$)')
        plt.ylabel('反射率')
        plt.legend()
        plt.title('反射率数据预处理')
        plt.grid(True, alpha=0.3)
        plt.show()

    return reflectance_baseline

# 计算折射角函数
def calculate_refraction_angle(n0, n1, theta_deg):
    """
    计算折射角
    n0: 入射介质折射率 (空气 1)
    n1: 硅折射率
    theta_deg: 入射角 (度)
    """
    theta_rad = math.radians(theta_deg)
    phi_rad = math.asin(n0 * math.sin(theta_rad) / n1)
    return phi_rad

# 反射率模型函数 R1和R2为波数的二次函数
def reflectance_model(wavenumber, d, R1_a, R1_b, R1_c, R2_a, R2_b, R2_c, delta_a, delta_b, delta_c, n0, n1, theta_deg):
    """
    反射率模型函数（波数版本）
    wavenumber: 波数 (cm⁻¹)
    d: 厚度 (m)
    R1_a, R1_b, R1_c: R1 = R1_a + R1_b*v + R1_c*v² 的系数
    R2_a, R2_b, R2_c: R2 = R2_a + R2_b*v + R2_c*v² 的系数
    delta_a, delta_b, delta_c:   = delta_a + delta_b*v + delta_c*v² 的系数
    n0: 空气折射率
    n1: 硅折射率
    theta_deg: 入射角 (度)
    """
    # 转换为SI单位 (m⁻¹)
    wavenumber_si = wavenumber * 100
    # 计算R1和R2的二次函数形式，并确保在合理范围内
    R1 = R1_a + R1_b * wavenumber_si + R1_c * wavenumber_si**2
    R2 = R2_a + R2_b * wavenumber_si + R2_c * wavenumber_si**2

    # 确保R1和R2在合理范围内
```

```
    R1 = np.maximum(0.01, np.minimum(0.9, R1))
    R2 = np.maximum(0.01, np.minimum(0.9, R2))

    # 计算折射角
    phi_rad = calculate_refraction_angle(n0, n1, theta_deg)
    cos_phi = math.cos(phi_rad)

    # 计算几何相位差（使用波数）
    # Δ = 4 * n * d * cos *
    geometric_phase = 4 * np.pi * n1 * (d * 1e-6) * cos_phi *
        wavenumber_si

    # 计算附加相位项   = delta_a + delta_b*v + delta_c*v²
    delta_phase = delta_a + delta_b * wavenumber_si + delta_c *
        wavenumber_si**2

    # 总相位差 = 几何相位差 + 附加相位项
    delta_phi = geometric_phase + delta_phase

    # 计算反射率
    sqrt_R1R2 = np.sqrt(R1 * R2)
    numerator = R1 + R2 + 2 * sqrt_R1R2 * np.cos(delta_phi)
    denominator = 1 + R1 * R2 + 2 * sqrt_R1R2 * np.cos(delta_phi)

    return numerator / denominator

def create_fit_function(theta_deg):
    """
    创建带有固定theta_deg的拟合函数
    """
    def fit_function(wavenumber, d, R1_a, R1_b, R1_c, R2_a, R2_b,
        R2_c, delta_a, delta_b, delta_c):
        n0 = 1.0  # 空气折射率
        n1 = 3.469  # 硅折射率
        return reflectance_model(wavenumber, d, R1_a, R1_b, R1_c,
            R2_a, R2_b, R2_c,
                                 delta_a, delta_b, delta_c, n0, n1,
                                    theta_deg)

    return fit_function

# 改进的主程序（包含附加相位项 ）
def main_improved(file_path, theta_deg, visualize=True):
    """
    改进的主拟合程序，包含附加相位项
    """
    # 读取数据
    data = pd.read_excel(file_path)
    wavenumber = data.iloc[:, 0].values  # 波数数据，单位cm ⁻¹
    reflectance_raw = data.iloc[:, 1].values / 100  # 转换为0-1范围

```

```
    # 数据预处理
    reflectance = preprocess_reflectance(wavenumber, reflectance_raw,
        visualize=visualize)

    # 8:2随机分割数据
    X_train, X_test, y_train, y_test = train_test_split(
        wavenumber, reflectance, test_size=0.2, random_state=42,
           shuffle=True
    )

    # 创建拟合函数
    fit_func = create_fit_function(theta_deg)

    # 初始参数 - 包含附加相位项 的系数
    initial_guess = [
        4.5,        # d: 厚度 (m)
        0.3,        # R1_a: R1常数项
        -1e-6,      # R1_b: R1一次项系数
        1e-13,      # R1_c: R1二次项系数
        0.3,        # R2_a: R2常数项
        -1e-7,      # R2_b: R2一次项系数
        1e-13,      # R2_c: R2二次项系数
        1,          # delta_a:  常数项
        0.0,        # delta_b:  一次项系数
        0.0         # delta_c:  二次项系数
    ]

    # 参数边界
    bounds = (
        [1.0, 0.1, -1e-6, -1e-12, 0.1, -1e-6, -1e-12, -10.0, -1e-6,
           -1e-12],  # 下限
        [1000.0, 0.8, 1e-6, 1e-12, 0.8, 1e-6, 1e-12, 10.0, 1e-6, 1e
           -12]      # 上限
    )

    # 拟合
    try:
        print("开始拟合...")
        popt, pcov = curve_fit(
            fit_func,
            X_train,
            y_train,
            p0=initial_guess,
            bounds=bounds,
            maxfev=20000,  # 增加迭代次数
            method='trf'
        )

        # 拟合参数的提取
        d_fit, R1_a_fit, R1_b_fit, R1_c_fit, R2_a_fit, R2_b_fit,
           R2_c_fit, delta_a_fit, delta_b_fit, delta_c_fit = popt
```

```
166
167         # 所有数据预测值的计算
168         wavenumber_sorted = np.sort(wavenumber)
169         y_full_pred = fit_func(wavenumber_sorted, *popt)
170
171         # 计算R1、R2和 的函数值
172         wavenumber_si = wavenumber_sorted * 100
173         R1_values = R1_a_fit + R1_b_fit * wavenumber_si + R1_c_fit *
               wavenumber_si**2
174         R2_values = R2_a_fit + R2_b_fit * wavenumber_si + R2_c_fit *
               wavenumber_si**2
175         delta_values = delta_a_fit + delta_b_fit * wavenumber_si +
               delta_c_fit * wavenumber_si**2
176
177         R1_values = np.maximum(0.01, np.minimum(0.9, R1_values))
178         R2_values = np.maximum(0.01, np.minimum(0.9, R2_values))
179
180         # 计算训练集和测试集预测
181         y_train_pred = fit_func(X_train, *popt)
182         y_test_pred = fit_func(X_test, *popt)
183
184         # 拟合指标的计算
185         train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred
               ))
186         test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
187         train_r2 = r2_score(y_train, y_train_pred)
188         test_r2 = r2_score(y_test, y_test_pred)
189
190         print(f"\n拟合结果 - 入射角: {theta_deg}°")
191         print(f"厚度 d = {d_fit:.2f}  m")
192         print(f"R1系数: a={R1_a_fit:.4f}, b={R1_b_fit:.2e}, c={
               R1_c_fit:.2e}")
193         print(f"R2系数: a={R2_a_fit:.4f}, b={R2_b_fit:.2e}, c={
               R2_c_fit:.2e}")
194         print(f" 系数: a={delta_a_fit:.4f}, b={delta_b_fit:.2e}, c={
               delta_c_fit:.2e}")
195         print(f"训练集 RMSE: {train_rmse:.4f}, R²: {train_r2:.4f}")
196         print(f"测试集 RMSE: {test_rmse:.4f}, R²: {test_r2:.4f}")
197
198         # 可视化结果
199         if visualize:
200             fig, axes = plt.subplots(2, 2, figsize=(15, 12))
201
202             # 主图: 完整数据拟合结果
203             axes[0, 0].plot(wavenumber, reflectance, 'b.', alpha=0.6,
                    label='所有数据', markersize=2)
204             axes[0, 0].plot(wavenumber_sorted, y_full_pred, 'r-',
                   linewidth=2, label='拟合曲线')
205             axes[0, 0].set_xlabel('波数 (cm$^{-1}$)')
206             axes[0, 0].set_ylabel('反射率')
207             axes[0, 0].set_title(f'完整数据拟合结果 (入射角: {
```

```
                theta_deg}°)')
208             axes[0, 0].legend()
209             axes[0, 0].grid(True, alpha=0.3)
210
211             # R1和R2函数
212             axes[0, 1].plot(wavenumber_sorted, R1_values, 'b-', label
                    ='R1(v)')
213             axes[0, 1].plot(wavenumber_sorted, R2_values, 'r-', label
                    ='R2(v)')
214             axes[0, 1].plot(wavenumber_sorted, delta_values, 'g-',
                    label=' (v)')
215             axes[0, 1].set_xlabel('波数 (cm$^{-1}$)')
216             axes[0, 1].set_ylabel('系数值')
217             axes[0, 1].set_title('R1、R2和 的函数形式')
218             axes[0, 1].legend()
219             axes[0, 1].grid(True, alpha=0.3)
220
221             # 测试集预测
222             axes[1, 0].plot(X_test, y_test, 'go', alpha=0.8, label='
                    测试数据', markersize=4)
223             axes[1, 0].plot(X_test, y_test_pred, 'ro', alpha=0.6,
                    label='预测值', markersize=3)
224             axes[1, 0].set_xlabel('波数 (cm$^{-1}$)')
225             axes[1, 0].set_ylabel('反射率')
226             axes[1, 0].set_title(f'测试集预测 (R$^2$ = {test_r2:.4f})
                    ')
227             axes[1, 0].legend()
228             axes[1, 0].grid(True, alpha=0.3)
229
230
231
232         return {
233             'd': d_fit,
234             'R1_a': R1_a_fit, 'R1_b': R1_b_fit, 'R1_c': R1_c_fit,
235             'R2_a': R2_a_fit, 'R2_b': R2_b_fit, 'R2_c': R2_c_fit,
236             'delta_a': delta_a_fit, 'delta_b': delta_b_fit, 'delta_c'
                    : delta_c_fit,
237             'train_rmse': train_rmse,
238             'test_rmse': test_rmse,
239             'train_r2': train_r2,
240             'test_r2': test_r2
241         }
242
243     except Exception as e:
244         print(f"拟合过程中出现错误: {e}")
245         import traceback
246         traceback.print_exc()
247         return None
248
249 # 运行两个角度的拟合
250 if __name__ == "__main__":
```

```
    # 文件路径
      # 获取当前脚本所在目录
    current_dir = Path(__file__).parent

    # 构建文件路径
    file_10deg = current_dir / 'test1.xlsx'
    file_15deg = current_dir / 'test2.xlsx'

    # 拟合10度数据
    print("=" * 50)
    print("开始拟合10°入射角数据...")
    results_10deg = main_improved(file_10deg, 10, visualize=True)

    # 拟合15度数据
    print("=" * 50)
    print("开始拟合15°入射角数据...")
    results_15deg = main_improved(file_15deg, 15, visualize=True)

    # 比较两个角度的结果
    if results_10deg and results_15deg:
        print("=" * 50)
        print("两个角度拟合结果比较:")
        print(f"10°厚度: {results_10deg['d']:.2f}  m")
        print(f"15°厚度: {results_15deg['d']:.2f}  m")
        print(f"厚度差异: {abs(results_10deg['d'] - results_15deg['d
            ']):.2f}  m")

        # 计算平均厚度
        avg_d = (results_10deg['d'] + results_15deg['d']) / 2
        print(f"平均厚度: {avg_d:.2f}  m")
```