基于 Drude-Sellmeier 模型的碳化硅外延层厚度的反演算法

摘　要

碳化硅是一种先进的半导体材料，其外延层厚度对器件性能具有重要影响。为实现对外延层厚度的准确测量，本文基于红外干涉法，结合实验数据，建立外延层厚度计算的数学模型，并进行数值求解。在此基础上进一步分析多光束干涉对测量精度的影响，并提出优化方法以提高模型的可靠性。

对于问题 1，在仅考虑两束反射光线的情况下，我们首先通过几何光学分析以及干涉条件，在外延层厚度为常数的条件下，推导出其厚度的解析公式。更进一步地，考虑到外延层折射率会随载流子浓度等因素变化，本文引入 Drude-Sellmeier 模型，将厚度、载流子浓度与散射率进行耦合，得到其隐式表达式，再通过数值迭代算法，在参数空间中搜索最优解，使得理论干涉级数与实验光谱极值点最大可能性吻合，最终反演得出外延层厚度。

对于问题 2，我们需要在问题 1 建立的 Drude-Sellmeier 模型基础上，解决在外延层厚度、载流子浓度以及散射率三者耦合情况下的参数反演问题。首先我们通过小波变换去噪和峰值检测提取数据中的干涉极值点，然后定义基于干涉级数残差平方和的目标函数，使理论计算结果与实验干涉条件的偏差最小化，最后为避免陷入局部最小值，我们选用差分进化算法作为全局优化方法，从而获得厚度、载流子浓度和散射率的最优解，并通过数值实验验证了模型的稳定性。

对于问题 3，我们需要解决碳化硅外延层厚度计算中多光束干涉的影响。我们首先分析了多光束干涉的必要条件，此外，对于多光束干涉，问题 1、问题 2 中外延层厚度计算模型存在局限性，因此，我们通过全谱拟合来建立更为精确的外延层厚度计算模型。此方法不再依赖极值点，而是利用传输矩阵法对整个光谱进行拟合。模型中采用 Drude-Sellmeier 模型描述外延层和衬底的复数介电函数，同时考虑了折射率实部对相位的影响以及虚部对干涉条纹幅度的作用。最后通过构造全谱残差平方和目标函数，结合数值优化算法实现厚度、外延层参数的拟合。并通过数值实验，验证了该模型的可靠性和适应性。

本研究通过对三个问题的逐步分析与求解，建立了从简化模型到复杂模型的完整外延层厚度计算框架。从理论推导到数值优化，再到全谱拟合，逐步提升了碳化硅外延层厚度测量的精度与适用性，为碳化硅器件性能表征与工艺优化提供了可靠的理论与方法支持。

关键词：小波去噪；Drude-Sellmeier 模型；差分优化算法；全谱拟合；矩阵传输法

一、问题重述

1.1 问题背景

碳化硅作为第三代宽禁带半导体材料，具有高热导率、高击穿电场强度以及优异的化学稳定性，已广泛应用于高功率和高频电子器件的制造过程中。其外延层厚度是决定碳化硅器件性能的关键参数之一，因此在不破坏外延层的情况下，建立一套科学、准确的碳化硅外延层厚度测试标准尤为重要。本研究采用红外干涉法进行测量，利用光束在外延层表面与衬底表面所产生的两束反射光相互干涉，通过研究干涉光谱并建立数学模型来推算出外延层的厚度。此外，多光束干涉对外延层厚度的计算也会产生影响，因此需要对模型进行完善与优化来确保结果的可靠性。

1.2 问题提出

题目中给出了红外干涉法的测量原理[1]，以及外延层厚度测量原理示意图。此外，题目提供了四个条件，分别是附件 1 和附件 2(入射角分别为 10° 和 15° 时针对同一块碳化硅晶圆片的光谱实测数据)、附件 3 和附件 4(入射角分别为 10° 和 15° 时针对同一块硅晶圆片的光谱实测数据)，四个附件第一列均为波数，第二列均为反射率。

根据上述题目背景和附件信息，我们需建立外延层厚度测量的数学模型来解决以下问题：

（1）仅考虑外延层和衬底界面只有一次反射、透射的情况下得到的反射光所产生的干涉条纹，基于这一理想化模型分析干涉条纹与外延层之间的厚度关系，建立确定外延层厚度的数学模型。

（2）在问第一建立的数学模型基础上进一步设计确定外延层厚度的具体算法，利用该算法，将附件 1、附件 2 提供的碳化硅晶圆片的光谱实测数据带入模型，得到外延层厚度的计算结果，并分析结果的可靠性。

（3）在实际情况中，光波会在外延层和衬底表面形成多次透射和反射[2]，从而产生多光束干涉。需探讨多光束干涉形成的条件，并考虑这一效应对计算精度的影响，并进一步确定硅外延层厚度计算的数学模型和算法。

二、问题分析

问题的研究对象是碳化硅外延层，研究的核心任务是建立测量外延层厚度的数学模型，并解决题目中的三个问题。我们将这三个层层递进的问题进行分解与分析，明确其数学本质和相互关系。

问题 1：此问题要求我们建立外延层厚度的数学模型。题目给出了红外干涉法[15]

的工作原理和外延层测量原理示意图，题干中规定了只考虑外延层和衬底界面只有一次反射、透射所产生的干涉条纹的情形。基于上述条件，在红外光入射到外延层后，会在外延层表面和衬底界面产生两束反射光，两束反射光因存在光程差而发生干涉形成干涉条纹。外延层厚度不同导致光程差不同，所以干涉条纹位置与外延层厚度直接相关。通过分析光程差与波长的关系，建立外延层厚度与干涉条纹特征的数学模型，从而推导出计算外延层厚度的公式。在此基础上，考虑折射率随外延层载流子浓度等因素变化的实际情况，将厚度、载流子浓度与散射率耦合，进一步优化模型。

问题 2：此问题要求我们基于问题一建立的模型，设计确定外延层厚度的算法，并对附件 1、附件 2 中块碳化硅晶圆片的光谱实测数据进行处理，计算出外延层的厚度。附件中包含波数和反射率，因此可直接在波数域展开分析。首先，我们需要对光谱实测数据进行数据预处理，消除测量过程中可能出现的随机误差。接着，我们通过峰值检测从处理后的光谱中提取干涉极值点，作为后续优化的约束条件。然后根据光谱中极值点的顺序及相对位置，结合干涉条件确定相应的干涉级次，基于干涉级数残差平方和定义目标函数，使理论计算结果与实验干涉条件的偏差最小化。最后通过差分进化算法进行全局优化，搜索最优解，得到厚度、载流子浓度和散射率的反演结果，计算出外延层厚度。最后，还需对计算结果的可靠性进行评估，比较附件 1 和附件 2 中不同入射角下的计算结果是否一致，并分析噪声和折射率不确定性对厚度的影响，以此验证建立的模型的稳定性和精确性。

问题 3：此问题要求我们探讨当光波在外延层界面和衬底界面产生多次反射和透射时，产生多光束干涉的必要条件，并分析这一现象对厚度精度测量的影响。如果此问题中仍采用问题一、二中所建立的外延层厚度计算模型，则会产生误差，因此需根据多光束干涉这一现象进一步确定硅外延层厚度计算的数学模型和算法，提高测量精度。因此，使用全谱拟合模型，利用传输矩阵法对整个光谱进行拟合，充分利用实验光谱中的全部信息。通过构造全谱残差平方和目标函数，结合数值优化算法实现厚度、外延层参数的拟合，并对模型进行可靠性分析。

三、模型假设

假设一：假设空气层、外延层以及衬底三层之间层间平行，外延层厚度均匀，且外延层和衬底在光学上为各向同性介质，其折射率与传播方向无关。

假设二：假设反射率与相位反转由菲涅耳公式确定，界面为理想介质界面，且材料在测量波段的吸收系数较小，不会强烈吸收光，表面散射可忽略。

假设三：假设问题一、问题二不考虑半波损失，问题三考虑半波损失。

假设四：假设入射光相干长度大于往返光程差，即可以形成稳定干涉。

假设五：用于拟合的参数在合理范围内且不会出现强耦合导致不可辨识。

四、符号说明

符号	含义	单位
d	外延层的物理厚度	μm
θ_0	光在空气中的入射角	°
θ_1	光在外延层中的折射角	°
n_1	空气的折射率	1
n_2	外延层的折射率	1
ν	波数	cm^{-1}
n_3	衬底的折射率	1
λ	光在真空中的波长	μm
Δ	光程差	μm
δ	相位差	rad
I	反射光的总强度	1
$m(\nu)$	干涉级数	1
γ	载流子散射率	s^{-1}
N	载流子浓度	cm^{-3}
$\epsilon(\nu)$	复数介电函数	1
$k(\nu)$	消光系数	1
B_j	Sellmeier 系数	1
C_j	Sellmeier 系数	1
c	光速	m/s
ω	光的角频率	rad/s
ω_p	等离子体频率	rad/s
e	元电荷	C
ϵ_0	真空介电常数	F/m
m_{eff}	载流子有效质量	kg
R	反射率	1
N_p	找到的极大值点数量	1
N_v	找到的极小值点数量	1
r	残差向量	1
J	目标函数或雅可比矩阵	1
r_i	第i个数据点或极值点的残差	1
$Cov(p)$	参数估计值p的协方差矩阵	（见参数单位）

σ^2	残差的方差	1
σ_d	厚度d的标准不确定度	μm
CV_d	厚度d变异系数	%
w_i	权重因子	1
$\tilde{n}(\nu)$	复数折射率	1
$\epsilon_1(\nu)$	介电函数实部，常规意义折射率	1
$\epsilon_2(\nu)$	介电函数虚部，消光系数	1
$\epsilon_{\text{bound}}(\nu)$	束缚电子跃迁引起的色散效应	1
$\epsilon_{\text{Drude}}(\nu)$	自由载流子的等离子体响应	1
cD_j	细节系数	（取决于具体定义）
$\mathrm{R_{t\acute{\imath}\ eory}(\nu)}$	理论反射光谱	1
$\mathrm{R_{exp}(\nu)}$	实验测量光谱	1
$\mathrm{m(\nu_{peak})}$	反射光谱中的极大值点	1
$\mathrm{m(\nu_{valley})}$	反射光谱中的极小值点	1
$\epsilon_{\text{bound}}(\lambda)$	电子在不同频率下对光场的响应	1
$\epsilon_{\text{Drude}}(\omega)$	载流子与晶格缺陷、声子等发生碰撞的平均频率	1
σ	噪声标准差	1
$\overrightarrow{v_i}$	变异向量	1
$(\overrightarrow{x_{\text{best}}})$	最优个体	（见参数单位）
RMSE_m	干涉级数均方根误差	1
p_j	参数	（见参数单位）
$\mathrm{\eta_j}$	光学导纳	1
$R_{\text{theory}}(\nu)$	理论反射率	1
$R_{\text{exp}}(\nu)$	实验曲线	1
$\mathrm{e_i}$	理论模型与原始数据的残差	1
$\nu_{p,j}$	第j个干涉极大值点对应的波数	cm^{-1}
$\nu_{v,k}$	第k个干涉极小值点对应的波数	cm^{-1}

五、模型的建立与求解

5.1 问题一的模型建立与求解

5.1.1 几何光学分析

本研究采用红外干涉法来测量碳化硅的外延层厚度。该方法利用光波在多层介质中

发生反射和干涉的现象，通过分析反射光的干涉条纹来测量材料的厚度[3]。当一束光照射到外延层表面时，光束会在空气-外延层界面与外延层-衬底界面发生反射，返回的两束光由于传播路径存在差异，形成光程差。这两束光的来源相同，满足频率相同、振动方向相同和相位差恒定的相干条件，会发生干涉现象。干涉后的总反射光强随光的波数变化而产生周期性振荡，形成干涉光谱。本模型的核心任务就是通过精确分析这个干涉光谱的特征[4]，反演出外延层的厚度等物理参数。

为简化问题复杂度，我们假设样品的两层界面均光滑且厚度均匀，只考虑两束主要反射，一束入射光在空气-外延层界面直接反射，另外一束为入射光折射进入外延层，到达外延层-衬底界面后反射，再次折射回到空气中。根据上述条件，我们假设以下物理量：外延层的物理厚度d、光在空气中的入射角θ_0、光在外延层中的折射角θ_1、空气的折射率n_1、外延层的折射率n_2、光在真空中的波长λ。

当光从一种介质入射到另外一种介质时，其传播方向会因折射而发生改变，入射角与折射角之间满足斯涅尔定律，公式如下：

$$n_1 \sin\theta_0 = n_2 \sin\theta_1$$

在本研究中，入射介质为空气，所以$n_1 \approx 1$，这里我们取$n_1 = 1$，于是可以得到折射角的表达式：

$$\sin\theta_1 = \frac{\sin\theta_0}{n_2}$$

该公式也是后续几何光程差推导以及外延层厚度计算的关键。

5.1.2 光程差推导

光程差（optical path difference, OPD）是干涉中两束光在传播过程中实际经过的光学路径长度之差，是产生相位差的根本原因，也是一切后续计算的基础[16]。

根据外延层厚度测量示意图，我们可以发现光束 2 透过外延层并在外延层-衬底界面反射后返回表面，此光束在厚度为d的外延层走过的路径为：

$$2 \times \frac{d}{\cos\theta_1}$$

与此同时，光束 1 在空气中前进了一段距离，这段距离等效于光束 2 在顶层界面横向移动的投影。为了进行相位比较，我们需要从光束 2 的出射点作一条垂线到光束 1 的路径上。通过几何关系可以证明，光束 2 的总光程需要减去这段等效的空气光程。通过严谨的几何推导，可以得到一个简洁而精确的公式。光束 2 在介质中传播的总光程为几何路径的n_2倍。两束光的光程差Δ为：

$$\Delta = 2n_2 \mathrm{d}\, cos\,\theta_1$$

由于折射角θ_1无法直接测量，我们利用三角恒等式和上文的斯涅尔定律，可将其用

入射角θ_0和折射率n_2来表示：

$$\cos\theta_1 = \sqrt{1-\sin^2\theta_1} = \sqrt{1-\left(\frac{sin\,\theta_0}{n_2}\right)^2} = \frac{\sqrt{{n_2}^2-\sin^2\theta_0}}{n_2}$$

将上式代入光程差公式，得到最终可用的表达式：

$$\Delta = 2n_2\mathrm{d}\frac{\sqrt{{n_2}^2-\sin^2\theta_0}}{n_2} = 2d\sqrt{{n_2}^2-\sin^2\theta_0}$$

5.1.3 干涉模型建立

当两束相干光产生干涉时，干涉的效果取决于它们的相位差δ，相位差由光程差Δ决定，两者关系可用下方关系式体现：

$$\delta = \frac{2\pi}{\lambda}\Delta = 2\pi\nu\Delta$$

如果两束光的相位差为2π的整数倍时（$\delta = 2k\pi$，k为整数），两束光同向叠加，反射光强度取极大值，这种干涉为**相长干涉**，此时要求光程差为波长的整数倍：

$$\Delta = k\lambda \Rightarrow \frac{\Delta}{\lambda} = k$$

当两束光的相位差为2π的半整数倍时（$\delta = (2k+1)\pi$，k为整数)，两束光反向叠加，反射光强度取极小值，这种干涉为相消干涉，此时要求光程差为波长的半奇数倍：

$$\Delta = (k+0.5)\lambda \Rightarrow \frac{\Delta}{\lambda} = k+0.5$$

注：此处的干涉条件忽略了反射过程中可能发生的固定相位移动（菲涅尔相移）。在实际薄膜模型中，这些相移是存在的，但对于一个给定的材料体系，其效应是系统性的，可以被包含在更广义的干涉模型中。当前模型将极值点直接与（半）整数条件关联，是一种有效且广泛使用的简化。

为了更直观地处理干涉条件，我们定义一个无量纲的物理量——干涉级数$m(\nu)$，它表示在某个波数λ下，光程差Δ等效于多少个真空波长：

$$m(\nu) = \frac{\Delta}{\lambda} = \nu\Delta$$

将光程差的最终表达式代入，得到干涉级数的核心方程：

$$m(\nu) = 2d\nu\sqrt{n_2(\nu)^2-\sin^2\theta_0}$$

反射光谱中的极值点必须满足极大值点$\mathrm{m}(\nu_{peak}) = \mathrm{k}$（k 为整数），极小值点$\mathrm{m}(\nu_{valley}) = \mathrm{k}+0.5$（k 为整数）。

5.1.4 外延层折射率为常数时厚度反演模型

如果暂时先不考虑题目中所提到的“外延层与衬底因掺杂载流子浓度的不同而有不同的折射率”，设折射率n_2是一个常数，不考虑外延层折射率n_2随波数ν而变化，那么干涉级数方程将大大简化，我们可以利用这一简化模型直接推导出厚度 d，即通过测量相邻极值点的波数将其直接解出。

设相邻的两个干涉极大值点，其对应的波数分别为ν_k和ν_{k+1}，干涉级数分别为k和$k+1$，根据干涉级数方程可得：

$$k = 2d\nu_k\sqrt{{n_2}^2 - \sin^2\theta_0}(1)$$
$$k+1 = 2d\nu_{k+1}\sqrt{{n_2}^2 - \sin^2\theta_0}(2)$$

这是一个关于未知数k和d的线性方程组。用式(2)减去式(1)，可以消去未知的干涉级数k：

$$(k+1) - k = 2d\sqrt{{n_2}^2 - \sin^2\theta_0}(\nu_{k+1} - \nu_k)$$
$$1 = 2d(\nu_{k+1} - \nu_k)\sqrt{{n_2}^2 - \sin^2\theta_0}$$

令$\Delta\nu = \nu_{k+1} - \nu_k$为相邻极大值点之间的波数间隔，我们可以解出厚度$d$：

$$d = \frac{1}{2\Delta\nu\sqrt{{n_2}^2 - \sin^2\theta_0}}$$

同理，对于两个相邻的极小值点，它们的干涉级数分别为$(k+0.5)$和$((k+1)+0.5)$，相减后差值同样为 1，因此上式同样适用。

这个公式清晰地表明，在折射率为常数的假设下，仅需测量干涉光谱[14]中相邻极值点的波数差，即可直接计算出外延层厚度。然而，这个模型的精度完全依赖于n_2的恒定性。对于会表现出显著色散效应的真实半导体材料，n_2随波数变化，导致$\Delta\nu$并非一个常数，因此该简化公式会导致系统误差。因此在下一节更复杂的模型中，需要引入色散修正以提高计算精度。

5.1.5 外延层折射率不为常数时厚度反演模型

在上一节的模型建立与推导中，算出了当外延层折射率为常数时厚度的解析解。此解之所以可行，是因为参数d可以从方程中被代数分离出来。然而，题目说到“外延层与衬底因掺杂载流子浓度的不同而有不同的折射率”，当折射率n_2成为波数函数$n_2(\nu)$后，此方法将失效。此时干涉级数方程可写为：

$$m(\nu) = 2d\nu\sqrt{n_2(\nu)^2 - \sin^2\theta_0}$$

两个相邻极大值点ν_k和ν_{k+1}的方程如下：

$$k = 2d\nu_k\sqrt{n_2(\nu_k)^2 - \sin^2\theta_0}$$

$$k+1 = 2d\nu_{k+1}\sqrt{n_2(\nu_{k+1})^2 - \sin^2\theta_0}$$

两式相减得到：

$$1 = 2d\left(\nu_{k+1}\sqrt{n_2(\nu_{k+1})^2 - \sin^2\theta_0} - \nu_k\sqrt{n_2(\nu_k)^2 - \sin^2\theta_0}\right)$$

此时我们不难发现，厚度d虽然被包含在表达式中，但折射率$n_2(\nu_{k+1})$本身依赖于其他未知的材料参数，例如载流子浓度N等。这意味着厚度、载流子浓度及散射率等多个未知数在方程中高度耦合，无法通过简单的代数消元将厚度d分离出来得到一个解析解。也就是说，每一个极值点的位置都同时受到这些位置参数的共同影响，所以传统的解析法在此情形下失效。

既然无法直接求解，我们必须转变思路。问题不再是"如何根据极值点解出厚度d是多少？"，而是"**是否存在一组参数，能够尽可能完美地解释实验观察到的所有极值点的位置？**"这一思想的转变，要求我们从寻求解析解转向寻求最优解。在这一框架下，为了提高建模的精度，就必须在数值优化框架下进一步引入更为合理的折射率模型，对材料的色散效应和掺杂效应进行综合刻画。

Step1 **复数介电函数与复数折射率**

为了精确描述外延层的光学特性，必须考虑其折射率n_2随波数ν的变化，以及由掺杂引入的自由载流子的影响，我们通过使用一个综合的复数介电函数$\epsilon(\nu)$模型来描述其特性。

材料的光学响应由其复数介电函数$\epsilon(\nu) = \epsilon_1(\nu) + i\epsilon_2(\nu)$或等价的复数折射率$\tilde{n}(\nu) = n(\nu) + ik(\nu)$来完整描述。

复数介电函数可以较好地描述材料在红外光作用下的响应，其中ν为波数，$\epsilon_1(\nu)$为实部，与材料的折射率[2]大小直接相关，$\epsilon_2(\nu)$为虚部，表示光在传播过程中被吸收的部分。复数折射率公式是描述光在实际材料中传播时行为的核心参数，实部$n(\nu)$是常规意义的折射率，决定光的相速度，虚部$k(\nu)$是消光系数，描述光在材料中传播时的能量吸收。两者之间的关系如下：

$$\epsilon(\nu) = [\tilde{n}(\nu)]^2$$

由此可以得到介电函数的实部与虚部：

$$\epsilon_1(\nu) = n(\nu)^2 - k(\nu)^2$$

$$\epsilon_2(\nu) = 2n(\nu)k(\nu)$$

对于掺杂的碳化硅外延层，其介电函数主要由两部分贡献构成：

$$\epsilon(\nu) = \epsilon_{\text{bound}}(\nu) + \epsilon_{\text{Drude}}(\nu)$$

其中$\epsilon_{\text{bound}}(\nu)$用于描述束缚电子跃迁引起的色散效应，可采用 Sellmeier 型模型表示，而$\epsilon_{\text{Drude}}(\nu)$用于描述自由载流子的等离子体响应，可采用 Drude 型模型表示，该组合模型能够同时考虑材料的本征色散特性与掺杂效应。

Step2 **晶格色散**

束缚在晶格原子上的价电子会对入射电磁波产生响应，它主导了材料在透明区域的色散行为。对于碳化硅的外延层，在红外波段内的色散特性通常可以通过 Sellmeier 色散方程[9]进行描述。该模型基于基于经典谐振子理论，可以有效描述价电子在不同频率下对光场的响应。数学表达式为：

$$\epsilon_{\text{bound}}(\lambda) = n_0^2(\lambda) = 1 + \sum_j \frac{B_j \lambda^2}{\lambda^2 - C_j}$$

该表达式中λ=$1/\nu$为真空波长，B_j和C_j统称为 Sellmeier 系数。对于不同类型的碳化硅，它的 Sellmeier 系数是通过实验数据精确标定的材料常数，可通过查阅相关文献资料得知，在后续建模中会作为已知参数引入。此模型能够反映材料在透明区域的折射率随波长的变化规律，为厚度反演与光谱拟合提供了必要的色散基础。

Step3 自由载流子色散

题目中提到“外延层与衬底因掺杂载流子浓度的不同而有不同的折射率”，外延层中的自由载流子在入射电磁波的驱动下会发生集体振荡，尤其是在中远红外波段，对材料的光学响应产生显著影响。该影响通常采用 Drude 模型描述：

$$\epsilon_{\text{Drude}}(\omega) = -\frac{{\omega_p}^2}{\omega^2 + i\gamma\omega}$$

其中ω是光的角频率，c 是光速，$\omega = 2\pi c\nu$。γ是载流子散射率，其物理意义表示为载流子与晶格缺陷、声子等发生碰撞的平均频率，是表征材料电学质量的关键参数，是待求解的未知参数之一。ω_p表示等离子体频率，反映自由载流子集体振荡的固有频率，其平方${\omega_p}^2$正比于载流子浓度N：

$$\omega_p^2 = \frac{Ne^2}{\epsilon_0 m_{\text{eff}}}$$

该式子中载流子浓度 N 是另一个待求解的关键未知参数。$(\mathrm{e}, \epsilon_0, \mathrm{m_{eff}})$ 分别是元电荷、真空介电常数和载流子有效质量，均为已知物理常数。

Step4 Drude-Sellmeier 综合模型

为了同时考虑晶格价电子的束缚效应与自由载流子的等离子体效应，将 Sellmeier 和 Drude 模型结合，可以在统一的理论框架之下同时考虑价电子的晶格色散贡献与自由载流子的等离子体效应，得到总的复数介电函数：

$$\epsilon(\nu; N, \gamma) = \epsilon_{\text{bound}}(\nu) + \epsilon_{\text{Drude}}(\nu; N, \gamma)$$

然后通过开平方根计算得到复数折射率：

$$\tilde{n}(\nu; N, \gamma) = n(\nu; N, \gamma) + ik(\nu; N, \gamma) = \sqrt{\epsilon_{\text{bound}}(\nu) + \epsilon_{\text{Drude}}(\nu; N, \gamma)}$$

此公式明确地建立了外延层折射率的实部$n(\nu; N, \gamma)$与虚部$k(\nu; N, \gamma)$之间的关系，即

模型中的折射率n_2与波数ν及两个核心未知材料参数N和γ之间的复杂函数关系。

由此可见，厚度反演问题不再仅仅是简单的几何光学条件应用，而必须在这一综合模型的框架下，通过数值优化的方法同时反演厚度d、载流子浓度N和散射率γ，才能得到符合实际的可靠结果。

Step5 最终厚度求解模型

基于步骤 4 中得出的依赖于波数和材料参数的折射率$n(\nu;N,\gamma)$，将它带入在上述论文中的干涉级数公式，我们得到了描述整个物理过程的最终数学模型：

$$m(\nu;d,N,\gamma)=2d\nu\sqrt{n(\nu;N,\gamma)^2-\sin^2\theta_0}$$

该模型的核心思想是：对于任何一个在实验光谱中观测到的极值点，无论是极大值$\nu_{p,j}$还是极小值$\nu_{v,k}$，其对应的干涉级数m都必须非常接近一个整数或半整数。然而，由于折射率 $\mathrm{n}(\nu;\mathrm{N},\gamma)$的函数形式非常复杂，且参数$(\mathrm{d},\mathrm{N},\gamma)$与每个极值点对应的整数级数都是未知数，该方程组无法被解析求解以得到一个类似 $\mathrm{d}=\mathrm{f}(\nu_{\mathrm{extrema}})$的直接表达式。因此，确定外延层厚度的数学模型是一个隐式模型，由所有极值点上的干涉条件共同约束。它的求解不能依靠直接的公式代入，需要依赖于数值优化方法，通过在参数空间中迭代搜索最优解来实现。具体方法将在下一节中详细叙述。

综上所述，问题一的数学模型可根据对折射率n_2的不同假设，表述为以下两种形式：

1. 简化模型（折射率为常数）:在此假设下，外延层厚度 d可由相邻干涉极值点的波数间隔 $\Delta\nu$直接解析求解：$d=\frac{1}{2\Delta\nu\sqrt{n_2^2-\sin^2\theta_0}}$，该模型形式简单，但忽略了材料的色散效应，会引入系统误差。

2. 综合物理模型（考虑色散）:此模型更符合物理实际，它将外延层的折射率n_2建模为依赖于波数 ν、载流子浓度 N和散射率 γ的函数 $n(\nu;N,\gamma)$。最终模型是一个由所有观测到的极值点 ν_{extrema} 共同构成的隐式方程组：

$$m(\nu_{\mathrm{extrema}};d,N,\gamma)=2d\nu_{\mathrm{extrema}}\sqrt{n(\nu_{\mathrm{extrema}};N,\gamma)^2-\sin^2\theta_0}\approx k_i\left(\text{整数或半整数}\right)$$

该模型的求解需要依赖于数值优化方法，通过寻找最优参数组合 $\{\mathrm{d},\mathrm{N},\gamma\}$ 来最小化理论干涉级数与（半）整数值的偏差。

5.2 问题二的模型进建立与求解

5.2.1 全局优化算法总体框架与求解思路

基于问题 1 中建立的数学模型，我们设计了一种基于全局优化的算法来同时求解外延层厚度d以及关键材料参数：载流子浓度N、散射率γ。此算法不直接求解方程组，而是将d、N、γ视为未知变量，通过优化算法寻找一组最佳参数$\{d,N,\gamma\}$，使得根据模型计

算出的干涉级数在所有实验极值点上最接近其理论上的整数或半整数。其核心思路是将实验光谱的极值点作为物理约束条件，并通过全局优化方法在参数空间中搜索最优解。

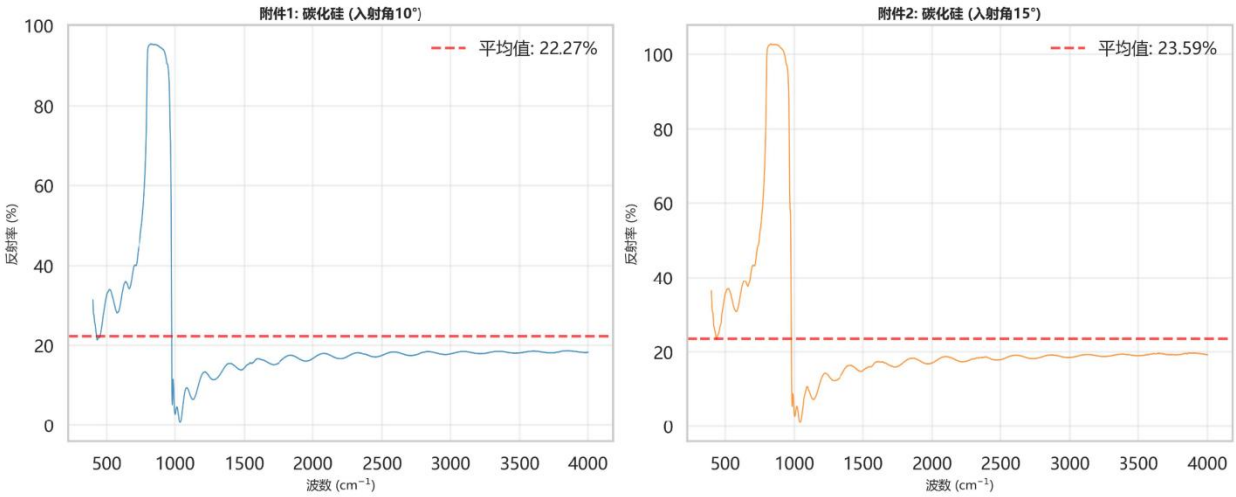
首先，将原始光谱数据波数序列与反射率序列、入射角及碳化硅材料的 Sellmeier 系数作为输入。接着，利用小波变换对光谱进行去噪以提高信噪比，并采用峰值查找算法提取干涉极大值与极小值。在此基础上，构建以干涉级数残差平方和为核心的目标函数，由于目标函数具有非凸性并可能存在多个局部极小值，本研究选用差分进化算法[7]作为优化策略。

在优化过程中需为各参数设定合理的搜索边界，以保证结果具有物理可解释性。最终，算法输出最优参数的估计值及其不确定性区间，实现了从原始光谱数据中准确反演物理参数的目标。

接下来，将一步步具体分析解题步骤：

Step1 数据预处理

在算法实施之前，需要对实验获得的光谱数据进行预处理，以减少噪声干扰并增强干涉条纹的可识别性。题目中给出的附件 1 和附件 2 是入射角分别为 10° 和 15° 时针对同一块碳化硅晶圆片的光谱实测数据。附件中的原始数据包括波数和反射率。



将这些数据可视化，我们可以看出，在波数约 900cm^{-1} 处出现强烈的反射峰，其反射率接近甚至略高于 100%。经查阅资料发现，此强烈的反射峰是碳化硅的特征声子吸收峰，这一现象的本质源于碳化硅材料本身的本征物理属性：碳化硅晶体属于极性半导体，其极性晶格存在红外活性光学声子振动模，当入射红外光子的能量与该声子振动能量共振时，会激发强烈的晶格振动，而光栅结构则通过调控光与声子的耦合作用，使在 700-1000 波段形成剩余射线带，进而呈现出极高的反射率表现。

在波数 1000cm^{-1} 之后，光谱中出现了交替分布的干涉极大值与极小值交替出现的现象，此段曲线较为平滑，这是非常典型的法布里-珀罗干涉条纹图样。干涉条纹的成因是在透明区域，外延层-衬底结构形成了法布里-珀罗谐振腔。

实验噪声和材料本征响应的叠加可能对干涉条纹的准确识别造成影响，为此，为了有效分离碳化硅材料特征与待分析的干涉条纹信号，本研究在预处理阶段引入了小波变换去噪[5]。小波变换能够将信号分解到不同的频率上，使得主要集中在高频、小尺度分量中的噪声得以有效识别和削弱，而干涉条纹等有用信息则主要保存在中低频分量中。同时，由于小波基具有良好的时频局部化特性，去噪过程能够在消除噪声的同时保留光谱中的局部特征，避免了传统滤波方法可能带来的过度平滑效应。此外，通过对高频小波系数进行阈值化处理，小波去噪能够有效去除噪声，而不会显著改变干涉条纹的极值位置和振幅，在提升信噪比的同时保持光谱曲线的物理真实性。

小波去噪的具体原理如下：

1. 分解:我们选用 Daubechies 4（db4）小波基函数。db4 是一种紧支撑、正交的小波，具有良好的正则性，适合捕捉光谱中的干涉条纹等局部特征。对原始反射率信号 $R(\nu)$ 进行多层离散小波变换 DWT 时，为了处理信号边界数据，采用对称延拓模式，这种模式通过镜像反射来延拓信号，可以有效减少边界效应引入的失真。此过程将信号分解为一系列近似系数 cA_L（代表信号的低频主要轮廓）和不同层级的细节系数 $\{cD_1, cD_2, ..., cD_L\}$（代表信号的高频细节和噪声）。

2. 阈值处理: 对各层级的细节系数cD_j 应用阈值函数。噪声通常表现为幅值较小的细节系数，而真实的信号特征（如干涉条纹的峰谷）则对应较大的系数。本研究采用软阈值法 (Soft Thresholding)，其定义为：

$$\widehat{cD_j}(k) = \begin{cases} sgn\big(\mathrm{cD_j(k)}\big)\big(\big|\mathrm{cD_j(k)}\big| - \lambda\big), & \big|\mathrm{cD_j(k)}\big| > \lambda \\ 0, & \big|\mathrm{cD_j(k)}\big| \le \lambda \end{cases}$$

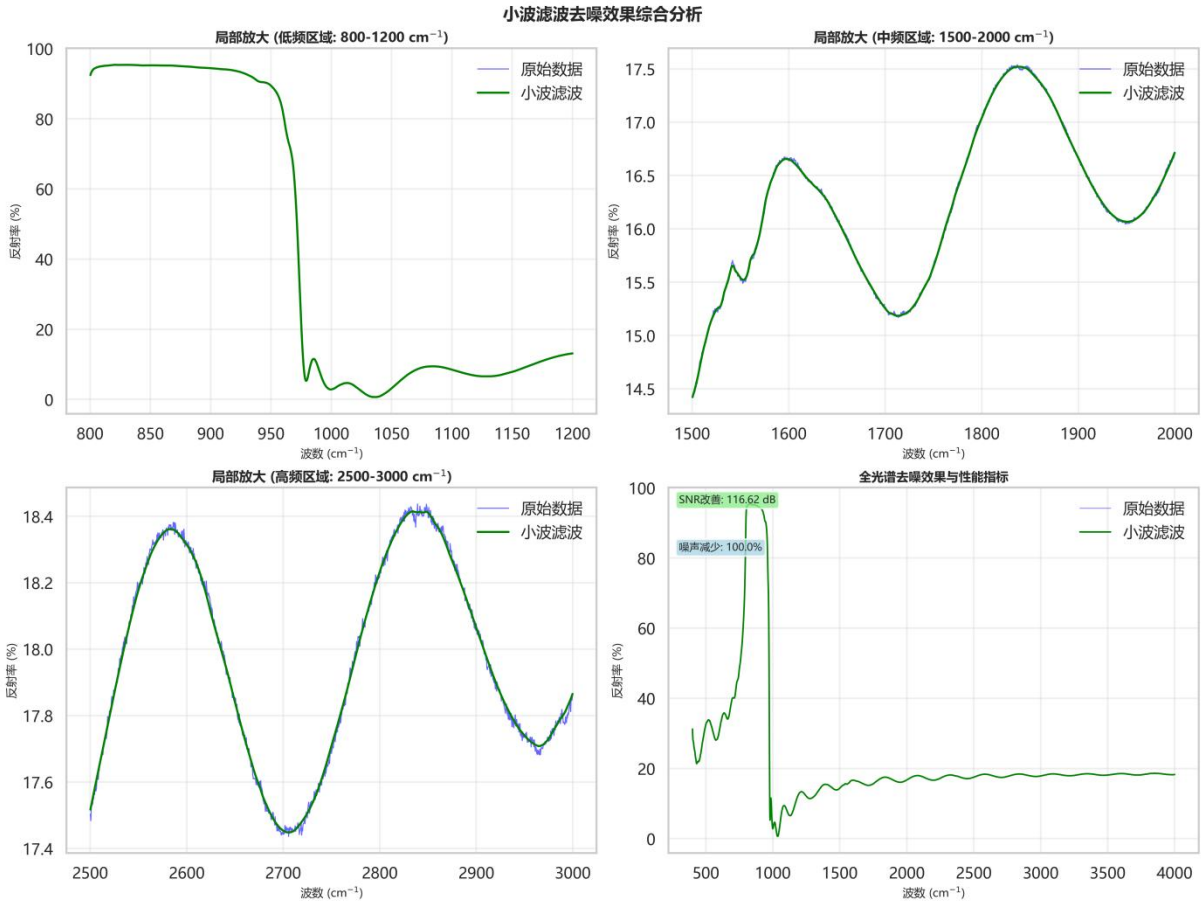
其中 λ 是预设的阈值。该方法将低于阈值的系数置零，并将高于阈值的系数向零收缩，相比于直接置零的硬阈值法，重构的信号更为平滑。阈值 λ 通常采用通用阈值准则进行估计：

$$\lambda = \sigma\sqrt{2\log N}$$

其中 N 是信号长度，噪声标准差 σ 通过第一层细节系数的绝对中位差(Median Absolute Deviation, MAD)稳健地估计得出。

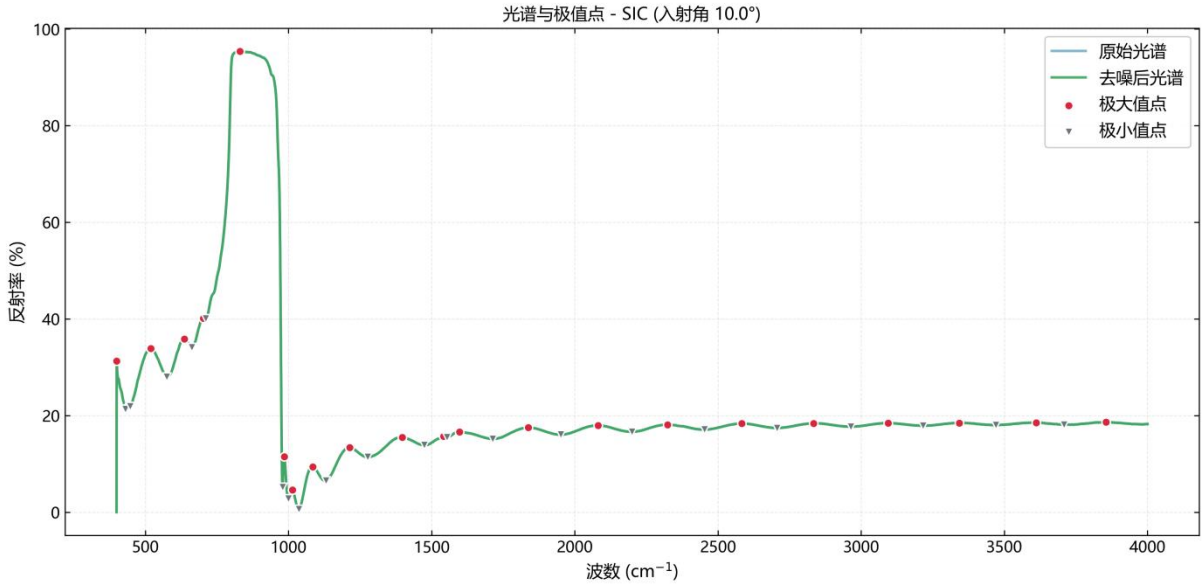
3. 重构：使用未经改变的近似系数cA_L 和经过阈值处理后的新细节系数 $\{\widehat{cD_j}\}$，通过逆离散小波变换 (IDWT) 重构出最终的去噪信号。

通过此过程，噪声得以有效滤除，同时保留了干涉条纹的位置和基本形态，为后续精确提取极值点奠定了基础。



以附件 1 为例，上面四幅图可以看出，小波去噪只对高频噪声分量进行阈值削弱，而对主要的中低频分量保留原样。它通过保留主要成分、仅抑制高频噪声，实现了在保持曲线整体形状和干涉条纹特征的同时，提高了数据质量。

以附件 1 为例，采用峰值查找算法精确提取干涉极大值与极小值的位置，形成用于优化的实验约束数据集：



Step2 目标函数定义

我们通过构建一个目标函数来量化理论模型与实验数据的匹配程度，该函数基于法布里-珀罗干涉原理，通过计算所有极值点上干涉级数残差的平方和，形成一个典型的最小二乘优化问题。

为了衡量参数$\{d, N, \gamma\}$的好坏程度，我们需要量化一个指标，因此我们需要引入目标函数，来量化理论模型与实验观测之间的总误差。此问题中的物理约束是：在极值点波数处干涉级数必须是半整数或者整数。因此，一个合理的参数组合应当使得计算得到的干涉级数与最近的半整数或者整数之差为零。因此一个合理的目标函数可以定义为所有极值点上的这个差值的平方和作为总误差：

$$J(d,N,\gamma)=\sum_{j=1}^{N_p}\left[m(\nu_{p,j};d,N,\gamma)-\mathrm{round}(m(\nu_{p,j}))\right]^2 + \sum_{k=1}^{N_v}\left[m(\nu_{v,k};d,N,\gamma)-\left(\mathrm{round}(m(\nu_{v,k})-0.5)+0.5\right)\right]^2$$

其中，$m(\nu;d,N,\gamma)$由问题 1 的最终模型给出，目标函数值$J(d,N,\gamma)$的大小直接反映了参数组合与物理干涉条件的符合程度，J值越小，表明该参数组合下理论计算的干涉级数越接近整数(对应极大值点)或半整数(对应极小值点)，即越符合法布里-珀罗干涉的物理条件，表示这组参数对实验数据的拟合度越高。

通过最小化目标函数 J，我们能够找到一组最优的参数估计，使理论模型最好地解释实验观测到的干涉现象。

Step3 差分进化优化算法实现

由以上分析，不难发现问题被转化成了在参数空间中寻找能使目标函数$J(d,N,\gamma)$达

到最小值的点，变成了一个数值优化问题：

$$\{d, N, \gamma\}_{\text{opt}} = arg \min_{d,N,\gamma} J(d, N, \gamma)$$

目标函数是一个复杂的非线性多峰函数，其参数空间中可能存在多个局部极小值。为可靠地获得全局最优解，我们不再进行代数推导，而是使用差分进化算法来系统地搜索参数空间，直到找到最优解。在每一代迭代中，算法通过父代个体之间的差分变异、与当前解的交叉操作，生成新的候选解，并通过适者生存原则保留表现更优的解。通过这种方式，算法能够有效探索和开发整个参数空间。

其核心迭代流程包括以下四个步骤：

1.初始化：在预设的参数边界内，随机生成一个包含 NP (Number of Population，即种群大小)个个体（候选解）的初始种群。每个个体都是一个包含待求参数 $\{\mathrm{d}, \mathrm{N}, \gamma\}$ 的向量。在我们的实现中，种群大小 NP 被设定为 20。

2.变异：遍历种群中的每一个个体（称为"目标向量"$\overrightarrow{x_i}$。为该个体生成一个"变异向量"$\overrightarrow{v_i}$。变异操作是选取当前种群中的最优个体$(\overrightarrow{x_{\text{best}}})$ 以及两个从种群中随机选择、且与目标向量不同的个体 $(\overrightarrow{x_{r1}}, \overrightarrow{x_{r2}})$，然后通过差分缩放生成变异向量：

$$\overrightarrow{v_i} = \overrightarrow{x_{\text{best}}} + F \cdot (\overrightarrow{x_{r1}} - \overrightarrow{x_{r2}})$$

其中 F 是缩放因子，参数设为 $(0.5, 1)$ 区间内的随机值，用于控制差异向量的缩放幅度。

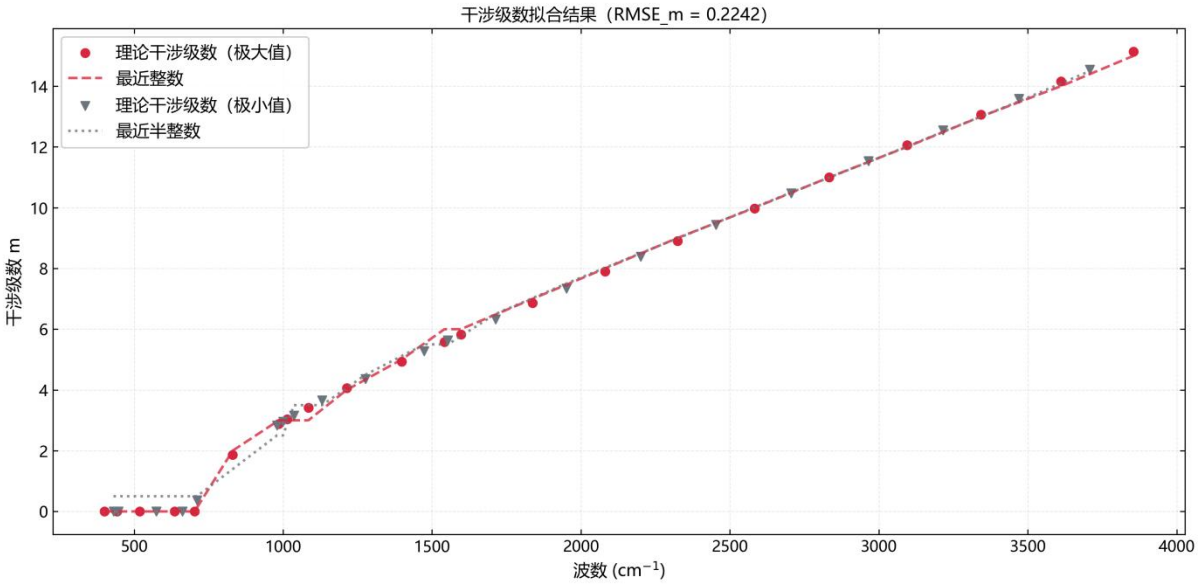
3.交叉：将变异向量 $\overrightarrow{v_i}$ 与目标向量 $\overrightarrow{x_i}$ 的对应参数进行混合，生成一个"试验向量" $\overrightarrow{u_i}$。具体地，对向量的每一维，以预设的交叉概率 CR（设为 0.7）决定是选择变异向量的参数还是目标向量的参数。这种机制保证了至少有一维参数来自变异向量，从而引入新信息。

4.选择：计算试验向量$\overrightarrow{u_i}$ 对应的目标函数值，并将其与目标向量 $\overrightarrow{x_i}$的目标函数值进行比较。如果试验向量的目标函数值更小（即解更优），则在下一代种群中用试验向量替换目标向量；否则，保留原目标向量。

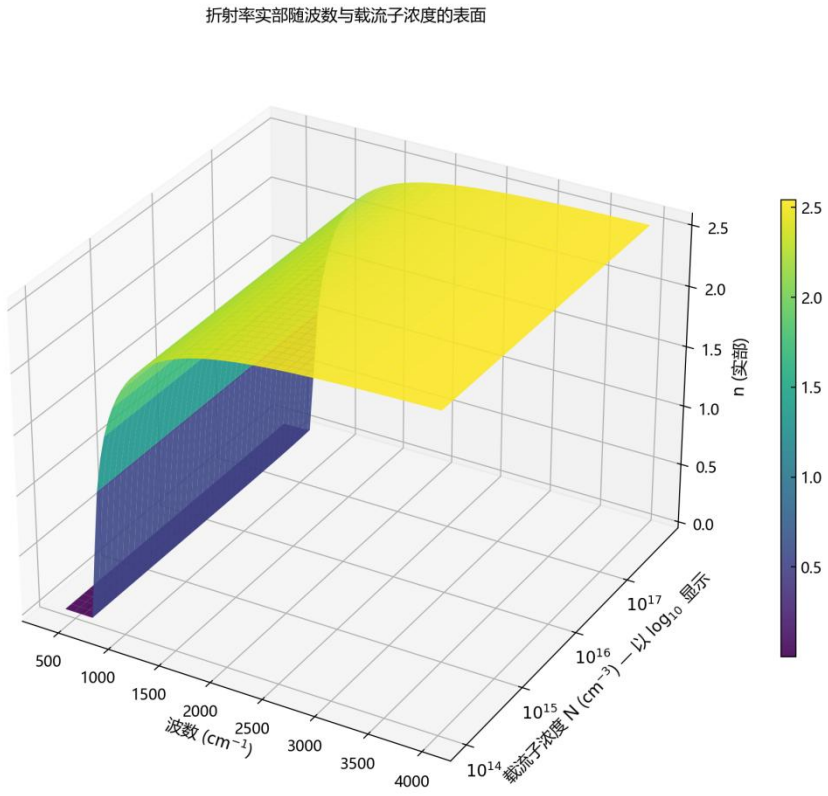
上述"变异-交叉-选择"的过程对种群中所有个体执行一次后，即完成了一代进化。算法将重复此过程，直到达到最大迭代次数或满足其他收敛条件，最终输出当前种群中的最优个体作为问题的解。

Step4 模型计算结果可视化分析

将原始光谱数据中的波数序列ν与反射率序列R，结合入射角θ_0以及碳化硅材料的 Sellmeier 系数作为算法输入。差分进化优化后，理论曲线与实验极值点在干涉级数空间的拟合程度如下图所示：



碳化硅材料的 Sellmeier 系数在 Drude-Sellmeier 综合模型中通过拟合得到折射率n_2与波数ν及核心未知材料参数N之间的复杂函数关系，可视化结果如下图所示：



上述步骤中，我们通过逐步分析，确定了测量外延层厚度的算法。将附件 1 中的数据带入模型，得出结果 d = 7.746 ± 0.036 μm；将附件 2 中的数据带入模型，得出结果 d: 7.646 ± 0.031 μm。最终推荐的厚度为两者加权平均，d =7.6883 ± 0.0237 μm。

5.2.2 结果评估与可靠性分析

为了保证所建立模型与优化算法的有效性，仅给出参数的最优解还不够，还必须对

结果的可信度进行系统评估。下面将从三个层面来分析：通过拟合优度定量刻画模型与实验数据的吻合程度、利用最小二乘理论评估参数的不确定度以检验模型结果的稳健性；通过变异系数对厚度参数的可靠性进行分级判定。

Step1 拟合优度

为了定量评估模型对实验数据的拟合效果，本研究引入干涉级数均方根误差RMSE_m作为指标。我们的建模目标是让实验极值点的干涉级数尽可能接近整数或半整数，而此方法正是对这种偏离程度的量化，直接反映了物理干涉条件是否满足，公式如下：

$$\mathrm{RMSE}_m = \sqrt{\frac{J_{\min}}{N_p + N_v}}$$

其中，$\mathrm{N_p}$和$\mathrm{N_v}$分别表示极大值点与极小值点的数量。$\mathrm{RMSE_m}$值越小，表明模型所预测的干涉级数与实验光谱中提取的极值点吻合程度越高，拟合效果越好。最终代码算出结果，附件 1 的干涉级数$\mathrm{RMSE_m}$为 0.2242，附件 2 的干涉级数$\mathrm{RMSE_m}$为 0.2241，此值表示实验极值点位置与理想模型预测之间的误差大约是不到 1/4 个干涉周期，属于合理的拟合水平。

此外，两者只差 0.001 远小于$\mathrm{RMSE_m}$数值本身，这表明尽管两次测量的厚度结果存在轻微差别，但模型在解释两组光谱的干涉条纹时具有同样的有效性和稳定性，拟合质量可以认为是等效的。由此可见建立的模型稳定性良好，不会因数据集的不同而产生显著差别。

Step2 参数不确定度估计

虽然差分进化算法能给出一组最优参数$\{d, N, \gamma\}$，但这只是一个点估计，为了让结果有统计意义，就要估计这些参数的不确定度，例如标准差、置信区间等。为此，本研究基于最小二乘理论的线性化近似方法，对参数的不确定度进行计算。

整个评估过程的核心逻辑是：先借助残差衡量模型与数据的偏差，再依据残差对参数的敏感程度构建矩阵，最终确定参数的波动范围。

首先，构造残差向量r，将每个极值点的干涉级数残差r_i按顺序排列，形成一个长向量，用于反映理论模型在所有极值点上与实验数据的偏离程度。接着，计算雅可比矩阵J，雅可比矩阵量化了每个残差对每个参数的敏感度，这里用p_j表示参数，即$p_j \in \{d, N, \gamma\}$，构造雅可比矩阵：

$$J_{ij} = \frac{\partial r_i}{\partial p_j}$$

该公式用于量化每个残差r_i对参数p_j的敏感度，J_{ij}越大，意味着参数p_j的微小变化，会引起残差r_i的显著波动。由于模型复杂，我们采用有限差分法进行数值计算，即通过人为微小改变参数值，观察残差的变化率。

然后，求解协方差矩阵。在最小二乘理论框架下，参数的协方差矩阵可近似为：

$$\mathrm{Cov}(p) \approx \sigma^2(\mathbf{J}^T\mathbf{J})^{-1}$$

其中，$(\mathbf{J}^T\mathbf{J})^{-1}$描述了不同参数间的耦合关系，而残差方差$\sigma^2$则用于衡量模型的整体拟合优度，其定义为：

$$\sigma^2 = \frac{\sum r_i^2}{N_{\text{extrema}} - N_{\text{params}}} = \frac{J_{\min}}{N_p + N_v - 3}$$

最后，通过上述步骤可得到各参数的标准不确定度，即协方差矩阵对角元的平方根：

$$\sigma_d = \sqrt{\mathrm{Cov}(p)_{11}},\quad \sigma_N = \sqrt{\mathrm{Cov}(p)_{22}},\quad \sigma_\gamma = \sqrt{\mathrm{Cov}(p)_{33}}$$

这些不确定度值量化了在给定数据和模型下，拟合参数的置信区间大小。附件 1 数据代入后，算得σ_d=7.746 μm ± 0.036， σ_N=5.73e+15 cm-³ ±0.00e+00， σ_γ=9.47e+14 s-1 ±0.00e+00，附件 2 数据代入后，算得σ_d=7.646 μm ± 0.031， σ_N=4.91e+15 cm-3 ± 0.00e+00， σ_γ=7.20e+14 s-1 ± 0.00e+00。

Step3 可靠性判据

根据计算出的厚度d及其不确定度σ_d，可以计算变异系数CV_d来作为可靠性的最终判据。该系数通过将参数的不确定度与估计值进行归一化处理，反映参数估计结果的相对波动程度，公式如下：

$$CV_d = \frac{\sigma_d}{d} \times 100\%$$

依据变异系数的数值大小，将可靠性划分为三个等级：当$\mathrm{CV_d} < 5\%$时，说明模型对该参数的拟合效果稳定且精确；当$5\% \le \mathrm{CV_d} < 10\%$时，表面参数估计存在一定程度的不确定性，模型对数据的拟合基本合理，但仍存在优化空间；当$\mathrm{CV} \ge 10\%$时，此时表明参数估计值的相对波动较大，模型对该参数的拟合效果欠佳。

由代码运行结果可以得知，附件 1 数值带入模型后算得得变异系数CV_d= 0.47%，可靠性高，附件 2 数值带入模型后算得得变异系数CV_d=0.41%,，可靠性高，加权平均之后厚度的变异系数CV_d=0.31%，最终结果的可靠性高。

由此可见，该外延层厚度计算模型在可靠性和实用性上是合理可行的，不仅能够较好地拟合实验光谱的干涉条纹，而且所得厚度结果在不同入射角条件下保持一致性，误差控制在合理范围内。这说明模型能够有效提取外延层的关键物理参数，具有较高的可靠性和应用价值。

5.3 问题三的模型进建立与求解

5.3.1 复数介电函数的关键作产生多光束干涉的必要条件

问题 1 和问题 2 中仅考虑两束主要光束，即假设光在外延层和衬底界面之间仅发生一次反射，忽略了多光束干涉效应。在问题三中，光波可以在外延层界面和衬底界面产生多次反射和透射，从而产生多光束干涉，通过在网上查阅资料以及推导，我们发现，发生多光束干涉有以下几个必要条件：

(1) 光源的相干长度必须至少不小于相邻两束光的光学程差，保证不同反射次数对应的光束具有相干性，可以产生稳定的干涉条纹。若相干条纹不足，则会导致干涉峰谷信息变少，拟合的不确定性增加，从而降低厚度计算的精度。

(2) 外延层上下界面应足够平整、相对平行，且粗糙度要远小于工作波长，否则散射和相位随机化会破坏多次反射的相干叠加，降低条纹对比度，影响厚度计算的精度。

(3) 空气-外延层与外延层-衬底两界面的反射系数不宜过小，且每次往返的衰减不能过大，否则高阶干涉条纹信号减弱，测量到的干涉信息减少。

(4) 材料在测量波段的吸收不能太强，否则多次反射的光在外延层内衰减过快，高阶干涉条纹将无法被观测到。

(5) 需保证光束入射角已知且可控，每次反射的几何光程与相位关系稳定。

5.3.2 复数介电函数的关键作用

问题 1 和问题 2 中提出的基于干涉光谱极值点的拟合方法，是一种有效且计算速度快的外延层厚度测量手段。但是对于多光束干涉，该方法存在一些局限性：如极值点信息利用率低、对噪声敏感、难以精确反演出材料的吸收等参数等。

针对这些缺点，我们需要建立更为精确的模型来计算外延层厚度d——全谱拟合模型。该模型不再依赖于极值点，而是通过一个严格的物理模型，直接对整个实验反射光谱进行拟合。这种方法利用了光谱中的所有信息，因而更加稳健和精确。下面将一步步介绍该模型的建立过程。

全谱拟合模型与极值点拟合模型的核心区别在于复数介电函数[6]$\epsilon(\nu) = \epsilon_1 + i\epsilon_2$的运用方式上[12]。

针对问题 1 与问题 2 中的极值点模型，该模型以干涉的相位条件为核心，其目标函数仅仅计算干涉级数 $m(\nu)$与整数或半整数之间的偏差：

$$m(\nu) = 2d\nu\sqrt{n(\nu)^2 - \sin^2\theta_0})$$

在此公式中，仅使用了外延层折射率的实部 $n(\nu)$，其潜在假设是把衬底可以看成一个理想反射体，者其光学特性对干涉极值点位置的影响可以忽略不计。但在本小问中提到：“光波可以在外延层界面和衬底界面产生多次反射和透射，从而产生多光束干涉”，因此，该模型未对衬底的复数介电函数进行建模。此外，由于该模型完全忽略了介电函数的虚部ϵ_2，而虚部主要决定了光在材料中的能量吸收，并且显著影响干涉条纹的振幅。而极值点模型未能利用这一关键信息，因此在参数提取上存在一定的局限性。

相比之下，全谱拟合方法更适合本题模型的建立。全谱拟合法基于传输矩阵，在建模过程中完整地考虑了外延层与衬底的复数介电函数。传输矩阵法的本质上是复数运算，将折射率的实部 n 和虚部 k 同时纳入计算，实部影响光程差和干涉条纹的位置，虚部影响反射和透射的强度，并最终决定干涉条纹的形态。

综上所述，全谱拟合模型通过完整地使用复数介电函数，将问题 1、问题 2 中有关“相位匹配”的问题，升级为了一个“振幅与相位协同匹配”的物理问题。这一改进使得模型能够从光谱的精细形态中提取更多信息，从而更精确地反演出厚度、载流子浓度和散射率在内的多个物理参数。

5.3.3 从“相位拟合”到“全谱拟合”

全谱拟合方法[13]的核心在于：通过寻找一组最优的物理参数 $\mathrm{d}, \mathrm{N_{epi}}, \gamma_{\mathrm{epi}}, \mathrm{N_{sub}}, \gamma_{\mathrm{sub}}$，使得由这些参数计算得到的理论反射光谱$\mathrm{R}_{theory}(\nu)$与实验测量光谱$\mathrm{R_{exp}}(\nu)$之间的差异达到最小。

为实现这一目标，我们定义了一个新的目标函数，即所有波数点上理论与实验光谱的加权残差平方和：

$$J = \sum_i w_i \left[R_{\mathrm{exp}}(\nu_i) - R_{\mathrm{theory}}\left(\nu_i; \mathrm{d}, N_{\mathrm{epi}}, \gamma_{\mathrm{epi}}, N_{\mathrm{sub}}, \gamma_{\mathrm{sub}}\right)\right]^2$$

其中，w_i是可选的权重。因此，我们的任务就变成了对该目标函数J的最小化问题：

$$\{\mathrm{d}, \mathrm{N_{epi}}, \gamma_{\mathrm{epi}}, \mathrm{N_{sub}}, \gamma_{\mathrm{sub}}\}_{opt} = arg\min J\,(\mathrm{d}, \mathrm{N_{epi}}, \gamma_{\mathrm{epi}}, \mathrm{N_{sub}}, \gamma_{\mathrm{sub}})$$

这个方法的关键点在于准确计算出理论反射光谱$\mathrm{R_{theory}}$，这需要一个能够精确描述光在多层介质中传播、反射和干涉过程的物理模型。

5.3.4 理论反射率计算：传输矩阵法

在全谱拟合中，目标函数的构建依赖于反射光谱$R_{theory}(\nu)$的计算。要得到这一光谱，就必须建立一个能够全面描述红外光在外延层和衬底中传播过程的物理模型。在这种空气-外延层-衬底的三层结构中，其总反射率由各界面的多次反射和光束在外延层内的多次相干叠加决定。传输矩阵法[8]是计算这种多层薄膜系统光学响应的严谨而通用的方法。它完美地包含了多光束干涉效应，并且能够处理材料具有吸收的真实情况。实现传输矩阵法由以下步骤完成：

Step1 材料光学常数模型 Drude-Sellmeier

为了精确计算反射率，需建立描述外延层和衬底材料光学特性的数学模型。在全谱拟合模型中，对两层材料均采用上述文章中提到的 Drude-Sellmeier 模型[9]来描述材料的整体复介电函数，该模型是描述掺杂半导体在红外波段光学响应的黄金标准。

由上述论文我们知道，复数介电函数$\epsilon(\nu)$由两部分构成：

$$\epsilon(\nu) = \epsilon_{\mathrm{bound}}(\nu) + \epsilon_{\mathrm{Drude}}(\nu)$$

其中 Sellmeier 部分表示晶格色散项，用于描述束缚电子的贡献，通过查阅文献资料[10,11]，我们发现对于此函数的实现采用标准的三项 Sellmeier 方程：

$$\epsilon_{\text{bound}}(\lambda) = 1 + \frac{A_1\lambda^2}{\lambda^2 - \lambda_1^2} + \frac{A_2\lambda^2}{\lambda^2 - \lambda_2^2} + \frac{A_3\lambda^2}{\lambda^2 - \lambda_3^2}$$

式中λ为波长，A_{i}和λ_i^2是实验中标定的材料常数，对于碳化硅是已知的。

Drude[12]部分表示自由载流子色散项，用来描述由掺杂引入的自由载流子的贡献。公式表示如下：

$$\epsilon_{\text{Drude}}(\omega) = -\frac{\omega_p^2}{\omega^2 + i\gamma\omega}$$

而等离子体频率ω_p由待求的载流子浓度N决定：

$$\omega_p^2 = \frac{Ne^2}{\epsilon_0 m_{\text{eff}}}$$

电磁波在材料中传播时，满足：

$$\tilde{n}(\nu)^2 = \epsilon(\nu)$$

也就是说，复折射率的平方等于复介电函数，所以直接取平方根：

$$\tilde{n}(\nu) = \sqrt{\epsilon(\nu)}$$

最终，我们得到复数折射率。在我们的模型中，将波数ν代入，外延层的复数折射率$\tilde{n}_1(\nu)$和衬底的复数折射率$\tilde{n}_2(\nu)$分别表示为：

$$\tilde{n}_1(\nu) = \tilde{n}(\nu; N_{\text{epi}}, \gamma_{\text{epi}})$$
$$\tilde{n}_2(\nu) = \tilde{n}(\nu; N_{\text{sub}}, \gamma_{\text{sub}})$$

因此，待求参数组变为$\{\text{d}, \text{N}_{\text{epi}}, \gamma_{\text{epi}}, \text{N}_{\text{sub}}, \gamma_{\text{sub}}\}$，这组参数是输入到传输矩阵法进行理论光谱计算的核心物理量。

Step2 理论光学模型

本光学系统包含空气、外延层、衬底三个区域，空气折射率n_0取值为 1，外延层复折射率为$\tilde{\text{n}}_1(\nu)$，由$\text{N}_{\text{epi}}, \gamma_{\text{epi}}$参数化描述，被视为半无限厚，复折射率为$\tilde{\text{n}}_2(\nu)$，由$\text{N}_{\text{sub}}, \gamma_{\text{sub}}$参数化描述。

Step3 传输矩阵的数学推导

传输矩阵法[8]的理论基础源于麦克斯韦方程组在介质界面处的电磁场连续性条件。其核心思想是，用一个 2×2 矩阵来描述光在单一介质层中的传播以及在介质界面上的透射和反射。

把空气-外延层界面作为参考平面，建立一个 z 轴垂直于样品表面的坐标系。外延层的厚度是 d，所以令其上界面为 $\text{z} = 0$，下界面为 $\text{z} = \text{d}$。在该层内部，切向电场 E(z)

和磁场 H(z)是前进波(+)和后退波(−)的叠加：

$$E(z) = E_1^+ e^{ik_{z1}z} + E_1^- e^{-ik_{z1}z}$$
$$H(z) = \eta_1(E_1^+ e^{ik_{z1}z} - E_1^- e^{-ik_{z1}z})$$

该公式中，k_{z1}是波矢量在 z 方向上的分量，$k_{z1} = \frac{2\pi}{\lambda}\tilde{n}_1 \cos\theta_1$，$\eta_1$是该层的光学导纳，描述该层的电磁特性。通过求解上述方程组，我们可以将 $z = 0$ 处的场$(E(0), H(0))$与 $z = d$ 处的场$(E(d), H(d))$关联起来。经过代数运算，可以得到如下关系：

$$\begin{pmatrix} E(0) \\ H(0) \end{pmatrix} = \begin{pmatrix} \cos(\beta_1) & \frac{i}{\eta_1}\sin(\beta_1) \\ i\eta_1 \sin(\beta_1) & \cos(\beta_1) \end{pmatrix} \begin{pmatrix} E(d) \\ H(d) \end{pmatrix}$$

我们定义描述外延层的特征矩阵M_1为：

$$\mathbf{M}_1 = \begin{pmatrix} \cos(\beta_1) & \frac{i}{\eta_1}\sin(\beta_1) \\ i\eta_1 \sin(\beta_1) & \cos(\beta_1) \end{pmatrix}$$

在该矩阵中，β_1是相位厚度，描述了光束单次穿过该层的复数相位变化：

$$\beta_1 = k_{z1}d = \frac{2\pi}{\lambda}\tilde{n}_1 d\cos\theta_1 = 2\pi\nu d\sqrt{\tilde{n}_1^2 - (n_0\sin\theta_0)^2}$$

光学导纳η_j则由介质的复折射率与入射角决定，当发生 s-偏振，即电场垂直于入射面时，$\eta_j = \tilde{n}_j\cos\theta_j$，当发生 p-偏振，即电场平行于入射面时，$\eta_j = \tilde{n}_j/\cos\theta_j$。实际的近正入射实验条件下，通常选用 s-偏振的近似形式：

$$\eta_j = \sqrt{\tilde{n}_j^2 - (n_0\sin\theta_0)^2}$$

对于多层体系，整个系统的总传输矩阵就是各层特征矩阵的乘积。对于我们的单层系统，总矩阵 $M = M_1$。

Step4 反射率的数学推导

在传输矩阵法框架下，将总传输矩阵将入射介质的场与出射介质的场关联起来：

$$\begin{pmatrix} E_0 \\ H_0 \end{pmatrix} = \mathbf{M}\begin{pmatrix} E_2 \\ H_2 \end{pmatrix}$$

在空气区域，电磁场是入射波 i 和反射波 r 的叠加，可表示为：

$$E_0 = E_i + E_r$$
$$H_0 = \eta_0(E_i - E_r)$$

其中η_0为光学导纳。在衬底区域，我们假设它是半无限厚的，因此只有透射波 t，没有后向反射波，可表示为：

$$E_2 = E_t$$
$$H_2 = \eta_2 E_t$$

将这些关系代入矩阵方程，得：

$$\begin{pmatrix} E_i + E_r \\ \eta_0(E_i - E_r) \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} E_t \\ \eta_2 E_t \end{pmatrix} = \begin{pmatrix} (M_{11} + M_{12}\eta_2)E_t \\ (M_{21} + M_{22}\eta_2)E_t \end{pmatrix}$$

结合上述两个方程两个方程，我们定义整个光学系统中外延层+衬底的等效光学导纳 Y：

$$Y = \frac{H_0}{E_0} = \frac{(M_{21} + M_{22}\eta_2)E_t}{(M_{11} + M_{12}\eta_2)E_t} = \frac{M_{21} + M_{22}\eta_2}{M_{11} + M_{12}\eta_2}$$

在入射介质中，电场与等效导纳的关系为：

$$\eta_0(E_i - E_r) = Y(E_i + E_r)$$

上式两边同除以$\mathrm{E_i}$，并引入振幅反射系数 r，根据 $\mathrm{r = E_r/E_i}$，可得：

$$\eta_0(1 - r) = Y(1 + r)$$

整理后得到：

$$r = \frac{\eta_0 - Y}{\eta_0 + Y}$$

最终，总反射率 R 就是振幅反射系数模的平方：

$$R_{\mathrm{theory}}(\nu) = |r|^2 = \left|\frac{\eta_0 - Y}{\eta_0 + Y}\right|^2$$

该公式可以精确计算在给定参数$\tilde{\mathrm{n}}_1, \tilde{\mathrm{n}}_2$和 d 之下计算理论反射率，并自然地包含了所有多光束干涉效应。

5.3.5 确定硅外延层厚度计算的数学模型与参数优化

根据上述步骤使用得传输矩阵法，以及 Drude-Sellmeier 折射率模型，我们构建了一个完整的物理模型。

首先该模型将输入的 5 个待求参数外延层厚度d、外延层流子浓度N_{epi}和散射率γ_{epi}、衬底的载流子浓度N_{sub}和散射率γ_{sub}；接着使用 Drude-Sellmeier 模型，根据载流子浓度和散射率计算出外延层和衬底在每个波数ν下的复折射率$\tilde{\mathrm{n}}_1(\nu)$和$\tilde{\mathrm{n}}_2(\nu)$；然后将复折射率和厚度 d 代入传输矩阵模型，计算出理论反射光谱$\mathrm{R_{theory}(\nu)}$，并将理论光谱与实验测量光谱$R_{\mathrm{exp}}(\nu)$进行对比，构造目标函数：

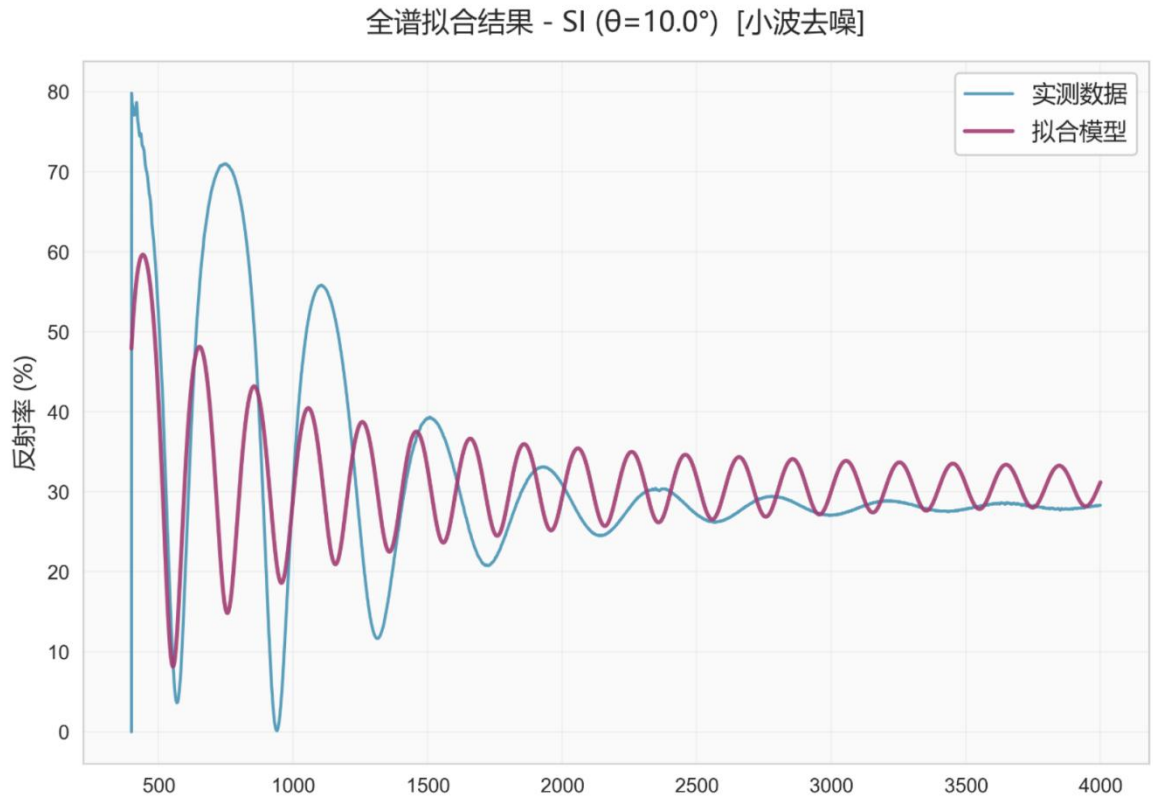
$$J = \sum_i w_i \left[R_{\mathrm{exp}}(\nu_i) - R_{\mathrm{theory}}(\nu_i; \mathrm{d}, N_{\mathrm{epi}}, \gamma_{\mathrm{epi}}, N_{\mathrm{sub}}, \gamma_{\mathrm{sub}})\right]^2$$

最后采用数值优化算法[17]进行参数调整，重复之前的步骤，直到找到使J最小的参数组合。

5.3.6 硅晶圆片多光束干涉模型的计算结果

根据流程步骤及公式，我们可以精确求解厚度。将附件 3、附件 4 中的波数带入，计算对应波数下的理论反射率$\mathrm{R_{theory}(\nu)}$，使得理论曲线不断接近实验曲线$R_{\mathrm{exp}}(\nu)$。

以附件 3 为例，得到的拟合理论反射率曲线和实验曲线对比如下：

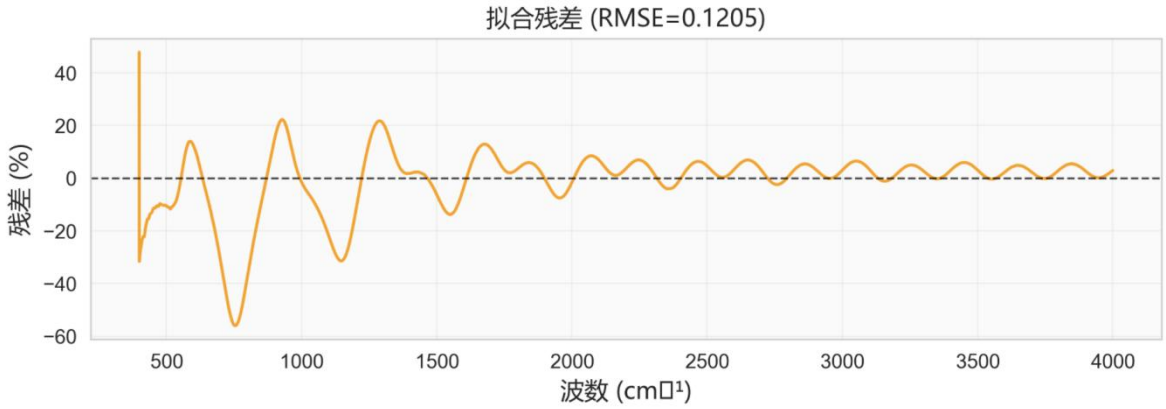


最终，根据附件 3 的数据，我们计算出外延层载流子浓度约为 $9.25 \times 10^{17} cm^{-3}$，散射率为 $1.69 \times 10^{14} \mathrm{s}^{-1}$。衬底的载流子浓度则达到 $2.66 \times 10^{20}\ \mathrm{cm}^{-3}$，散射率为 $4.62 \times 10^{15}\ \mathrm{s}^{-1}$，外延层厚度$d$为 $7.313 \pm 0.056 \mu\mathrm{m}$。根据附件 4 的数据，我们计算出外延层载流子浓度约为 $9.77 \times 10^{17} cm^{-3}$，散射率为 $1.84 \times 10^{14} \mathrm{s}^{-1}$。衬底的载流子浓度则达到 $4.65 \times 10^{20}\ \mathrm{cm}^{-3}$，散射率为 $6.36 \times 10^{15}\ \mathrm{s}^{-1}$，外延层厚度$d$为 $7.272 \pm 0.063 \mu\mathrm{m}$。

对于该模型计算出的结果，我们对本模型进行可靠性与不确定度分析，对于复杂的非线性模型，自助法是一种更稳健的参数不确定度估计方法，基本步骤如下：

首先，使用最优参数计算出理论模型与原始数据的残差：

$$\mathrm{e_i} = \mathrm{R_{exp}}(\nu_\mathrm{i}) - \mathrm{R_{theory}}(\nu_\mathrm{i})$$



然后进行残差重采样，从原始残差集合$\{e_i\}$中有放回地随机抽取一组新的残差$\{e_i^*\}$，构建一组“伪”实验数据：

$$R_{pseudo}(\nu_i) = R_{theory}(\nu_i) + e_i^*$$

使用与原始数据相同的流程，对这组伪数据进行全谱拟合，得到一组新的最优参数，并将此过程重复执行多次，这里假设为 B 次。经过 B 次循环后，我们得到了 B 组最优参数的分布。通过计算这个分布的标准差，就可以得到每个参数的标准不确定度。同时，我们根据此也计算出置信区间约为 95%。

综合对比，两组数据在外延层厚度d上存在约 0.04μm 的差异，且其置信区间高度重叠，充分验证了多光束干涉模型在厚度求解方面的稳定性与鲁棒性。且拟合得到的载流子浓度与散射率处于合理范围，能够区分外延层与衬底的电学差异，说明该模型在测量外延层厚度d时中表现出较高的精度与稳定性。

5.3.7 碳化硅晶圆片多光束干涉模型的计算结果

根据上述研究，我们认为，在碳化硅晶圆片的红外干涉光谱测试中，光波在外延层表面与衬底界面间可能产生多次反射和透射，也有可能形成多光束干涉现象，这一效应会对外延层厚度的计算精度产生潜在影响。

在本研究中，我们不仅仅简单地消除其影响，而是将附件 1、附件 2 中的数据带入全谱拟合模型进行处理。通过全谱拟合方法对附件 1 和附件 2 进行拟合分析，得到以下结果：

将附件 1 数据带入计算，得到外延层厚度d为 $7.257 \pm 0.087\mu m$，根据附件 2 的数据，外延层厚度d为 $7.284 \pm 0.053\mu m$。附件 1 与附件 1 的拟合结果显示，尽管存在多光束干涉，其厚度计算结果与之前未考虑多光束干涉的结果差距并非很大，仍具有较高的置信度和稳定性，说明该模型在处理复杂干涉光谱时的可靠性与适用性。

六、模型的评价、改进与推广

6.1 模型的评价

6.1.1 模型的优点

(1) 该模型物理基础扎实，理论推导严谨，层次分明且逻辑严密，针对题目的三个问题，我们依次构建了极值点拟合模型、全局优化反演算法及全谱传输矩阵模型。模型设计遵循由简至繁、层层深入的原则。

(3) 该模型数值方法高效可靠且该模型结果稳定且实用性强，该建模创新性地采用差分进化算法进行全局优化，并结合最小二乘理论进行参数不确定性评估，为结果提供了统计意义上的可信度保证。

6.1.2 模型的缺点与评价

（1）该模型计算复杂度高，全谱拟合模型涉及传输矩阵的复数运算和多元参数优化，计算复杂度较高，对计算机资源要求较高，可以研究使用主成分分析等降维技术提高计算速度。

（2）该模型模型假设的局限性，模型假设界面光滑、层间平行、材料各向同性，与实际制备的样品可能存在一定差异，可在模型中引入界面粗糙度或过渡层的描述，通过有效介质近似等方法，使模型更贴近真实样品的界面特性。

6.2 模型的推广

本研究提出的模型框架具有良好的可扩展性和应用价值，其核心算法可广泛应用于其他半导体材料的外延层参数测量，只需调整相应的光学常数即可快速适配，具有很强的通用性。

七、参考文献

[1]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.

[2]鲍森,王宗缇,郑改革.碳化硅光栅结构中的强吸收及热辐射调控[J].光散射学报,2021,33(01):79-83.DOI:10.13883/j.issn1004-5929.202101011.

[3]郭明瑞,赵智炎,李健,等.飞秒激光制备高红外吸收碳化硅表面[J/OL].吉林大学学报(理学版),1-10[2025-09-07].https://doi.org/10.13413/j.cnki.jdxblxb.2024082.

[4]郭志成,董会宁,邓博,等.PbMoO4 晶体光学性质的第一性原理研究[J].材料导报,2010,24(S1):237-239.

[5]张维强,宋国乡.基于一种新的阈值函数的小波域信号去噪[J].西安电子科技大学学报,2004,(02):296-299+303.

[6]苏安,高英俊.含复介电常数一维光子晶体的滤波特性[J].中国激光,2009,36(06):1535-1538.

[7]刘波,王凌,金以慧.差分进化算法研究进展[J].控制与决策,2007,(07):721-729.DOI:10.13195/j.cd.2007.07.3.liub.001.

[8]唐军,杨华军,徐权,等.传输矩阵法分析一维光子晶体传输特性及其应用[J].红外与激光工程,2010,39(01):76-80.

[9]刘铁诚,胡敬佩,朱玲琳,等.Sellmeier 模型表征混合液晶双折射率色散的实验研究[J].中国激光,2020,47(08):155-160.

[10]Refractiveindex 数据库，https://refractiveindex.info/?shelf=main&book=SiC&page=Wang-4H-o

[11]Refractiveindex 数据库，https://refractiveindex.info/?book=Si&page=Salzberg&

shelf=main

[12]李建明.基于物理吸收和薄膜厚度的贵金属介电参数Lorentz-Drude模型优化分析研究[D].兰州大学,2014.

[13]马礼敦.X射线粉末衍射的新起点—Rietveld全谱拟合[J].物理学进展,1996,(02):115-135.

[14]Sextl E ,Kudritzki P R ,Bresolin F , et al.The TYPHOON Stellar Population Synthesis Survey. II. Pushing Full Spectral Fitting to the Limit in the Nearby Grand Design Barred Spiral M83[J].The Astrophysical Journal,2025,987(2):138-138.

[15]Zaccardi S ,Brahimetaj R ,Rodriguez E , et al.Impact of infrared interference on Azure Kinect's motion tracking performance during validation studies against marker-based gold standard[J].Gait & Posture,2024,113(S1):262-263.

[16]B ,B. F ,Zs. N , et al.Analysis of malaria infection byproducts with Mueller matrix transmission ellipsometry[J].Thin Solid Films,2023,766

[17]Shendi J A ,Mohammadian L ,Ghanizadeh R , et al.Optimal synchronization of overcurrent relays in active distribution networks using a hybrid genetic-differential optimization algorithm[J].Electrical Engineering,2025,107(6):1-16.

附录

附录一

介绍：支撑材料的文件列表

数据文件和代码使用说明.txt

AI 工具使用详情.pdf

代码文件：

1. 问题二中数据预处理及可视化分析的代码（data_visualization_analysis.py）
2. 问题一、二中极值点迭代优化算法及可视化分析的代码实现（iterative_extremum_fitter.py）
3. 问题一、二中极值点迭代优化算法及可视化分析的代码实现（iterative_extremum_fitter.py）
4. 问题二中多角度测量一致性分析的代码（run_multi_angle_analysis.py）
5. 问题三的基于全谱拟合的 SiC 外延层厚度拟合算法的代码（full_spectrum_thickness.py）
6. 将 Excel 转化为 CSV 文件的代码（xlsx_to_csv.py）

数据文件：

1. data 路径下 csv 格式的数据文件，包括附件 1.csv、附件 2.csv、附件 3.csv

2. data/denoised 路径去噪后的光谱数据下的 denoised_附件 1.csv、denoised_附件 2.csv、denoised_附件 3.csv

程序运行输出的可视化图片文件（data/figures 路径下）：

1. raw_spectra_analysis.png(原始数据光谱可视化图)
2. wavelet_denoising_summary.png(小波滤波去噪效果综合分析图)
3. denoising_residual_analysis.png(去噪残差分析图)
4. spectrum_extrema.png(光谱与极值点图)
5. residuals_vs_index.png(残差随样本索引分布图)
6. residuals_hist.png(干涉级数残差直方图)
7. refractive_index.png(拟合折射率 n 与 k 曲线图)
8. n_surface.png(折射率实部随波数与载流子浓度的表面可视化图)
9. interference_orders.png(干涉级数拟合结果图)
10. de_history.png(差分进化优化历史与参数轨迹图)
11. combined_summary.png(拟合摘要综合图)
12. 附件 1_fit_summary.png(附件 1 全谱拟合结果图、拟合残差、残差分布综合图)
13. 附件 2_fit_summary.png(附件 2 全谱拟合结果图、拟合残差、残差分布综合图)
14. 附件 3_fit_summary.png(附件 3 全谱拟合结果图、拟合残差、残差分布综合图)
15. 附件 4_fit_summary.png(附件全谱拟合结果图、拟合残差、残差分布综合图)

附录二

介绍：问题二中数据预处理及可视化分析的代码（data_visualization_analysis.py）

```
"""
碳化硅外延层厚度测量数据可视化分析
红外干涉法光谱数据分析

数据说明：
- 附件 1,2：碳化硅晶圆片，入射角 10°和 15°
- 附件 3,4：硅晶圆片，入射角 10°和 15°
- 数据格式：波数(cm^-1), 反射率(%)
"""

from cycler import cycler
from scipy import signal  # 信号处理库
from scipy.ndimage import uniform_filter1d
```

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import pywt  # 小波变换库
PYWT_AVAILABLE = True

class SpectralAnalyzer:
    """
    光谱数据分析器，用于处理和可视化碳化硅外延层厚度测量的红外干涉光谱数据。

    主要功能包括：
    - 数据加载与预处理
    - 数据质量评估
    - 原始光谱可视化
    - 基于小波变换的信号去噪
    - 去噪效果的定量与定性分析
    - 分析结果的图形化展示
    """

    def __init__(self):
        """初始化分析器，设置数据和绘图样式"""
        self.data = {}
        self.materials = {
            '附件 1': ('碳化硅', '10°'),
            '附件 2': ('碳化硅', '15°'),
            '附件 3': ('硅', '10°'),
            '附件 4': ('硅', '15°')
        }
        self._setup_plotting_style()

    def _setup_plotting_style(self):
        """设置全局绘图样式和中文字体"""
        import matplotlib

        sns.set_style('whitegrid')
        sns.set_context('talk', font_scale=0.9)
        matplotlib.rcParams['axes.prop_cycle'] = cycler('color', sns.color_palette('deep', 8))
        matplotlib.rcParams['figure.figsize'] = (12, 8)
        matplotlib.rcParams['axes.titleweight'] = 'bold'
        matplotlib.rcParams['axes.titlesize'] = 13
        matplotlib.rcParams['axes.labelsize'] = 11
        matplotlib.rcParams['legend.frameon'] = False
        matplotlib.rcParams['savefig.dpi'] = 300
        matplotlib.rcParams['figure.dpi'] = 100
        matplotlib.rcParams['mathtext.fontset'] = 'dejavusans'
        matplotlib.rcParams['axes.unicode_minus'] = True

        # 设置中文字体
        self.setup_chinese_fonts()

    def setup_chinese_fonts(self):
```

```
    """设置中文字体"""
    import platform
    import os
    from matplotlib.font_manager import FontProperties, fontManager
    system = platform.system()

    try:
        import matplotlib
        matplotlib.font_manager._rebuild()
    except:
        pass

    # 根据系统选择字体
    if system == "Windows":
        font_candidates = ['Microsoft YaHei', 'SimHei', 'KaiTi', 'FangSong', 'Microsoft JhengHei']
    elif system == "Darwin":  # macOS
        font_candidates = ['PingFang SC', 'STHeiti', 'SimHei', 'Arial Unicode MS']
    else:  # Linux
        font_candidates = ['WenQuanYi Micro Hei', 'SimHei', 'DejaVu Sans']

    font_set = False
    for font in font_candidates:
        try:
            # 设置全局字体
            plt.rcParams['font.sans-serif'] = [font] + plt.rcParams['font.sans-serif']
            # 确保负号能正常显示
            plt.rcParams['axes.unicode_minus'] = True

            # 简单测试
            test_fig = plt.figure(figsize=(1, 1))
            test_ax = test_fig.add_subplot(111)
            test_ax.text(0.5, 0.5, '中文测试', fontsize=10)
            plt.close(test_fig)

            print(f"成功设置中文字体: {font}")
            font_set = True
            break
        except Exception as e:
            continue

    if not font_set:
        print("无法设置中文字体，将使用备用方案")
        # 直接设置字体参数
        plt.rcParams['font.family'] = 'sans-serif'
        plt.rcParams['font.sans-serif'] = ['DejaVu Sans', 'Arial', 'Liberation Sans']
        plt.rcParams['axes.unicode_minus'] = False

def save_figure(self, fig, filename):
    """统一保存图片到 data/figures 目录并关闭图像以释放内存"""
    out_dir = 'data/figures'
    if not os.path.exists(out_dir):
        os.makedirs(out_dir)
    path = os.path.join(out_dir, filename)
    try:
        fig.savefig(path, dpi=300, bbox_inches='tight')
        print(f"已保存图片: {path}")
```

```
    except Exception as e:
        print(f"保存图片失败: {e}")
    plt.close(fig)

def load_data(self):
    """加载所有数据文件"""
    print("正在加载数据文件...")

    for filename in ['附件 1.csv', '附件 2.csv', '附件 3.csv', '附件 4.csv']:
        try:
            filepath = f'data/{filename}'
            df = pd.read_csv(filepath)

            # 标准化列名
            df.columns = ['wavenumber', 'reflectance']

            # 数据清洗：移除无效值
            df = df[df['reflectance'] > 0]  # 移除反射率为 0 的点
            df = df.dropna()  # 移除缺失值

            key = filename.replace('.csv', '')
            self.data[key] = df

            print(f"✓ {filename}: {len(df)} 个数据点")

        except Exception as e:
            print(f"✗ 加载 {filename} 失败: {e}")

def basic_statistics(self):
    """基本统计信息"""
    print("数据基本统计信息")

    for key, df in self.data.items():
        material, angle = self.materials[key]
        print(f"\n{key} ({material}, 入射角{angle}):")
        print(f"  波数范围: {df['wavenumber'].min():.1f} - {df['wavenumber'].max():.1f} cm^-1")
        print(f"  反射率范围: {df['reflectance'].min():.2f} - {df['reflectance'].max():.2f} %")
        print(f"  平均反射率: {df['reflectance'].mean():.2f} %")
        print(f"  反射率标准差: {df['reflectance'].std():.2f} %")

def plot_raw_spectra(self):
    """绘制原始光谱数据"""
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12), constrained_layout=True)
    axes = [ax1, ax2, ax3, ax4]

    colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

    for i, (key, df) in enumerate(self.data.items()):
        material, angle = self.materials[key]
        ax = axes[i]

        ax.plot(df['wavenumber'], df['reflectance'],
                color=colors[i], linewidth=1, alpha=0.8)
        ax.set_title(f'{key}: {material} (入射角{angle})', fontsize=12, fontweight='bold')
        ax.set_xlabel('波数  (cm$^{-1}$)')
```

```
            ax.set_ylabel('反射率 (%)')
            ax.grid(True, alpha=0.3)

            # 添加统计信息
            mean_ref = df['reflectance'].mean()
            ax.axhline(y=mean_ref, color='red', linestyle='--', alpha=0.7,
                       label=f'平均值: {mean_ref:.2f}%')
            ax.legend()

        # 保存并展示
        self.save_figure(fig, 'raw_spectra_analysis.png')
        try:
            fig.show()
        except:
            pass


    def wavelet_threshold_filter(self, y, wavelet='db4', sigma=None):
        """
        小波阈值滤波去噪

        参数:
        y: 输入信号
        wavelet: 小波类型
        sigma: 噪声标准差 (如果为 None 则自动估计)

        返回:
        filtered_y: 滤波后的信号
        """
        if not PYWT_AVAILABLE:
            print("PyWavelets 库不可用，返回原始信号")
            return y.copy()

        try:
            # 小波分解
            coeffs = pywt.wavedec(y, wavelet, mode='symmetric')

            # 估计噪声标准差
            if sigma is None:
                # 使用最高频细节系数估计噪声
                  # 0.6745 是高斯分布中位数绝对偏差(MAD)与标准差的关系常数 (sigma = MAD / 0.6745)
                sigma = np.median(np.abs(coeffs[-1])) / 0.6745

            # 计算阈值 (软阈值)
            threshold = sigma * np.sqrt(2 * np.log(len(y)))

            # 对细节系数进行阈值处理
            coeffs_thresh = list(coeffs)
            coeffs_thresh[1:] = [pywt.threshold(detail, threshold, mode='soft')
                                 for detail in coeffs[1:]]

            # 小波重构
            filtered_y = pywt.waverec(coeffs_thresh, wavelet, mode='symmetric')
```

```
            # 确保输出长度与输入相同
            if len(filtered_y) != len(y):
                filtered_y = filtered_y[:len(y)]

            return filtered_y

        except Exception as e:
            print(f"小波滤波失败: {e}")
            return y.copy()

    def estimate_noise_level(self, y, wavelet='db4'):
        """
        使用小波系数的绝对中位差(MAD)稳健地估计噪声水平

        参数:
        y: 输入信号
        wavelet: 小波类型

        返回:
        sigma: 估计的噪声标准差
        """
        if not PYWT_AVAILABLE:
            return np.std(np.diff(y))

        try:
            coeffs = pywt.wavedec(y, wavelet, mode='symmetric')
            # 根据高斯噪声的 MAD 公式估计 sigma
            # 0.6745 是高斯分布中位数绝对偏差(MAD)与标准差的关系常数
            sigma = np.median(np.abs(coeffs[-1])) / 0.6745
            return sigma
        except Exception as e:
            print(f"噪声估计失败: {e}")
            return np.std(np.diff(y))

    def calculate_snr(self, original, filtered):
        """计算信噪比改善"""
        try:
            # 使用稳健的小波方法估计噪声
            noise_original = self.estimate_noise_level(original)
            noise_filtered = self.estimate_noise_level(filtered)

            # 信号功率
            signal_power = np.var(filtered)

            # SNR 计算 (添加 epsilon 防止除以零)
            snr_original = 10 * np.log10(signal_power / (noise_original**2 + np.finfo(float).eps))
            snr_filtered = 10 * np.log10(signal_power / (noise_filtered**2 + np.finfo(float).eps))

            improvement = snr_filtered - snr_original

            return snr_original, snr_filtered, improvement

        except Exception as e:
            print(f"SNR 计算失败: {e}")
            return 0, 0, 0
```

```
def detailed_denoising_analysis(self):
    """详细的小波滤波性能分析，并保存去噪后的数据"""
    print("\n 正在进行详细小波滤波性能分析...")

    # 创建用于保存去噪数据的目录
    output_dir = 'data/denoised'
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
        print(f"创建目录: {output_dir}")

    # 分析所有数据文件
    results = {}

    for key, df in self.data.items():
        material, angle = self.materials[key]
        x = df['wavenumber'].values
        y = df['reflectance'].values

        # 应用小波滤波
        y_wavelet = self.wavelet_threshold_filter(y)

        # 保存去噪后的数据
        denoised_df = pd.DataFrame({'wavenumber': x, 'reflectance': y_wavelet})
        output_path = os.path.join(output_dir, f'denoised_{key}.csv')
        denoised_df.to_csv(output_path, index=False)
        print(f"  ✓ {key}: 去噪数据已保存到 {output_path}")

        # 计算性能指标
        snr_orig, snr_wt, snr_improve_wt = self.calculate_snr(y, y_wavelet)

        # 计算均方根误差 (相对于高度平滑版本)
        # 使用移动平均作为参考基准
        window_size = max(51, len(y)//20)
        y_smooth = uniform_filter1d(y, size=window_size)
        rmse_orig = np.sqrt(np.mean((y - y_smooth)**2))
        rmse_wt = np.sqrt(np.mean((y_wavelet - y_smooth)**2))

        # 计算保边缘能力 (梯度保持)
        grad_orig = np.gradient(y)
        grad_wt = np.gradient(y_wavelet)

        edge_preserve_wt = np.corrcoef(grad_orig, grad_wt)[0,1]

        # 计算噪声减少 (添加 epsilon 防止除以零)
        noise_orig = self.estimate_noise_level(y)
        noise_wt = self.estimate_noise_level(y_wavelet)
        noise_reduction = (1 - noise_wt / (noise_orig + np.finfo(float).eps)) * 100

        results[key] = {
            'material': material,
            'angle': angle,
            'snr_improve_wt': snr_improve_wt,
            'rmse_reduction_wt': (rmse_orig - rmse_wt) / rmse_orig * 100,
            'edge_preserve_wt': edge_preserve_wt,
            'noise_reduction': noise_reduction
```

```
        }

    # 打印分析结果
    print("\n" + "="*80)
    print("小波滤波详细性能分析结果")
    print("="*80)

    for key, result in results.items():
        print(f"\n{key} ({result['material']}, 入射角{result['angle']}):")
        print(f"   小波阈值滤波性能:")
        print(f"     SNR 改善: {result['snr_improve_wt']:.2f} dB")
        print(f"     RMSE 减少: {result['rmse_reduction_wt']:.2f}%")
        print(f"     边缘保持: {result['edge_preserve_wt']:.4f}")
        print(f"     噪声减少: {result['noise_reduction']:.1f}%")

        # 性能评估
        overall_score = (result['snr_improve_wt'] + result['rmse_reduction_wt']/10 +
                         result['edge_preserve_wt']*10 + result['noise_reduction']/10)

        if overall_score > 15:
            quality = "优秀"
        elif overall_score > 10:
            quality = "良好"
        elif overall_score > 5:
            quality = "一般"
        else:
            quality = "较差"

        print(f"     综合评价: {quality} (得分: {overall_score:.2f})")

    # 总体统计
    print(f"\n 总体统计  (平均值):")
    avg_snr_wt = np.mean([r['snr_improve_wt'] for r in results.values()])
    avg_edge_wt = np.mean([r['edge_preserve_wt'] for r in results.values()])
    avg_noise_reduction = np.mean([r['noise_reduction'] for r in results.values()])

    print(f"   小波阈值滤波:")
    print(f"     平均 SNR 改善: {avg_snr_wt:.2f} dB")
    print(f"     平均边缘保持: {avg_edge_wt:.4f}")
    print(f"     平均噪声减少: {avg_noise_reduction:.1f}%")

    return results

def plot_zoomed_denoising_comparison(self):
    """
    绘制合并后的去噪效果分析图。
    包含三个局部放大区域和一个带 SNR 改善指标的全光谱对比图。
    """
    print("\n 正在生成合并去噪分析图...")

    # 选择附件 1 进行分析
    sample_key = '附件 1'
    df = self.data[sample_key]
    x = df['wavenumber'].values
```

```
y = df['reflectance'].values

# 应用小波滤波
y_wavelet = self.wavelet_threshold_filter(y)

# 选择三个有代表性的区域进行放大
# 区域 1：低频区域 (800-1200 cm^-1)
# 区域 2：中频区域 (1500-2000 cm^-1)
# 区域 3：高频区域 (2500-3000 cm^-1)

fig, axes = plt.subplots(2, 2, figsize=(16, 12), constrained_layout=True)
fig.suptitle('小波滤波去噪效果综合分析', fontsize=16, fontweight='bold')

# 区域 1：低频区域
mask1 = (x >= 800) & (x <= 1200)
x1 = x[mask1]
y1 = y[mask1]
y1_wt = y_wavelet[mask1]

ax = axes[0, 0]
ax.plot(x1, y1, 'b-', linewidth=1, alpha=0.6, label='原始数据')
ax.plot(x1, y1_wt, 'g-', linewidth=2, label='小波滤波')
ax.set_title('局部放大 (低频区域: 800-1200 cm$^{-1}$)', fontweight='bold')
ax.set_xlabel('波数 (cm$^{-1}$)')
ax.set_ylabel('反射率 (%)')
ax.grid(True, alpha=0.3)
ax.legend()

# 区域 2：中频区域
mask2 = (x >= 1500) & (x <= 2000)
x2 = x[mask2]
y2 = y[mask2]
y2_wt = y_wavelet[mask2]

ax = axes[0, 1]
ax.plot(x2, y2, 'b-', linewidth=1, alpha=0.6, label='原始数据')
ax.plot(x2, y2_wt, 'g-', linewidth=2, label='小波滤波')
ax.set_title('局部放大 (中频区域: 1500-2000 cm$^{-1}$)', fontweight='bold')
ax.set_xlabel('波数 (cm$^{-1}$)')
ax.set_ylabel('反射率 (%)')
ax.grid(True, alpha=0.3)
ax.legend()

# 区域 3：高频区域
mask3 = (x >= 2500) & (x <= 3000)
x3 = x[mask3]
y3 = y[mask3]
y3_wt = y_wavelet[mask3]

ax = axes[1, 0]
ax.plot(x3, y3, 'b-', linewidth=1, alpha=0.6, label='原始数据')
ax.plot(x3, y3_wt, 'g-', linewidth=2, label='小波滤波')
ax.set_title('局部放大 (高频区域: 2500-3000 cm$^{-1}$)', fontweight='bold')
ax.set_xlabel('波数 (cm$^{-1}$)')
```

```
    ax.set_ylabel('反射率 (%)')
    ax.grid(True, alpha=0.3)
    ax.legend()

    # 第四个子图：替换为带 SNR 和噪声减少指标的全光谱图
    ax = axes[1, 1]
    ax.plot(x, y, 'b-', linewidth=1, alpha=0.4, label='原始数据')
    ax.plot(x, y_wavelet, 'g-', linewidth=1.5, label='小波滤波')
    ax.set_title('全光谱去噪效果与性能指标', fontweight='bold', fontsize=12)
    ax.set_xlabel('波数 (cm$^{-1}$)')
    ax.set_ylabel('反射率 (%)')
    ax.grid(True, alpha=0.3)
    ax.legend()

    # 计算 SNR 改善
    snr_orig, snr_wt, snr_improve_wt = self.calculate_snr(y, y_wavelet)
    ax.text(0.02, 0.98, f'SNR 改善: {snr_improve_wt:.2f} dB',
            transform=ax.transAxes, va='top', fontsize=10,
            bbox=dict(boxstyle='round', facecolor='lightgreen', alpha=0.8))

    # 计算噪声减少百分比
    noise_orig = self.estimate_noise_level(y)
    noise_wt = self.estimate_noise_level(y_wavelet)
    noise_reduction = (1 - noise_wt / (noise_orig + np.finfo(float).eps)) * 100
    ax.text(0.02, 0.85, f'噪声减少: {noise_reduction:.1f}%',
            transform=ax.transAxes, va='top', fontsize=10,
            bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.8))

    # 保存合并后的图
    self.save_figure(fig, 'wavelet_denoising_summary.png')
    try:
        fig.show()
    except:
        pass

def plot_denoising_residual(self, sample_key='附件 1'):
    """
    对去噪过程的残差进行时域和频域分析。
    一个好的去噪算法，其残差应表现为白噪声特性。
    """
    print(f"\n 正在对 {sample_key} 进行去噪残差分析...")

    df = self.data[sample_key]
    x = df['wavenumber'].values
    y_orig = df['reflectance'].values

    # 获取去噪后的数据
    y_denoised = self.wavelet_threshold_filter(y_orig)

    # 计算残差
    residual = y_orig - y_denoised

    fig, axes = plt.subplots(1, 2, figsize=(16, 6), constrained_layout=True)
    fig.suptitle('去噪残差分析', fontsize=16, fontweight='bold')
```

```
        # 时域分析
        ax1 = axes[0]
        ax1.plot(x, residual, color='gray', linewidth=0.8, alpha=0.9)
        ax1.axhline(0, color='red', linestyle='--', linewidth=1.5)
        ax1.set_title('时域残差 (原始信号 - 去噪信号)', fontweight='bold')
        ax1.set_xlabel('波数 (cm$^{-1}$)')
        ax1.set_ylabel('残差幅值')
        ax1.grid(True, alpha=0.3)

        # 添加统计信息
        mean_res = np.mean(residual)
        # 避免因浮点数精度问题显示 "-0.0000"
        if abs(mean_res) < 5e-5:  # 阈值小于格式化显示的最小精度
            mean_res = 0.0
        std_res = np.std(residual)
        ax1.text(0.02, 0.98, f'均值: {mean_res:.4f}\n 标准差: {std_res:.4f}',
                 transform=ax1.transAxes, va='top', fontsize=10,
                 bbox=dict(boxstyle='round', facecolor='white', alpha=0.9))

        # 频域分析 (功率谱密度)
        ax2 = axes[1]
        # 使用 signal.welch 来分析残差的频率成分
        fs = 1 / np.mean(np.diff(x)) # 采样频率的倒数是波数间隔
        freqs, psd = signal.welch(residual, fs=fs, nperseg=1024)

        ax2.semilogy(freqs, psd, color='darkblue')
        ax2.set_title('残差功率谱密度 (PSD)', fontweight='bold')
        ax2.set_xlabel('频率 (cycles / cm$^{-1}$)')
        ax2.set_ylabel('功率 / 频率 (dB/Hz)')
        ax2.grid(True, which='both', linestyle='--', alpha=0.3)
        ax2.text(0.98, 0.98, '注: 理想的去噪残差应接近白噪声,其能量在频域中分布平坦, 无明显峰值。',
                 transform=ax2.transAxes, va='top', ha='right', fontsize=9,
                 bbox=dict(boxstyle='round', facecolor='lightyellow', alpha=0.8))

        self.save_figure(fig, 'denoising_residual_analysis.png')
        try:
            fig.show()
        except:
            pass

    def data_quality_assessment(self):
        """数据质量评估"""
        print("\n" + "="*60)
        print("数据质量评估")
        print("="*60)

        for key, df in self.data.items():
            material, angle = self.materials[key]
            print(f"\n{key} ({material}, 入射角{angle}):")

            # 数据完整性
            total_points = len(df)
            valid_points = df['reflectance'].notna().sum()
            print(f"  数据完整性: {valid_points}/{total_points} ({100*valid_points/total_points:.1f}%)")
```

```
            # 数据范围合理性
            ref_range = df['reflectance'].max() - df['reflectance'].min()
            print(f"   反射率动态范围: {ref_range:.2f}%")

            # 数据平滑性 (基于更稳健的噪声估计)
            noise_level = self.estimate_noise_level(df['reflectance'].values)
            smoothness = 1 / (1 + noise_level)
            print(f"   数据平滑性指标: {smoothness:.4f} (基于小波噪声估计)")


    def run_complete_analysis(self):
        """运行完整分析"""
        print("开始碳化硅外延层厚度测量数据可视化分析")
        print("="*60)

        self.load_data()  # 加载数据

        self.basic_statistics()  # 基本统计

        print("\n 正在生成原始光谱图...")
        self.plot_raw_spectra()  # 绘制原始光谱

        print("\n" + "="*60)
        print("开始去噪分析")
        self.detailed_denoising_analysis()  # 详细去噪分析与数据保存

        print("\n 正在生成合并去噪分析图...")
        self.plot_zoomed_denoising_comparison()  # 合并去噪效果综合分析图

        self.plot_denoising_residual()  # 残差分析

        self.data_quality_assessment()  # 数据质量评估

        print("\n" + "="*80)
        print("分析完成！")

if __name__ == "__main__":
    analyzer = SpectralAnalyzer()
    analyzer.run_complete_analysis()
```

附录三

介绍：问题一、二中极值点迭代优化算法及可视化分析的代码实现（iterative_extremum_fitter.py）

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
极值点解析迭代优化法

物理原理：
- 基于极值点的干涉条件，但考虑了折射率 n 是波数ν的函数 n=n(ν)。
- 由于 n(ν)依赖于未知的载流子浓度(N)和散射率(γ)，导致厚度(d)与这些参数耦合，无法直接求解。
- 本方法将 d, N, γ 等作为未知参数，通过优化算法寻找最佳参数组合。
- 优化目标：使得在所有实验光谱的极值点上，根据物理模型计算出的干涉级数 m 最接近于整数（或
```

半整数）。

算法核心：
1. 从实验光谱中精确提取极大值和极小值点的位置（波数）。
2. 定义一个目标函数，衡量在给定参数(d, N_epi, γ_epi)下，所有极值点计算出的干涉级数偏离整数/半整数的程度。
3. 使用全局优化算法（差分进化）来最小化该目标函数，从而找到最佳的物理参数。
"""

```
import numpy as np
import pandas as pd
from scipy.optimize import differential_evolution
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
import logging
import time
import warnings
import os
from dataclasses import dataclass
from typing import Dict, Tuple, Optional, List
from matplotlib.font_manager import FontProperties

# 小波去噪相关库
try:
    import pywt
    PYWT_AVAILABLE = True
except ImportError:
    PYWT_AVAILABLE = False
    warnings.warn("PyWavelets 库未安装，将跳过小波去噪功能")

def find_chinese_font():
    """
    在系统中查找可用的中文字体文件，并返回一个 FontProperties 对象。
    """
    if os.name == 'nt':
        font_paths = [
            'C:/Windows/Fonts/msyh.ttc',
            'C:/Windows/Fonts/simhei.ttf',
            'C:/Windows/Fonts/simsun.ttc',
        ]
        for font_path in font_paths:
            if os.path.exists(font_path):
                print(f"找到系统字体: {font_path}")
                return FontProperties(fname=font_path, size=12)
    try:
        font_names = ['Microsoft YaHei', 'SimHei', 'SimSun', 'KaiTi', 'FangSong']
        for font_name in font_names:
            try:
                return FontProperties(font_name, size=12)
            except Exception:
                continue
    except Exception:
        pass
    return None

logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
```

```
logger = logging.getLogger(__name__)

CHINESE_FONT = find_chinese_font()
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['figure.figsize'] = (12, 8)
plt.style.use('default')

def load_spectrum_data(filepath: str) -> Tuple[np.ndarray, np.ndarray]:
    df = pd.read_csv(filepath)
    wavenumber = df.iloc[:, 0].values
    reflectance = df.iloc[:, 1].values
    mask = ~(np.isnan(wavenumber) | np.isnan(reflectance))
    wavenumber = wavenumber[mask]
    reflectance = reflectance[mask]
    sort_idx = np.argsort(wavenumber)
    wavenumber = wavenumber[sort_idx]
    reflectance = reflectance[sort_idx]
    return wavenumber, reflectance

@dataclass
class MaterialParams:
    """材料参数数据类"""
    A: float
    B: float
    C: float
    m_eff: float
    eps_inf: float
    # Sellmeier 系数列表，每项为 (B_j, C_j) 对，应以波长 (μm) 为单位 (C_j 为 μm^2)
    sellmeier_terms: Optional[List[Tuple[float, float]]] = None

class IterativeExtremumFitter:
    """
    基于极值点迭代优化的拟合器
    """
    C_LIGHT = 2.99792458e10
    E_CHARGE = 1.602176634e-19
    EPS_0 = 8.854187817e-12
    M_ELECTRON = 9.10938356e-31

    # 为 sic 提供 topic.md 中的 Sellmeier 系数（lambda 单位 μm）
    MATERIALS = {
        'sic': MaterialParams(
            A=6.7, B=1.5e4, C=200, m_eff=0.5, eps_inf=6.5,
            sellmeier_terms=[
                (0.20075, -12.07224),
                (5.54861, 0.02641),
                (35.65066, 1268.24708),
            ]
        ),
        'si': MaterialParams(A=11.4, B=0.0, C=0, m_eff=0.26, eps_inf=11.7)
    }

    def __init__(self, material='sic'):
        """
        初始化拟合器

        Parameters:
```

```
    ----------
    material : str
        材料类型, 'sic'  或  'si'
    """
    if material not in self.MATERIALS:
        raise ValueError(f"不支持的材料类型: {material}")
    self.material_params = self.MATERIALS[material]
    self.material = material
    self.wavelet_params = {
        'wavelet': 'db4',
        'mode': 'symmetric',
        'threshold_mode': 'soft'
    }

def calculate_complex_permittivity(self, wavenumber: np.ndarray,
                                   carrier_density: float,
                                   scattering_rate: float) -> np.ndarray:
    """
    计算 Drude-Sellmeier 复介电函数
    """
    eps_bound = None
    if self.material_params.sellmeier_terms:
        # topic.md 中的 Sellmeier 形式使用波长 (μm)。代码中输入为波数 (cm^-1).
        # 需要将波数转换为波长 (μm): lambda_um = 1e4 / wavenumber (cm^-1) -> μm
        lambda_um = 1e4 / wavenumber
        # Sellmeier: n^2 - 1 = sum( B_j * lambda^2 / (lambda^2 - C_j) )
        # Here sellmeier_terms contains (B_j, C_j) with C_j in μm^2
        s = np.zeros_like(lambda_um, dtype=float)
        for Bj, Cj in self.material_params.sellmeier_terms:
            denom_s = lambda_um**2 - Cj
            denom_s = np.where(np.abs(denom_s) < 1e-12, 1e-12, denom_s)
            s += Bj * (lambda_um**2) / denom_s
        n2 = 1.0 + s
        eps_bound = n2.astype(complex)
    else:
        # 回退到原始简单共振项表示，输入均以波数 (cm^-1) 为单位
        A, B, C = self.material_params.A, self.material_params.B, self.material_params.C
        denom = wavenumber**2 - C**2
        denom = np.where(np.abs(denom) < 1e-6, 1e-6, denom)
        eps_bound = (A + B / denom).astype(complex)

    # 支持 scalar 或 array 的 carrier_density（按元素计算 Drude 项）。
    omega = 2.0 * np.pi * self.C_LIGHT * wavenumber
    # 将载流子浓度规范化为 numpy 数组
    N_arr = np.asarray(carrier_density)
    m_star = self.material_params.m_eff * self.M_ELECTRON

    # 如果 carrier_density 为标量
    if N_arr.ndim == 0:
        if N_arr <= 0:
            return eps_bound
        N_m3 = float(N_arr) * 1e6
        omega_p_squared = (N_m3 * self.E_CHARGE**2) / (self.EPS_0 * m_star)
        denom_drude = omega**2 + 1j * scattering_rate * omega
        epsilon_drude = -omega_p_squared / denom_drude
        return (eps_bound + epsilon_drude).astype(complex)
```

```
    # 否则按元素计算（期望 carrier_density 与 wavenumber 形状可广播或相同）
    N_m3 = N_arr * 1e6
    # 尝试广播 N_m3 到与 omega 相同的形状
    try:
        N_m3_b = np.broadcast_to(N_m3, omega.shape)
    except Exception:
        # 若广播失败，尝试展平再匹配长度
        N_m3_b = np.asarray(N_m3).ravel()
        if N_m3_b.size != omega.ravel().size:
            raise ValueError('carrier_density 与 wavenumber 的长度无法匹配进行向量化计算。')
        N_m3_b = N_m3_b.reshape(omega.shape)

    omega_p_squared = (N_m3_b * self.E_CHARGE**2) / (self.EPS_0 * m_star)
    denom_drude = omega**2 + 1j * scattering_rate * omega
    epsilon_drude = -omega_p_squared / denom_drude

    # 对于非正的载流子浓度保持不加 Drude 项
    mask_nonpositive = (np.asarray(carrier_density) <= 0)
    if np.any(mask_nonpositive):
        # 广播 mask 到形状
        try:
            mask_b = np.broadcast_to(mask_nonpositive, omega.shape)
        except Exception:
            mask_b = np.broadcast_to(np.asarray(mask_nonpositive).ravel(), omega.shape)
        epsilon_drude = np.where(mask_b, 0.0 + 0.0j, epsilon_drude)

    return (eps_bound + epsilon_drude).astype(complex)

def calculate_refractive_index(self, wavenumber: np.ndarray,
                               carrier_density: float,
                               scattering_rate: float) -> np.ndarray:
    """
    计算复折射率 ñ = n + ik
    """
    epsilon = self.calculate_complex_permittivity(wavenumber, carrier_density, scattering_rate)
    return np.sqrt(epsilon)

def wavelet_denoise(self, reflectance: np.ndarray,
                    wavelet: Optional[str] = None,
                    sigma: Optional[float] = None) -> np.ndarray:
    """
    小波阈值去噪处理
    """
    if not PYWT_AVAILABLE:
        warnings.warn("PyWavelets 库不可用，跳过小波去噪")
        return reflectance.copy()
    try:
        wavelet_type = wavelet or self.wavelet_params['wavelet']
        mode = self.wavelet_params['mode']
        threshold_mode = self.wavelet_params['threshold_mode']
        coeffs = pywt.wavedec(reflectance, wavelet_type, mode=mode)
        if sigma is None:
            sigma = np.median(np.abs(coeffs[-1])) / 0.6745
        threshold = sigma * np.sqrt(2 * np.log(len(reflectance)))
        coeffs_thresh = list(coeffs)
```

```
            coeffs_thresh[1:] = [pywt.threshold(detail, threshold, mode=threshold_mode)
                                 for detail in coeffs[1:]]
            denoised = pywt.waverec(coeffs_thresh, wavelet_type, mode=mode)
            if len(denoised) != len(reflectance):
                denoised = denoised[:len(reflectance)]
            return denoised
        except Exception as e:
            warnings.warn(f"小波去噪失败: {e}，返回原始数据")
            return reflectance.copy()

    def find_extrema(self, wavenumber: np.ndarray, reflectance: np.ndarray,
                     prominence: float = 0.1, distance: int = 10) -> Tuple[np.ndarray, np.ndarray]:
        """
        从光谱数据中寻找极大值和极小值点
        """
        peak_indices, _ = find_peaks(reflectance, prominence=prominence, distance=distance)
        valley_indices, _ = find_peaks(-reflectance, prominence=prominence, distance=distance)
        peaks = wavenumber[peak_indices]
        valleys = wavenumber[valley_indices]
        logger.info("找到 %d 个极大值点, %d 个极小值点。", len(peaks), len(valleys))
        return peaks, valleys

    def calculate_interference_order(self, params: Dict, wavenumber: np.ndarray,
                                     incident_angle: float) -> np.ndarray:
        """
        计算理论干涉级数 m
        """
        d = params['d']
        n_epi = self.calculate_refractive_index(
            wavenumber, params['N_epi'], params['gamma_epi']
        )
        theta0_rad = np.radians(incident_angle)
        sqrt_term = np.sqrt(
            np.maximum(0, n_epi.real**2 - np.sin(theta0_rad)**2)
        )
        d_cm = d * 1e-4
        m_theory = 2 * d_cm * wavenumber * sqrt_term
        return m_theory

    def objective_function(self, x: np.ndarray, param_names: List[str],
                           peaks: np.ndarray, valleys: np.ndarray,
                           incident_angle: float) -> float:
        """
        目标函数：衡量干涉级数偏离整数/半整数的程度
        """
        params = dict(zip(param_names, x))
        cost = 0.0
        if len(peaks) > 0:
            m_peaks = self.calculate_interference_order(params, peaks, incident_angle)
            cost += np.sum((m_peaks - np.round(m_peaks))**2)
        if len(valleys) > 0:
            m_valleys = self.calculate_interference_order(params, valleys, incident_angle)
            cost += np.sum((m_valleys - (np.round(m_valleys - 0.5) + 0.5))**2)
        return cost

    def _residuals_vector(self, x: np.ndarray, param_names: List[str],
                          peaks: np.ndarray, valleys: np.ndarray,
```

```
                    incident_angle: float) -> np.ndarray:
    """
    构建残差向量：极大值对应 m - round(m)，极小值对应 m - (round(m-0.5)+0.5)
    用于数值雅可比和不确定度估计。
    """
    params = dict(zip(param_names, x))
    res_list = []
    if len(peaks) > 0:
        m_peaks = self.calculate_interference_order(params, peaks, incident_angle)
        res_peaks = m_peaks - np.round(m_peaks)
        res_list.append(res_peaks.astype(float))
    if len(valleys) > 0:
        m_valleys = self.calculate_interference_order(params, valleys, incident_angle)
        res_valleys = m_valleys - (np.round(m_valleys - 0.5) + 0.5)
        res_list.append(res_valleys.astype(float))
    if len(res_list) == 0:
        return np.zeros(0, dtype=float)
    return np.concatenate(res_list)

def estimate_parameter_uncertainties(self, fitted_params: Dict[str, float],
                                     param_names: List[str],
                                     peaks: np.ndarray, valleys: np.ndarray,
                                     incident_angle: float,
                                     epsilon: float = 1e-6) -> Tuple[Dict[str, float], np.ndarray]:
    # 使用有限差分估计参数不确定度。
    x0 = np.array([fitted_params[name] for name in param_names], dtype=float)
    n_params = len(x0)
    r0 = self._residuals_vector(x0, param_names, peaks, valleys, incident_angle)
    n_resid = len(r0)
    if n_resid == 0 or n_params == 0:
        logger.warning("无法计算不确定度：残差或参数数目为 0。")
        return {name: np.nan for name in param_names}, np.full((n_params, n_params), np.nan)

    J = np.zeros((n_resid, n_params), dtype=float)
    for i in range(n_params):
        xi = x0[i]
        delta = epsilon * max(1.0, abs(xi))
        x_pert = x0.copy()
        x_pert[i] = xi + delta
        r_pert = self._residuals_vector(x_pert, param_names, peaks, valleys, incident_angle)
        if len(r_pert) != n_resid:
            delta = epsilon * max(1.0, abs(xi)) * 0.1
            x_pert[i] = xi + delta
            r_pert = self._residuals_vector(x_pert, param_names, peaks, valleys, incident_angle)
            if len(r_pert) != n_resid:
                raise RuntimeError("数值雅可比计算时残差长度不匹配。")
        J[:, i] = (r_pert - r0) / delta

    dof = max(1, n_resid - n_params)
    rss = float(np.sum(r0**2))
    sigma2 = rss / dof

    JTJ = J.T.dot(J)
    try:
        JTJ_inv = np.linalg.pinv(JTJ)
    except Exception as e:
        logger.warning("计算 JTJ 逆矩阵失败: %s", e)
```

```
            JTJ_inv = np.linalg.pinv(JTJ)

        covariance = JTJ_inv * sigma2
        uncertainties = np.sqrt(np.abs(np.diag(covariance)))
        param_uncertainties = {name: float(uncertainties[i]) for i, name in enumerate(param_names)}
        return param_uncertainties, covariance

    def fit_spectrum(self, wavenumber: np.ndarray, reflectance: np.ndarray,
                     incident_angle: float, initial_guess: Optional[Dict] = None,
                     bounds: Optional[Dict] = None,
                     use_denoise: bool = True) -> Dict:
        """
        执行拟合过程
        """
        reflectance_original = reflectance.copy()
        if use_denoise:
            logger.info("应用小波去噪预处理...")
            reflectance = self.wavelet_denoise(reflectance)

        peaks, valleys = self.find_extrema(wavenumber, reflectance)
        if len(peaks) + len(valleys) < 5:
            warnings.warn("找到的极值点过少，结果可能不可靠。")
            return {'success': False, 'message': '极值点太少'}

        if initial_guess is None:
            initial_guess = {
                'd': 10.0,
                'N_epi': 1e16,
                'gamma_epi': 1e13,
            }

        if bounds is None:
            bounds = {
                'd': (4.0, 20.0),
                'N_epi': (1e14, 1e18),
                'gamma_epi': (1e11, 1e15),
            }

        param_names = list(initial_guess.keys())
        bounds_list = [bounds[name] for name in param_names]

        logger.info("=== 极值点迭代优化 (入射角 %s°) ===", incident_angle)
        logger.info("参数边界: %s", bounds)

        start_time = time.time()

        de_history = {'x': [], 'cost': [], 'convergence': []}

        def _de_callback(xk, convergence=None):
            try:
                cost_xk = float(self.objective_function(xk, param_names, peaks, valleys, incident_angle))
            except Exception:
                cost_xk = np.nan
            de_history['x'].append(np.array(xk, dtype=float))
            de_history['cost'].append(cost_xk)
            de_history['convergence'].append(float(convergence) if convergence is not None else np.nan)
            return False
```

```
result = differential_evolution(
    self.objective_function,
    bounds=bounds_list,
    args=(param_names, peaks, valleys, incident_angle),
    strategy='best1bin',
    maxiter=500,
    popsize=20,
    tol=0.01,
    mutation=(0.5, 1),
    recombination=0.7,
    seed=42,
    callback=_de_callback
)

elapsed_time = time.time() - start_time

fitted_params = dict(zip(param_names, result.x))
final_cost = result.fun

m_peaks = self.calculate_interference_order(fitted_params, peaks, incident_angle)
m_valleys = self.calculate_interference_order(fitted_params, valleys, incident_angle)

peak_residuals = m_peaks - np.round(m_peaks)
valley_residuals = m_valleys - (np.round(m_valleys - 0.5) + 0.5)

rmse_m = np.sqrt(np.mean(np.concatenate([peak_residuals**2, valley_residuals**2])))

try:
    param_uncertainties, covariance = self.estimate_parameter_uncertainties(
        fitted_params, param_names, peaks, valleys, incident_angle
    )
except Exception as e:
    logger.warning("参数不确定度估计失败: %s", e)
    param_uncertainties = {name: np.nan for name in param_names}
    covariance = np.full((len(param_names), len(param_names)), np.nan)

fit_result = {
    'success': result.success,
    'params': fitted_params,
    'de_history': de_history,
    'param_uncertainties': param_uncertainties,
    'covariance': covariance,
    'cost': final_cost,
    'rmse_m': rmse_m,
    'elapsed_time': elapsed_time,
    'extrema': {'peaks': peaks, 'valleys': valleys},
    'm_theory': {'peaks': m_peaks, 'valleys': m_valleys},
    'denoised_reflectance': reflectance if use_denoise else None
}

logger.info("优化完成  (%.1fs)", elapsed_time)
logger.info("成功: %s", result.success)
logger.info("最终 Cost: %.4f", final_cost)
logger.info("干涉级数 RMSE: %.4f", rmse_m)
logger.info("拟合参数:")
for name, value in fitted_params.items():
```

```
            if name == 'd':
                                                logger.info("      %s: %.3f μm ± %.3f", name, value,
fit_result['param_uncertainties'].get(name, np.nan))
            elif 'N_' in name:
                                               logger.info("      %s: %.2e cm⁻³ ± %.2e", name, value,
fit_result['param_uncertainties'].get(name, np.nan))
            elif 'gamma_' in name:
                                                logger.info("      %s: %.2e s⁻¹ ± %.2e", name, value,
fit_result['param_uncertainties'].get(name, np.nan))

        return fit_result

    def plot_fit_result(self, wavenumber: np.ndarray, reflectance_data: np.ndarray,
                        fit_result: Dict, incident_angle: float, save_path: Optional[str] = None,
                        save_dir: Optional[str] = None):
        if save_dir:
            os.makedirs(save_dir, exist_ok=True)

        figs = []

        fig = self._plot_spectrum_with_extrema(wavenumber, reflectance_data, fit_result, incident_angle)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'spectrum_extrema.png'))

        fig = self._plot_interference_orders(fit_result, incident_angle)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'interference_orders.png'))

        fig = self._plot_refractive_index(wavenumber, fit_result)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'refractive_index.png'))

        fig = self._plot_residuals_histogram(fit_result)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'residuals_hist.png'))

        fig = self._plot_residuals_vs_index(fit_result)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'residuals_vs_index.png'))

        fig = self._plot_optimization_history(fit_result)
        if fig is not None:
            figs.append(fig)
            if save_dir:
                self._save_current_fig(fig, os.path.join(save_dir, 'de_history.png'))

        if save_path:
```

```
                self._plot_combined_summary(wavenumber, reflectance_data, fit_result, incident_angle,
save_path)

        try:
            if len(figs) > 0:
                plt.show()
        finally:
            for f in figs:
                try:
                    plt.close(f)
                except Exception:
                    pass

    def _save_current_fig(self, path: str):
        try:
            fig = plt.gcf()
                                                fig.savefig(path,    dpi=plt.rcParams.get('savefig.dpi',    300),
bbox_inches=plt.rcParams.get('savefig.bbox', 'tight'))
            logger.info('Saved figure to %s', path)
        except Exception as e:
            logger.warning('Failed to save figure %s: %s', path, e)
        finally:
            plt.close('all')

    def _save_current_fig(self, fig, path: str):
        try:
                                                fig.savefig(path,    dpi=plt.rcParams.get('savefig.dpi',    300),
bbox_inches=plt.rcParams.get('savefig.bbox', 'tight'))
            logger.info('Saved figure to %s', path)
        except Exception as e:
            logger.warning('Failed to save figure %s: %s', path, e)
        finally:
            try:
                plt.close(fig)
            except Exception:
                plt.close('all')

    def _apply_plot_style(self):
        """
        应用统一的绘图样式：配色、线宽、网格和中文字体等。
        在所有绘图函数开头调用以保证风格一致。
        """
        try:
            # 尝试设置全局字体为中文字体
            if CHINESE_FONT is not None:
                try:
                    fname = CHINESE_FONT.get_name()
                    if fname:
                        plt.rcParams['font.family'] = fname
                except Exception:
                    # FontProperties 可能由文件路径构造，跳过全局设置
                    pass

            # 基础样式
            plt.rcParams['figure.figsize'] = (12, 6)
            plt.rcParams['axes.titlesize'] = 14
            plt.rcParams['axes.labelsize'] = 12
```

```
            plt.rcParams['xtick.labelsize'] = 11
            plt.rcParams['ytick.labelsize'] = 11
            plt.rcParams['legend.fontsize'] = 11
            plt.rcParams['lines.linewidth'] = 1.8
            plt.rcParams['lines.markersize'] = 6
            plt.rcParams['grid.linewidth'] = 0.8
            plt.rcParams['grid.alpha'] = 0.28
            plt.rcParams['xtick.direction'] = 'in'
            plt.rcParams['ytick.direction'] = 'in'
            plt.rcParams['xtick.top'] = True
            plt.rcParams['ytick.right'] = True
            plt.rcParams['savefig.dpi'] = 300
            plt.rcParams['savefig.bbox'] = 'tight'
            # 颜色调色板
            self._palette = [
                '#2E86AB',  # 深蓝
                '#F6A01A',  # 明橙
                '#3CA55C',  # 绿色
                '#D7263D',  # 红
                '#6A4C93',  # 紫
                '#8C6239',  # 棕
                '#E76F8A',  # 粉
                '#6C757D',  # 中灰
                '#9AAE00',  # 黄绿
                '#17BECF',  # 青
            ]
            try:
                plt.rcParams['axes.prop_cycle'] = plt.cycler(color=self._palette)
            except Exception:
                pass
        except Exception:
            pass

    def _plot_spectrum_with_extrema(self, wavenumber: np.ndarray, reflectance_data: np.ndarray,
                                    fit_result: Dict, incident_angle: float):
        self._apply_plot_style()
        fig, ax = plt.subplots(1, 1)
          palette = getattr(self, '_palette', None) or plt.rcParams.get('axes.prop_cycle').by_key().get('color',
None)
        color_raw = palette[0] if palette else '#1f77b4'
        color_denoised = palette[2] if palette and len(palette) > 2 else '#2ca02c'
        ax.plot(wavenumber, reflectance_data, color=color_raw, alpha=0.6, label='原始光谱')
        if fit_result.get('denoised_reflectance') is not None:
                ax.plot(wavenumber, fit_result['denoised_reflectance'], color=color_denoised, alpha=0.9,
label='去噪后光谱')
        peaks = fit_result['extrema']['peaks']
        valleys = fit_result['extrema']['valleys']
        reflect_interp = np.interp(np.concatenate([peaks, valleys]), wavenumber, reflectance_data)
        marker_peak_color = palette[3] if palette and len(palette) > 3 else '#d62728'
        marker_valley_color = palette[7] if palette and len(palette) > 7 else '#7f7f7f'
                ax.scatter(peaks, reflect_interp[:len(peaks)], marker='o', color=marker_peak_color,
edgecolor='white', label='极大值点', zorder=3)
                ax.scatter(valleys, reflect_interp[len(peaks):], marker='v', color=marker_valley_color,
edgecolor='white', label='极小值点', zorder=3)
        ax.set_ylabel('反射率 (%)', fontproperties=CHINESE_FONT)
```

```
        ax.set_xlabel('波数 (cm$^{-1}$)', fontproperties=CHINESE_FONT)
            ax.set_title(f'光谱与极值点 - {self.material.upper()} (入射角 {incident_angle}°)',
fontproperties=CHINESE_FONT)
        leg = ax.legend(prop=CHINESE_FONT, frameon=True, loc='upper right')
        leg.get_frame().set_alpha(0.95)
        ax.grid(True, linestyle='--', linewidth=0.6)
        plt.tight_layout()
        return fig

    def _plot_interference_orders(self, fit_result: Dict, incident_angle: float):
        peaks = fit_result['extrema']['peaks']
        valleys = fit_result['extrema']['valleys']
        m_peaks_theory = fit_result['m_theory']['peaks']
        m_valleys_theory = fit_result['m_theory']['valleys']
        m_peaks_int = np.round(m_peaks_theory)
        m_valleys_int = np.round(m_valleys_theory - 0.5) + 0.5
        self._apply_plot_style()
        fig, ax = plt.subplots(1, 1)
        palette = getattr(self, '_palette', None)
        p_peak = palette[3] if palette else '#d62728'
        p_valley = palette[7] if palette else '#7f7f7f'
        ax.scatter(peaks, m_peaks_theory, marker='o', color=p_peak, label='理论干涉级数（极大值）')
        ax.plot(peaks, m_peaks_int, linestyle='--', color=p_peak, alpha=0.8, label='最近整数')
        ax.scatter(valleys, m_valleys_theory, marker='v', color=p_valley, label='理论干涉级数（极小值）
')
        ax.plot(valleys, m_valleys_int, linestyle=':', color=p_valley, alpha=0.8, label='最近半整数')
        ax.set_xlabel('波数 (cm$^{-1}$)', fontproperties=CHINESE_FONT)
        ax.set_ylabel('干涉级数 m', fontproperties=CHINESE_FONT)
            ax.set_title(f'干涉级数拟合结果（RMSE_m = {fit_result["rmse_m"]:.4f}）',
fontproperties=CHINESE_FONT)
        leg = ax.legend(prop=CHINESE_FONT, loc='best', frameon=True)
        leg.get_frame().set_alpha(0.95)
        ax.grid(True, linestyle='--', linewidth=0.6)
        plt.tight_layout()
        return fig

    def _plot_refractive_index(self, wavenumber: np.ndarray, fit_result: Dict):
        params = fit_result['params']
        n_complex = self.calculate_refractive_index(wavenumber, params['N_epi'], params['gamma_epi'])
        n_real = n_complex.real
        n_imag = n_complex.imag
        self._apply_plot_style()
        fig, ax = plt.subplots(1, 1)
        palette = getattr(self, '_palette', None)
        color_n = palette[0] if palette else '#1f77b4'
        color_k = palette[3] if palette else '#d62728'
        ax.plot(wavenumber, n_real, color=color_n, label='n（实部）')
        ax.plot(wavenumber, n_imag, color=color_k, label='k（虚部）')
        ax.set_xlabel('波数 (cm$^{-1}$)', fontproperties=CHINESE_FONT)
        ax.set_ylabel('折射率', fontproperties=CHINESE_FONT)
        ax.set_title('拟合折射率 n 与 k', fontproperties=CHINESE_FONT)
        leg = ax.legend(prop=CHINESE_FONT, loc='best', frameon=True)
        leg.get_frame().set_alpha(0.95)
        ax.grid(True, linestyle='--', linewidth=0.6)
        plt.tight_layout()
        return fig
```

```
    def _plot_residuals_histogram(self, fit_result: Dict):
        peaks = fit_result['extrema']['peaks']
        valleys = fit_result['extrema']['valleys']
        m_peaks = fit_result['m_theory']['peaks']
        m_valleys = fit_result['m_theory']['valleys']
        peak_residuals = m_peaks - np.round(m_peaks)
        valley_residuals = m_valleys - (np.round(m_valleys - 0.5) + 0.5)
        residuals = np.concatenate([peak_residuals, valley_residuals])
        self._apply_plot_style()
        fig, ax = plt.subplots(1, 1, figsize=(8, 4))
        ax.hist(residuals, bins=20, color='#7f7f7f', edgecolor='white')
        ax.set_xlabel('干涉级数残差', fontproperties=CHINESE_FONT)
        ax.set_ylabel('计数', fontproperties=CHINESE_FONT)
        ax.set_title('干涉级数残差直方图', fontproperties=CHINESE_FONT)
        ax.grid(True, linestyle='--', linewidth=0.6)
        plt.tight_layout()
        return fig

    def _plot_residuals_vs_index(self, fit_result: Dict):
        """
        绘制残差随样本索引的分布（便于检查系统性趋势）
        """
        peaks = fit_result['extrema']['peaks']
        valleys = fit_result['extrema']['valleys']
        m_peaks = fit_result['m_theory']['peaks']
        m_valleys = fit_result['m_theory']['valleys']
        res_peaks = m_peaks - np.round(m_peaks)
        res_valleys = m_valleys - (np.round(m_valleys - 0.5) + 0.5)
        self._apply_plot_style()
        fig, ax = plt.subplots(1, 1)
        palette = getattr(self, '_palette', None)
        p_peak = palette[3] if palette else '#d62728'
        p_valley = palette[7] if palette else '#7f7f7f'
        if len(res_peaks) > 0:
            ax.scatter(np.arange(len(res_peaks)), res_peaks, marker='o', color=p_peak, label='极大值残差')
        if len(res_valleys) > 0:
            ax.scatter(np.arange(len(res_peaks), len(res_peaks) + len(res_valleys)), res_valleys, marker='v',
color=p_valley, label='极小值残差')
        ax.axhline(0, color='gray', linestyle='--')
        ax.set_xlabel('样本索引', fontproperties=CHINESE_FONT)
        ax.set_ylabel('干涉级数残差', fontproperties=CHINESE_FONT)
        ax.set_title('残差随样本索引分布', fontproperties=CHINESE_FONT)
        leg = ax.legend(prop=CHINESE_FONT, frameon=True)
        leg.get_frame().set_alpha(0.95)
        ax.grid(True, linestyle='--', linewidth=0.6)
        plt.tight_layout()
        return fig

    def _plot_optimization_history(self, fit_result: Dict):
        """
        可视化差分进化的优化历史（cost 随迭代变化，以及参数轨迹的投影）。
        要求 fit_result 包含 'de_history'。
        """
        de_hist = fit_result.get('de_history')
        if not de_hist or len(de_hist.get('cost', [])) == 0:
```

```
            logger.info('没有差分进化历史可用以绘图。')
            return

        costs = np.array(de_hist['cost'], dtype=float)
        xs = np.array(de_hist.get('x', []))

        if xs.size != 0 and xs.ndim == 2 and xs.shape[1] >= 2:
            fig, (ax0, ax1) = plt.subplots(2, 1, figsize=(10, 8), gridspec_kw={'height_ratios': [1, 1]})
            ax0.plot(np.arange(len(costs)), costs, '-o', markersize=4)
            ax0.set_yscale('log')
            ax0.set_xlabel('回调次数', fontproperties=CHINESE_FONT)
            ax0.set_ylabel('目标函数  cost (log scale)', fontproperties=CHINESE_FONT)
            ax0.set_title('差分进化优化历史', fontproperties=CHINESE_FONT)
            ax0.grid(True, alpha=0.2)

            ax1.plot(xs[:, 0], xs[:, 1], '-o', markersize=3)
            ax1.set_xlabel('param 0', fontproperties=CHINESE_FONT)
            ax1.set_ylabel('param 1', fontproperties=CHINESE_FONT)
            ax1.set_title('参数轨迹（前两参数）', fontproperties=CHINESE_FONT)
            ax1.grid(True, alpha=0.2)
            plt.tight_layout()
            return fig
        else:
            fig, ax = plt.subplots(1, 1, figsize=(10, 4))
            ax.plot(np.arange(len(costs)), costs, '-o', markersize=4)
            ax.set_yscale('log')
            ax.set_xlabel('回调次数', fontproperties=CHINESE_FONT)
            ax.set_ylabel('目标函数  cost (log scale)', fontproperties=CHINESE_FONT)
            ax.set_title('差分进化优化历史', fontproperties=CHINESE_FONT)
            ax.grid(True, alpha=0.2)
            plt.tight_layout()
            return fig

    def _plot_combined_summary(self, wavenumber: np.ndarray, reflectance_data: np.ndarray,
                               fit_result: Dict, incident_angle: float, save_path: str):
        """Create a combined summary figure with small panels and save to path."""
        params = fit_result['params']
        peaks = fit_result['extrema']['peaks']
        valleys = fit_result['extrema']['valleys']
        m_peaks_theory = fit_result['m_theory']['peaks']
        m_valleys_theory = fit_result['m_theory']['valleys']
        n_complex = self.calculate_refractive_index(wavenumber, params['N_epi'], params['gamma_epi'])
        n_real = n_complex.real
        n_imag = n_complex.imag

        fig, axes = plt.subplots(2, 2, figsize=(14, 10))
        ax0 = axes[0, 0]
        ax0.plot(wavenumber, reflectance_data, 'b-', alpha=0.6)
        if fit_result.get('denoised_reflectance') is not None:
            ax0.plot(wavenumber, fit_result['denoised_reflectance'], 'g-', alpha=0.8)
        ax0.plot(peaks, np.interp(peaks, wavenumber, reflectance_data), 'ro')
        ax0.plot(valleys, np.interp(valleys, wavenumber, reflectance_data), 'kv')
        ax0.set_title('光谱与极值点', fontproperties=CHINESE_FONT)

        ax1 = axes[0, 1]
        ax1.plot(peaks, m_peaks_theory, 'ro')
```

```
        ax1.plot(peaks, np.round(m_peaks_theory), 'r--')
        ax1.plot(valleys, m_valleys_theory, 'kv')
        ax1.plot(valleys, np.round(m_valleys_theory - 0.5) + 0.5, 'k:')
        ax1.set_title('干涉级数', fontproperties=CHINESE_FONT)

        ax2 = axes[1, 0]
        ax2.plot(wavenumber, n_real, label='n')
        ax2.plot(wavenumber, n_imag, label='k')
        ax2.set_title('折射率  n,k', fontproperties=CHINESE_FONT)
        ax2.legend(prop=CHINESE_FONT)

        ax3 = axes[1, 1]
        peak_residuals = m_peaks_theory - np.round(m_peaks_theory)
        valley_residuals = m_valleys_theory - (np.round(m_valleys_theory - 0.5) + 0.5)
        residuals = np.concatenate([peak_residuals, valley_residuals])
        ax3.hist(residuals, bins=20, color='gray', edgecolor='black')
        ax3.set_title('残差直方图', fontproperties=CHINESE_FONT)

                                    plt.suptitle(f'  拟  合  摘  要  -  {self.material.upper()}
(θ={incident_angle}°)  RMSE_m={fit_result["rmse_m"]:.4f}', fontproperties=CHINESE_FONT)
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        try:
            fig.savefig(save_path, dpi=200)
            logger.info('已保存摘要图到 %s', save_path)
        except Exception as e:
            logger.warning('保存摘要图失败: %s', e)
        plt.show()

    def _plot_refractive_index_surface(self, wavenumber: np.ndarray, fit_result: Dict,
                                       wavenumber_grid_points: int = 80, carrier_grid_points: int = 80,
                                       save_path: Optional[str] = None):
        """
        绘制三维折射率（实部/虚部）随波数与载流子浓度变化的表面图。
        提供可视化色散和吸收对薄膜光学性质的影响。
        """
        try:
            from mpl_toolkits.mplot3d import Axes3D  # noqa: F401
        except Exception:
            logger.warning('mpl_toolkits.mplot3d 不可用，跳过三维可视化。')
            return

        self._apply_plot_style()
        wn_min, wn_max = float(np.min(wavenumber)), float(np.max(wavenumber))
        wn_grid = np.linspace(wn_min, wn_max, wavenumber_grid_points)
        N_fit = fit_result['params'].get('N_epi', 1e16)
        N_min, N_max = max(1e14, N_fit / 100.0), min(1e18, N_fit * 100.0)
        N_grid = np.logspace(np.log10(N_min), np.log10(N_max), carrier_grid_points)
        WN, NGRID = np.meshgrid(wn_grid, N_grid)
        gamma_fit = fit_result['params'].get('gamma_epi', 1e13)
        rn_flat = self.calculate_refractive_index(WN.ravel(), NGRID.ravel(), gamma_fit).real
        rn = rn_flat.reshape(NGRID.shape)

        # plot surface (实部)
        fig = plt.figure(figsize=(14, 10))
        ax = fig.add_subplot(111, projection='3d')
        surf = ax.plot_surface(WN, np.log10(NGRID), rn, cmap='viridis', edgecolor='none', alpha=0.9)
```

```
        # 为3D图设置适中的中文字体
        chinese_font_small = None
        if CHINESE_FONT is not None:
            try:
                                chinese_font_small = FontProperties(fname=CHINESE_FONT.get_name() if
hasattr(CHINESE_FONT, 'get_name') else None, size=12)
                if chinese_font_small.get_name() is None and hasattr(CHINESE_FONT, '_fname'):
                    chinese_font_small = FontProperties(fname=CHINESE_FONT._fname, size=12)
            except Exception:
                chinese_font_small = FontProperties(size=12)
        else:
            chinese_font_small = FontProperties(size=12)

        ax.set_xlabel('波数 (cm$^{-1}$)', fontproperties=chinese_font_small)
                ax.set_ylabel('载流子浓度 N (cm$^{-3}$) — 以 $\log_{10}$ 显示',
fontproperties=chinese_font_small)
        ax.set_zlabel('n (实部)', fontproperties=chinese_font_small)
          ax.set_title('折射率实部随波数与载流子浓度的表面', fontproperties=chinese_font_small,
pad=20)
        fig.colorbar(surf, ax=ax, shrink=0.6, aspect=30)

        try:
            common_powers = np.array([14, 15, 16, 17, 18])
            common_vals = 10.0 ** common_powers
            within = (common_vals >= N_min) & (common_vals <= N_max)
            if np.any(within):
                ytick_vals = common_vals[within]
                ytick_positions = np.log10(ytick_vals)
                ax.set_yticks(ytick_positions)
                ytick_labels = [f'$10^{{{int(p)}}}$' for p in common_powers[within]]
                ax.set_yticklabels(ytick_labels, fontproperties=chinese_font_small)
        except Exception:
            pass
        plt.tight_layout()
        if save_path:
            try:
                fig.savefig(save_path, dpi=200)
                logger.info('已保存三维表面图到 %s', save_path)
            except Exception as e:
                logger.warning('保存三维表面图失败: %s', e)
        plt.show()

def main():
    """主函数"""
    import argparse

    parser = argparse.ArgumentParser(description='极值点迭代优化拟合器')
    parser.add_argument('file', help='CSV 文件路径')
    parser.add_argument('--angle', type=float, default=10.0, help='入射角度 (默认 10°)')
    parser.add_argument('--material', choices=['sic', 'si'], default='sic', help='材料类型')
    parser.add_argument('--plot', action='store_true', help='显示拟合结果图（包含高级图）')
    parser.add_argument('--out-dir', type=str, default=None, help='保存图像的目录，默认为输入文件同
目录下的 figures 文件夹')
    parser.add_argument('--no-advanced', action='store_true', help='在 --plot 时不显示高级图（示意图、
```

```
3D）')
    parser.add_argument('--range', nargs=2, type=float, help='波数范围 (最小值 最大值)')
    parser.add_argument('--no-denoise', action='store_true', help='禁用小波去噪预处理')

    args = parser.parse_args()

    print(f"加载光谱数据: {args.file}")
    wavenumber, reflectance = load_spectrum_data(args.file)

    if args.range:
        mask = (wavenumber >= args.range[0]) & (wavenumber <= args.range[1])
        wavenumber = wavenumber[mask]
        reflectance = reflectance[mask]
        print(f"应用波数范围过滤: {args.range[0]}-{args.range[1]} cm$^{-1}$")

    fitter = IterativeExtremumFitter(material=args.material)

    use_denoise = not args.no_denoise
    result = fitter.fit_spectrum(
        wavenumber, reflectance, args.angle,
        use_denoise=use_denoise
    )

    if not result['success']:
        print(f"警告: 拟合失败或未收敛 ({result.get('message', '')})")

    if args.plot and result['success']:
        out_dir = args.out_dir if args.out_dir else os.path.join(os.path.dirname(args.file) or '.', 'figures')
        os.makedirs(out_dir, exist_ok=True)
        combined_path = os.path.join(out_dir, 'combined_summary.png')
            fitter.plot_fit_result(wavenumber, reflectance, result, args.angle, save_path=combined_path,
save_dir=out_dir)

        if not args.no_advanced:
            try:
                fitter._plot_refractive_index_surface(wavenumber, result, save_path=os.path.join(out_dir,
'n_surface.png'))
            except Exception as e:
                logger.warning('绘制/保存三维折射率表面失败: %s', e)

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

附录四

介绍：问题二中多角度测量一致性分析的代码（run_multi_angle_analysis.py）

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
多角度测量一致性分析脚本

功能:
1. 对多个不同入射角的光谱数据（如附件 1 和附件 2）分别进行厚度拟合。
2. 收集每个角度的拟合结果（厚度 d, 不确定度 σ_d）。
3. 评估单次测量的内部可靠性（CV）。
```

4. 进行角度间的一致性检验（相对差异、置信区间重叠）。
5. 使用加权平均法合并一致的结果，给出一个更可靠的最终厚度值。
"""

```
import numpy as np
import os
from dataclasses import dataclass
from typing import List, Dict, Any

# 确保 iterative_extremum_fitter.py 在同一目录或 Python 路径下
try:
    from iterative_extremum_fitter import IterativeExtremumFitter, load_spectrum_data
except ImportError:
    print("无法导入 'IterativeExtremumFitter'。")
    exit(1)

@dataclass
class AngleResult:
    """一个数据类，用于存储单次角度拟合的结果"""
    angle: float
    filepath: str
    thickness: float
    uncertainty: float
    fit_result: Dict[str, Any]  # 存储完整的拟合结果以备后续分析

def analyze_consistency(results: List[AngleResult]):
    """
    分析来自多个角度的厚度测量结果的一致性。

    Args:
        results: AngleResult 对象的列表。
    """
    if len(results) < 2:
        print("需要至少两个角度的测量结果才能进行一致性分析。")
        return

    print("\n" + "="*60)
    print("  多角度测量一致性与合并结果分析")
    print("="*60)

    # 次测量内部可靠性 使用变异系数(CV)评估
    print("\n--- 1. 单次测量内部可靠性评估 ---")
    all_reliable = True
    for res in results:
        cv = (res.uncertainty / res.thickness) * 100 if res.thickness > 0 else float('inf')
        if cv < 5:
            reliability = "高"
        elif cv < 10:
            reliability = "中等"
        else:
            reliability = "低"

        print(f"角度 {res.angle}°: d = {res.thickness:.3f} ± {res.uncertainty:.3f} μm (CV = {cv:.2f}%, 可
靠性: {reliability})")
        if reliability == "低":
```

```
            all_reliable = False

    if not all_reliable:
        print("\n 警告：存在低可靠性的单次测量，最终合并结果的置信度可能下降。")

    # 将主要对前两个结果进行配对比较
    res1, res2 = results[0], results[1]
    d1, sigma1 = res1.thickness, res1.uncertainty
    d2, sigma2 = res2.thickness, res2.uncertainty

    # 角度间一致性检验
    print("\n--- 2. 角度间一致性检验 (比较前两个结果) ---")

    # 相对差异
    relative_diff = np.abs(d1 - d2) / ((d1 + d2) / 2) * 100
    print(f"相对差异 (|d1-d2|/mean(d1,d2)): {relative_diff:.2f}%")
    if relative_diff > 5.0:
        print("   -> 警告：不同角度测量的厚度相对差异较大 (>5%)，结果可能不一致。")
    else:
        print("   -> 信息：厚度测量值具有良好的一致性。")

    # 置信区间重叠检验
    ci1 = (d1 - sigma1, d1 + sigma1)
    ci2 = (d2 - sigma2, d2 + sigma2)
    overlap = max(ci1[0], ci2[0]) < min(ci1[1], ci2[1])
    print(f"1-sigma 置信区间重叠: {'是' if overlap else '否'}")
    if not overlap:
        print("   -> 警告：两个测量结果的 1-sigma 置信区间不重叠，可能存在系统偏差。")

    # 最终推荐值 (加权平均)
    print("\n--- 3. 最终推荐值 (基于所有高/中等可靠性结果) ---")

    # 筛选出可靠的结果用于最终计算
    reliable_results = [r for r in results if (r.uncertainty / r.thickness * 100) < 10]

    if len(reliable_results) < 1:
        print("没有可靠的测量结果可用于计算最终值。")
        return

    weights = np.array([1 / (res.uncertainty**2) for res in reliable_results])
    thicknesses = np.array([res.thickness for res in reliable_results])

    # 加权平均厚度
    d_final = np.sum(weights * thicknesses) / np.sum(weights)

    # 合并不确定度
    sigma_final = np.sqrt(1 / np.sum(weights))

    # 最终变异系数 (CV)
    cv_final = (sigma_final / d_final) * 100
    if cv_final < 5:
        final_reliability = "高"
    elif cv_final < 10:
        final_reliability = "中等"
    else:
```

```
        final_reliability = "低"

    print(f"最终厚度 (加权平均): {d_final:.4f} ± {sigma_final:.4f} μm")
    print(f"最终变异系数 (CV): {cv_final:.2f}%")
    print(f"最终结果可靠性: {final_reliability}")
    print("="*60)

def main():
    """主执行函数"""
    # 定义待分析的数据文件和对应的入射角
    measurements = [
        {'file': os.path.join('data', '附件 1.csv'), 'angle': 10.0},
        {'file': os.path.join('data', '附件 2.csv'), 'angle': 15.0},
    ]

    fitter = IterativeExtremumFitter(material='sic')
    all_results: List[AngleResult] = []

    for measurement in measurements:
        filepath = measurement['file']
        angle = measurement['angle']

        if not os.path.exists(filepath):
            print(f"错误: 文件不存在 {filepath}，跳过该测量。")
            continue

        print(f"\n>>> 正在处理文件: {filepath} @ {angle}° <<<")

        wavenumber, reflectance = load_spectrum_data(filepath)

        result = fitter.fit_spectrum(
            wavenumber, reflectance, angle, use_denoise=True
        )

        if result.get('success'):
            d = result['params']['d']
            sigma_d = result['param_uncertainties'].get('d', float('nan'))

            # 检查不确定度是否有效
            if np.isnan(sigma_d):
                print(f"警告: 角度 {angle}° 的不确定度计算失败，该结果将不用于最终合并。")
                continue

            all_results.append(AngleResult(
                angle=angle,
                filepath=filepath,
                thickness=d,
                uncertainty=sigma_d,
                fit_result=result
            ))
            print(f"拟合成功: d = {d:.3f} ± {sigma_d:.3f} μm")
        else:
            print(f"拟合失败: {result.get('message', '未知错误')}")

    # 当所有拟合都完成后，进行一致性分析
    if len(all_results) > 1:
```

```
        analyze_consistency(all_results)
    else:
        print("\n 未能成功完成足够的拟合来进行一致性分析。")

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

附录五

介绍：问题三的基于全谱拟合的 SiC 外延层厚度拟合算法的代码（full_spectrum_thickness.py）

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
基于全谱拟合的 SiC 外延层厚度

物理原理：
- 小波阈值去噪预处理提高数据质量
- 使用标准三项 Sellmeier 公式计算束缚电子对折射率的贡献
- 结合 Drude 模型计算自由载流子对复折射率的影响
- 通过传输矩阵法计算单层薄膜的反射光谱
- 使用非线性拟合同时确定外延层厚度 d 和载流子浓度 N
"""

import argparse
import numpy as np
import pandas as pd
from scipy.optimize import least_squares, differential_evolution
from scipy.stats import bootstrap
import matplotlib.pyplot as plt
import matplotlib as mpl
try:
    import seaborn as sns
    SEABORN_AVAILABLE = True
except Exception:
    SEABORN_AVAILABLE = False

# 字体解决方案
from matplotlib.font_manager import FontProperties
import os

def find_chinese_font():
    """
    在系统中查找可用的中文字体文件，并返回一个 FontProperties 对象。
    """
    if os.name == 'nt':
        font_paths = [
            'C:/Windows/Fonts/msyh.ttc',
            'C:/Windows/Fonts/simhei.ttf',
            'C:/Windows/Fonts/simsun.ttc',
        ]
        for font_path in font_paths:
            if os.path.exists(font_path):
                print(f"找到系统字体: {font_path}")
                return FontProperties(fname=font_path, size=12)
    try:
        font_names = ['Microsoft YaHei', 'SimHei', 'SimSun', 'KaiTi', 'FangSong']
```

```
            for font_name in font_names:
                try:
                    return FontProperties(font_name, size=12)
                except Exception:
                    continue
        except Exception:
            pass
        print("未能在系统中找到任何指定的中文字体，图形中的中文可能无法显示。")
        return None

# 获取中文字体
CHINESE_FONT = find_chinese_font()

# 基础 matplotlib 配置 - 修复数学表达式负号显示问题
def configure_matplotlib_fonts():
    """配置 matplotlib 字体，确保中文和数学表达式都能正确显示"""
    # 设置数学文本字体为系统默认，确保负号等符号正确显示
    mpl.rcParams['mathtext.fontset'] = 'dejavusans'
    mpl.rcParams['mathtext.default'] = 'regular'

    # 确保负号正确显示
    mpl.rcParams['axes.unicode_minus'] = False

    # 设置全局字体大小
    mpl.rcParams['font.size'] = 11
    mpl.rcParams['axes.titlesize'] = 13
    mpl.rcParams['axes.labelsize'] = 12
    mpl.rcParams['xtick.labelsize'] = 10
    mpl.rcParams['ytick.labelsize'] = 10
    mpl.rcParams['legend.fontsize'] = 11

    # 提高图形质量
    mpl.rcParams['figure.dpi'] = 100
    mpl.rcParams['savefig.dpi'] = 300
    mpl.rcParams['figure.facecolor'] = 'white'
    mpl.rcParams['axes.facecolor'] = 'white'

    # 设置更美观的默认样式
    mpl.rcParams['axes.linewidth'] = 0.8
    mpl.rcParams['axes.grid'] = True
    mpl.rcParams['grid.alpha'] = 0.3
    mpl.rcParams['grid.linewidth'] = 0.5

    # 改善科学记数法显示
    mpl.rcParams['axes.formatter.use_mathtext'] = True
    mpl.rcParams['axes.formatter.useoffset'] = False

# 应用字体配置
configure_matplotlib_fonts()
import warnings
from dataclasses import dataclass
from typing import Dict, Tuple, Optional, List
import time

# 小波去噪相关库
try:
```

```
    import pywt
    PYWT_AVAILABLE = True
except ImportError:
    PYWT_AVAILABLE = False
    warnings.warn("PyWavelets 库未安装，将跳过小波去噪功能")

@dataclass
class MaterialParams:
    """材料参数数据类"""
    # 标准三项 Sellmeier 系数（束缚电子贡献）
    # n²(λ) = 1 + Σ(A_i * λ² / (λ² - λ_i²))
    A1: float  # 第一项系数
    A2: float  # 第二项系数
    A3: float  # 第三项系数
    lambda1_sq: float  # 第一项特征波长的平方（μm²）
    lambda2_sq: float  # 第二项特征波长的平方（μm²）
    lambda3_sq: float  # 第三项特征波长的平方（μm²）
    # 物理常数
    m_eff: float  # 有效质量（单位：m_e）
    eps_inf: float  # 高频介电常数

class FullSpectrumAnalyzer:
    """
    全谱拟合 SiC 外延层分析器

    该类实现了基于物理模型的厚度和载流子浓度联合测量，包括：
    - 小波阈值去噪预处理
    - 标准三项 Sellmeier 公式计算束缚电子折射率贡献
    - Drude 模型计算自由载流子对复介电函数的影响
    - 传输矩阵法计算反射率
    - 非线性优化拟合
    - 多角度数据联合分析
    """

    # 物理常数
    C_LIGHT = 2.99792458e10  # 光速 cm/s
    E_CHARGE = 1.602176634e-19  # 电子电荷 C
    EPS_0 = 8.854187817e-12  # 真空介电常数 F/m
    M_ELECTRON = 9.10938356e-31  # 电子质量 kg

    # 材料参数库
    MATERIALS = {
        'sic': MaterialParams(
            # SiC 的三项 Sellmeier 系数
            A1=0.20075, A2=5.54861, A3=35.65066,
            lambda1_sq=-12.07224, lambda2_sq=0.02641, lambda3_sq=1268.24708,
            m_eff=0.5, eps_inf=6.5
        ),
        'si': MaterialParams(
            # Si 的三项 Sellmeier 系数
            A1=10.6684293, A2=0.0030434748, A3=1.54133408,
            lambda1_sq=0.09091243, lambda2_sq=1.28764888, lambda3_sq=1218816.0,
            m_eff=0.26, eps_inf=11.7
        )
```

```
}

def __init__(self, material='sic'):
    """
    初始化分析器

    Parameters:
    -----------
    material : str
        材料类型，'sic' 或 'si'
    """
    if material not in self.MATERIALS:
        raise ValueError(f"不支持的材料类型: {material}")

    self.material_params = self.MATERIALS[material]
    self.material = material

    # 小波去噪参数
    self.wavelet_params = {
        'wavelet': 'db4',  # Daubechies 小波
        'mode': 'symmetric',  # 边界处理模式
        'threshold_mode': 'soft'  # 软阈值
    }

def calculate_complex_permittivity(self, wavenumber: np.ndarray,
                                   carrier_density: float,
                                   scattering_rate: float) -> np.ndarray:
    """
    计算 Drude-Sellmeier 复介电函数

    物理模型：
    ε(ω) = ε_bound(ω) + ε_drude(ω)

    其中：
    - ε_bound = n²(λ) = 1 + Σ(A_i*λ²/(λ²-λ_i²))  [三项 Sellmeier 公式，束缚电子]
    - ε_drude = -ωp²/(ω² + iγω)  [Drude 项，自由载流子]
    - ωp² = Ne²/(εωm*)           [等离子体频率]

    Parameters:
    -----------
    wavenumber : array_like
        波数，单位 cm⁻¹
    carrier_density : float
        载流子浓度，单位 cm⁻³
    scattering_rate : float
        散射率，单位 s⁻¹

    Returns:
    --------
    epsilon : ndarray (complex)
        复介电函数
    """
    # Sellmeier 项（束缚电子贡献）
    # 波数转换为波长：λ(μm) = 10000 / ν(cm⁻¹)
    wavelength = 10000.0 / wavenumber  # μm
```

```
    wavelength_sq = wavelength**2

    # 三项Sellmeier 公式：n²(λ) = 1 + Σ(A_i*λ²/(λ²-λ_i²))
    params = self.material_params

    # 第一项
    denom1 = wavelength_sq - params.lambda1_sq
    denom1 = np.where(np.abs(denom1) < 1e-12, 1e-12, denom1)
    term1 = params.A1 * wavelength_sq / denom1

    # 第二项
    denom2 = wavelength_sq - params.lambda2_sq
    denom2 = np.where(np.abs(denom2) < 1e-12, 1e-12, denom2)
    term2 = params.A2 * wavelength_sq / denom2

    # 第三项
    denom3 = wavelength_sq - params.lambda3_sq
    denom3 = np.where(np.abs(denom3) < 1e-12, 1e-12, denom3)
    term3 = params.A3 * wavelength_sq / denom3

    # 总的折射率平方（即介电常数的束缚电子部分）
    epsilon_bound = 1.0 + term1 + term2 + term3

    # 如果载流子浓度为零，仅返回束缚电子项
    if carrier_density <= 0:
        return epsilon_bound.astype(complex)

    # Drude 项（自由载流子贡献）
    # 波数转角频率：ω = 2πcν
    omega = 2.0 * np.pi * self.C_LIGHT * wavenumber

    # 等离子体频率（SI 单位）
    N_m3 = carrier_density * 1e6   # cm⁻³ → m⁻³
    m_star = self.material_params.m_eff * self.M_ELECTRON
    omega_p_squared = (N_m3 * self.E_CHARGE**2) / (self.EPS_0 * m_star)

    # Drude 项：ε_D = -ωp²/(ω² + iγω)
    denom_drude = omega**2 + 1j * scattering_rate * omega
    epsilon_drude = -omega_p_squared / denom_drude

    # 总介电函数
    epsilon_total = epsilon_bound + epsilon_drude

    return epsilon_total.astype(complex)

def calculate_refractive_index(self, wavenumber: np.ndarray,
                               carrier_density: float,
                               scattering_rate: float) -> np.ndarray:
    """
    计算复折射率 ñ = n + ik

    Parameters:
    -----------
    wavenumber : array_like
        波数，单位 cm⁻¹
    carrier_density : float
```

```
        载流子浓度，单位 cm⁻³
    scattering_rate : float
        散射率，单位 s⁻¹

    Returns:
    --------
    n_complex : ndarray (complex)
        复折射率
    """
    epsilon = self.calculate_complex_permittivity(wavenumber, carrier_density, scattering_rate)
    return np.sqrt(epsilon)

def wavelet_denoise(self, reflectance: np.ndarray,
                    wavelet: Optional[str] = None,
                    sigma: Optional[float] = None) -> np.ndarray:
    """
    小波阈值去噪处理

    Parameters:
    -----------
    reflectance : array_like
        反射率数据
    wavelet : str, optional
        小波类型，默认使用'db4'
    sigma : float, optional
        噪声标准差，如果为 None 则自动估计

    Returns:
    --------
    denoised_reflectance : ndarray
        去噪后的反射率数据
    """
    if not PYWT_AVAILABLE:
        warnings.warn("PyWavelets 库不可用，跳过小波去噪")
        return reflectance.copy()

    try:
        # 使用默认参数或传入参数
        wavelet_type = wavelet or self.wavelet_params['wavelet']
        mode = self.wavelet_params['mode']
        threshold_mode = self.wavelet_params['threshold_mode']

        # 小波分解
        coeffs = pywt.wavedec(reflectance, wavelet_type, mode=mode)

        # 估计噪声标准差
        if sigma is None:
            # 使用最高频细节系数估计噪声
            sigma = np.median(np.abs(coeffs[-1])) / 0.6745

        # 计算阈值（软阈值）
        threshold = sigma * np.sqrt(2 * np.log(len(reflectance)))

        # 对细节系数进行阈值处理
        coeffs_thresh = list(coeffs)
        coeffs_thresh[1:] = [pywt.threshold(detail, threshold, mode=threshold_mode)
```

```
                                for detail in coeffs[1:]]

            # 小波重构
            denoised = pywt.waverec(coeffs_thresh, wavelet_type, mode=mode)

            # 确保输出长度与输入相同
            if len(denoised) != len(reflectance):
                denoised = denoised[:len(reflectance)]

            return denoised

        except Exception as e:
            warnings.warn(f"小波去噪失败: {e}，返回原始数据")
            return reflectance.copy()


    def transfer_matrix_single_layer(self, wavenumber: np.ndarray,
                                     thickness: float,
                                     n_film: np.ndarray,
                                     n_substrate: np.ndarray,
                                     incident_angle: float,
                                     n_ambient: float = 1.0) -> np.ndarray:
        """
        使用传输矩阵法计算单层薄膜的反射率

        物理原理：
        - 基于 Maxwell 方程和边界条件
        - 考虑薄膜内的多次反射
        - 适用于任意复折射率

        Parameters:
        -----------
        wavenumber : array_like
            波数，单位 cm⁻¹
        thickness : float
            薄膜厚度，单位 μm
        n_film : array_like (complex)
            薄膜复折射率
        n_substrate : array_like (complex)
            衬底复折射率
        incident_angle : float
            入射角，单位度
        n_ambient : float
            环境介质折射率（默认空气，n=1）

        Returns:
        --------
        reflectance : ndarray
            反射率（实数，0-1）
        """
        # 角度转弧度
        theta0 = np.radians(incident_angle)

        # 波长（cm）
        wavelength = 1.0 / wavenumber
```

```
    # 厚度转换：μm → cm
    d_cm = thickness * 1e-4

    # Snell 定律计算各层中的折射角
    sin_theta0 = np.sin(theta0)

    # 薄膜中的折射角（复数）
    sin_theta1 = n_ambient * sin_theta0 / n_film
    cos_theta1 = np.sqrt(1.0 - sin_theta1**2)

    # 衬底中的折射角
    sin_theta2 = n_ambient * sin_theta0 / n_substrate
    cos_theta2 = np.sqrt(1.0 - sin_theta2**2)

    # 菲涅尔系数：环境-薄膜界面
    r01 = (n_ambient * np.cos(theta0) - n_film * cos_theta1) / \
          (n_ambient * np.cos(theta0) + n_film * cos_theta1)
    t01 = 2 * n_ambient * np.cos(theta0) / \
          (n_ambient * np.cos(theta0) + n_film * cos_theta1)

    # 菲涅尔系数：薄膜-衬底界面
    r12 = (n_film * cos_theta1 - n_substrate * cos_theta2) / \
          (n_film * cos_theta1 + n_substrate * cos_theta2)
    t12 = 2 * n_film * cos_theta1 / \
          (n_film * cos_theta1 + n_substrate * cos_theta2)

    # 薄膜中的相位厚度
    beta = 2 * np.pi * n_film * d_cm * cos_theta1 / wavelength

    # 薄膜传输矩阵元素
    M11 = np.cos(beta)
    M12 = 1j * np.sin(beta) / (n_film * cos_theta1)
    M21 = 1j * n_film * cos_theta1 * np.sin(beta)
    M22 = np.cos(beta)

    # 总的反射系数
    # 考虑薄膜传输矩阵和界面反射
    Y = n_substrate * cos_theta2  # 衬底导纳

    # 从传输矩阵计算总导纳
    Y_total = (M21 + M22 * Y) / (M11 + M12 * Y)

    # 总反射系数
    r_total = (n_ambient * np.cos(theta0) - Y_total) / \
              (n_ambient * np.cos(theta0) + Y_total)

    # 反射率
    reflectance = np.abs(r_total)**2

    return reflectance

def forward_model(self, wavenumber: np.ndarray, params: Dict,
                  incident_angle: float) -> np.ndarray:
    """
    前向模型：根据参数计算理论反射光谱
```

```
    Parameters:
    -----------
    wavenumber : array_like
        波数，单位 cm⁻¹
    params : dict
        拟合参数字典，包含：
        - 'd': 厚度 (μm)
        - 'N_epi': 外延层载流子浓度 (cm⁻³)
        - 'gamma_epi': 外延层散射率 (s⁻¹)
        - 'N_sub': 衬底载流子浓度 (cm⁻³)
        - 'gamma_sub': 衬底散射率 (s⁻¹)
    incident_angle : float
        入射角度

    Returns:
    --------
    reflectance : ndarray
        理论反射率
    """
    # 计算外延层复折射率
    n_epi = self.calculate_refractive_index(
        wavenumber, params['N_epi'], params['gamma_epi']
    )

    # 计算衬底复折射率
    n_sub = self.calculate_refractive_index(
        wavenumber, params['N_sub'], params['gamma_sub']
    )

    # 使用传输矩阵法计算反射率
    reflectance = self.transfer_matrix_single_layer(
        wavenumber, params['d'], n_epi, n_sub, incident_angle
    )

    return reflectance

def objective_function(self, x: np.ndarray, wavenumber: np.ndarray,
                       reflectance_data: np.ndarray, incident_angle: float,
                       param_names: List[str], weights: Optional[np.ndarray] = None) -> np.ndarray:
    """
    目标函数：计算模型与数据的残差

    Parameters:
    -----------
    x : array_like
        待拟合参数数组
    wavenumber : array_like
        波数数据
    reflectance_data : array_like
        实测反射率数据
    incident_angle : float
        入射角度
    param_names : list
        参数名称列表
```

```
    weights : array_like, optional
        数据点权重

    Returns:
    --------
    residuals : ndarray
        残差数组
    """
    # 构建参数字典
    params = dict(zip(param_names, x))

    try:
        # 计算理论光谱
        model_reflectance = self.forward_model(wavenumber, params, incident_angle)
        # 计算残差
        residuals = model_reflectance - reflectance_data
        # 应用权重
        if weights is not None:
            residuals *= weights

        return residuals

    except Exception as e:
        warnings.warn(f"前向模型计算失败: {e}")
        return np.full_like(reflectance_data, 1e6)

def fit_spectrum(self, wavenumber: np.ndarray, reflectance: np.ndarray,
                 incident_angle: float, initial_guess: Optional[Dict] = None,
                 bounds: Optional[Dict] = None, method: str = 'lm',
                 use_wavelet_denoise: bool = True) -> Dict:
    """
    拟合单个角度的反射光谱

    Parameters:
    -----------
    wavenumber : array_like
        波数，单位 cm⁻¹
    reflectance : array_like
        反射率，单位 %
    incident_angle : float
        入射角，单位度
    initial_guess : dict, optional
        初始参数猜测
    bounds : dict, optional
        参数边界
    method : str
        优化方法：'lm'(Levenberg-Marquardt) 或 'global'(全局优化)
    use_wavelet_denoise : bool
        是否使用小波去噪预处理

    Returns:
    --------
    result : dict
        拟合结果，包含参数值、不确定性、拟合质量等
    """
```

```
# 数据预处理
reflectance_original = reflectance.copy()  # 保存原始数据
reflectance = reflectance / 100.0  # % → 小数

# 小波去噪预处理
reflectance_denoised = reflectance.copy()

if use_wavelet_denoise and PYWT_AVAILABLE:
    print("应用小波阈值去噪...")
    reflectance_denoised = self.wavelet_denoise(reflectance)
    print("  小波去噪处理完成")

    # 使用去噪后的数据进行拟合
    reflectance = reflectance_denoised
elif use_wavelet_denoise and not PYWT_AVAILABLE:
    print("警告: 小波去噪功能不可用，使用原始数据")

# 默认初始猜测
if initial_guess is None:
    initial_guess = {
        'd': 10.0,              # μm
        'N_epi': 1e16,          # cm⁻³
        'gamma_epi': 1e13,      # s⁻¹
        'N_sub': 1e19,          # cm⁻³
        'gamma_sub': 1e14       # s⁻¹
    }

# 默认边界
if bounds is None:
    bounds = {
        'd': (6.0, 15.0),
        'N_epi': (1e14, 1e18),
        'gamma_epi': (1e11, 1e15),
        'N_sub': (1e17, 1e21),
        'gamma_sub': (1e12, 1e16)
    }

# 构建参数数组和边界
param_names = list(initial_guess.keys())
x0 = np.array([initial_guess[name] for name in param_names])
lower_bounds = np.array([bounds[name][0] for name in param_names])
upper_bounds = np.array([bounds[name][1] for name in param_names])

print(f"开始拟合 (入射角 {incident_angle}°)...")
print(f"初始参数: {dict(zip(param_names, x0))}")

start_time = time.time()

if method == 'global':
    # 全局优化（差分进化算法）
    def objective_wrapper(x):
        residuals = self.objective_function(
            x, wavenumber, reflectance, incident_angle, param_names
        )
        return np.sum(residuals**2)
```

```
        opt_result = differential_evolution(
            objective_wrapper,
            bounds=list(zip(lower_bounds, upper_bounds)),
            seed=42,
            maxiter=500,
            popsize=20
        )
        x_opt = opt_result.x
        success = opt_result.success
        cost = opt_result.fun
    else:
        # 局部优化（Levenberg-Marquardt）
        opt_result = least_squares(
            self.objective_function,
            x0,
            args=(wavenumber, reflectance, incident_angle, param_names),
            bounds=(lower_bounds, upper_bounds),
            method='trf',
            max_nfev=2000
        )
        x_opt = opt_result.x
        success = opt_result.success
        cost = opt_result.cost
    elapsed_time = time.time() - start_time
    # 构建结果字典
    fitted_params = dict(zip(param_names, x_opt))
    # 计算拟合质量指标
    model_reflectance = self.forward_model(wavenumber, fitted_params, incident_angle)
    residuals = model_reflectance - reflectance

    rmse = np.sqrt(np.mean(residuals**2))
    mae = np.mean(np.abs(residuals))
    r_squared = 1 - np.var(residuals) / np.var(reflectance)

    result = {
        'success': success,
        'params': fitted_params,
        'cost': cost,
        'rmse': rmse,
        'mae': mae,
        'r_squared': r_squared,
        'residuals': residuals,
        'model_reflectance': model_reflectance,
        'elapsed_time': elapsed_time,
        'method': method,
        'denoising': {
            'applied': use_wavelet_denoise and PYWT_AVAILABLE,
            'original_reflectance': reflectance_original,
            'denoised_reflectance': reflectance_denoised * 100.0 if use_wavelet_denoise else None
        }
    }

    print(f"拟合完成 ({elapsed_time:.1f}s)")
    print(f"成功: {success}")
    print(f"RMSE: {rmse:.6f}")
    print(f"R²: {r_squared:.6f}")
    print(f"拟合参数:")
```

```
    for name, value in fitted_params.items():
        if name == 'd':
            print(f"   {name}: {value:.3f} μm")
        elif 'N_' in name:
            # 使用安全的科学记数法格式化，避免负号显示问题
            exp_str = f"{value:.2e}".replace('e-', 'e⁻').replace('e+', 'e+')
            print(f"   {name}: {exp_str} cm⁻³")
        elif 'gamma_' in name:
            exp_str = f"{value:.2e}".replace('e-', 'e⁻').replace('e+', 'e+')
            print(f"   {name}: {exp_str} s⁻¹")

    return result

def bootstrap_uncertainty(self, wavenumber: np.ndarray, reflectance: np.ndarray,
                          incident_angle: float, fitted_params: Dict,
                          n_bootstrap: int = 100) -> Dict:
    """
    使用 Bootstrap 方法估算参数不确定性

    Parameters:
    -----------
    wavenumber : array_like
        波数数据
    reflectance : array_like
        反射率数据
    incident_angle : float
        入射角度
    fitted_params : dict
        拟合得到的参数
    n_bootstrap : int
        Bootstrap 采样次数

    Returns:
    --------
    uncertainty : dict
        参数不确定性统计
    """
    print(f"开始 Bootstrap 不确定性分析 (N={n_bootstrap})...")

    # 计算残差
    model_reflectance = self.forward_model(wavenumber, fitted_params, incident_angle)
    residuals = reflectance / 100.0 - model_reflectance

    # 残差的标准差（作为噪声水平估计）
    noise_std = np.std(residuals)

    param_names = list(fitted_params.keys())
    bootstrap_params = []

    rng = np.random.default_rng(42)

    for i in range(n_bootstrap):
        # 生成带噪声的数据
        noise = rng.normal(0, noise_std, len(reflectance))
        noisy_reflectance = (model_reflectance + noise) * 100.0
```

```
            try:
                # 拟合带噪声的数据
                result = self.fit_spectrum(
                    wavenumber, noisy_reflectance, incident_angle,
                    initial_guess=fitted_params, method='lm'
                )

                if result['success']:
                    bootstrap_params.append([result['params'][name] for name in param_names])

            except Exception:
                continue

        if len(bootstrap_params) < 10:
            warnings.warn("Bootstrap 样本数量不足，不确定性估计可能不可靠")

        bootstrap_params = np.array(bootstrap_params)

        # 计算统计量
        uncertainty = {}
        for i, name in enumerate(param_names):
            values = bootstrap_params[:, i]
            uncertainty[name] = {
                'mean': np.mean(values),
                'std': np.std(values, ddof=1),
                'ci95': np.percentile(values, [2.5, 97.5])
            }

        print(f"Bootstrap 分析完成 (有效样本: {len(bootstrap_params)}/{n_bootstrap})")

        return uncertainty

def load_spectrum_data(filepath: str) -> Tuple[np.ndarray, np.ndarray]:
    """
    加载光谱数据文件

    Parameters:
    -----------
    filepath : str
        CSV 文件路径

    Returns:
    --------
    wavenumber : ndarray
        波数，单位 cm⁻¹
    reflectance : ndarray
        反射率，单位 %
    """
    df = pd.read_csv(filepath)
    wavenumber = df.iloc[:, 0].values
    reflectance = df.iloc[:, 1].values

    # 过滤 NaN 值
    mask = ~(np.isnan(wavenumber) | np.isnan(reflectance))
    wavenumber = wavenumber[mask]
    reflectance = reflectance[mask]
```

```
    # 按波数排序
    sort_idx = np.argsort(wavenumber)
    wavenumber = wavenumber[sort_idx]
    reflectance = reflectance[sort_idx]

    return wavenumber, reflectance

def main():
    """主函数"""
    parser = argparse.ArgumentParser(description='基于全谱拟合的 SiC 外延层厚度和载流子浓度测量')
    parser.add_argument('file', help='CSV 文件路径')
    parser.add_argument('--angle', type=float, default=10.0, help='入射角度 (默认 10°)')
    parser.add_argument('--material', choices=['sic', 'si'], default='sic', help='材料类型')
    parser.add_argument('--method', choices=['lm', 'global'], default='lm', help='优化方法')
    parser.add_argument('--bootstrap', type=int, default=50, help='Bootstrap 采样次数')
    parser.add_argument('--plot', action='store_true', help='显示拟合结果图')
    parser.add_argument('--range', nargs=2, type=float, help='波数范围 (最小值 最大值)')
    parser.add_argument('--no-denoise', action='store_true', help='禁用小波去噪预处理')
    parser.add_argument('--wavelet', default='db4', help='小波类型 (默认 db4)')

    # 图片保存相关参数
    parser.add_argument('--save-figures', action='store_true', help='保存拟合结果图片')
    parser.add_argument('--output-dir', default='data/figures', help='图片输出目录 (默认 data/figures)')
     parser.add_argument('--figure-format', choices=['png', 'pdf', 'svg', 'jpg'], default='png', help='图片格式
(默认 png)')
    parser.add_argument('--figure-dpi', type=int, default=300, help='图片分辨率 (默认 300)')
    parser.add_argument('--save-data', action='store_true', help='保存拟合数据到 CSV 文件')

    args = parser.parse_args()

    # 加载数据
    print(f"加载光谱数据: {args.file}")
    wavenumber, reflectance = load_spectrum_data(args.file)

    # 应用波数范围过滤
    if args.range:
        mask = (wavenumber >= args.range[0]) & (wavenumber <= args.range[1])
        wavenumber = wavenumber[mask]
        reflectance = reflectance[mask]
        print(f"应用波数范围过滤: {args.range[0]}-{args.range[1]} cm⁻¹")

    print(f"数据点数量: {len(wavenumber)}")
    print(f"波数范围: {wavenumber.min():.1f} - {wavenumber.max():.1f} cm⁻¹")
    print(f"反射率范围: {reflectance.min():.1f} - {reflectance.max():.1f} %")

    # 创建分析器
    analyzer = FullSpectrumAnalyzer(material=args.material)

    # 设置小波参数
    if args.wavelet != 'db4':
        analyzer.wavelet_params['wavelet'] = args.wavelet

    # 执行拟合
```

```
use_denoise = not args.no_denoise
result = analyzer.fit_spectrum(
    wavenumber, reflectance, args.angle,
    method=args.method, use_wavelet_denoise=use_denoise
)

if not result['success']:
    print("警告: 拟合未收敛，结果可能不可靠")

# Bootstrap 不确定性分析
if args.bootstrap > 0:
    # 使用与拟合相同的数据（原始或去噪后）
    bootstrap_reflectance = reflectance
    if result['denoising']['applied']:
        bootstrap_reflectance = result['denoising']['denoised_reflectance']

    uncertainty = analyzer.bootstrap_uncertainty(
        wavenumber, bootstrap_reflectance, args.angle,
        result['params'], n_bootstrap=args.bootstrap
    )

    def safe_format_e(value):
        """将科学记数法中的'e'替换为'×10^'，用于终端输出，避免乱码"""
        return f"{value:.2e}".replace('e', '×10^')

    print("\n===  参数不确定性  (Bootstrap) ===")
    for name, stats in uncertainty.items():
        if name == 'd':
            print(f"{name}: {stats['mean']:.3f} ± {stats['std']:.3f} μm "
                  f"(95% CI: [{stats['ci95'][0]:.3f}, {stats['ci95'][1]:.3f}])")
        elif 'N_' in name:
            mean_str = safe_format_e(stats['mean'])
            std_str = safe_format_e(stats['std'])
            ci_low_str = safe_format_e(stats['ci95'][0])
            ci_high_str = safe_format_e(stats['ci95'][1])
            print(f"{name}: {mean_str} ± {std_str} cm⁻³ "
                  f"(95% CI: [{ci_low_str}, {ci_high_str}])")
        elif 'gamma_' in name:
            mean_str = safe_format_e(stats['mean'])
            std_str = safe_format_e(stats['std'])
            ci_low_str = safe_format_e(stats['ci95'][0])
            ci_high_str = safe_format_e(stats['ci95'][1])
            print(f"{name}: {mean_str} ± {std_str} s⁻¹ "
                  f"(95% CI: [{ci_low_str}, {ci_high_str}])")

# 可视化
if args.plot:
    # 应用美学样式
    if SEABORN_AVAILABLE:
        sns.set_style("whitegrid")
        sns.set_palette("husl")
    else:
        # 使用更现代的样式
        try:
            plt.style.use('seaborn-v0_8-whitegrid')
        except:
```

```
        plt.style.use('default')

# 主要图：反射率与拟合模型，右侧为残差直方图，底部为残差随波数
fig = plt.figure(constrained_layout=False, figsize=(15, 10))
gs = fig.add_gridspec(3, 3, hspace=0.35, wspace=0.3)

ax_main = fig.add_subplot(gs[0:2, 0:2])
ax_residual_line = fig.add_subplot(gs[2, 0:2], sharex=ax_main)
ax_resid_hist = fig.add_subplot(gs[0:2, 2])

# 定义颜色方案
colors = {
    'data': '#2E86AB',       # 深蓝色
    'model': '#A23B72',      # 深红紫色
    'residual': '#F18F01',   # 橙色
    'hist': '#C73E1D'        # 深红色
}

# 主图：数据与模型
ax_main.plot(wavenumber, reflectance, color=colors['data'], linewidth=1.5,
             alpha=0.8, label='实测数据', zorder=2)
ax_main.plot(wavenumber, result['model_reflectance'] * 100, color=colors['model'],
             linewidth=2, alpha=0.9, label=f'拟合模型', zorder=3)

ax_main.set_ylabel('反射率 (%)', fontproperties=CHINESE_FONT, fontsize=12)
title = f'全谱拟合结果 - {args.material.upper()} (θ={args.angle}°)'
if result['denoising']['applied']:
    title += '  [小波去噪]'
ax_main.set_title(title, fontproperties=CHINESE_FONT, fontsize=14, pad=15)
ax_main.legend(frameon=True, prop=CHINESE_FONT, loc='best',
               framealpha=0.9, fancybox=True)
ax_main.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)
ax_main.set_facecolor('#FAFAFA')

# 底部：残差随波数
residuals_pct = result['residuals'] * 100
ax_residual_line.plot(wavenumber, residuals_pct, color=colors['residual'],
                      linewidth=1.5, alpha=0.8)
ax_residual_line.axhline(0, color='k', linestyle='--', alpha=0.7, linewidth=1)
ax_residual_line.set_xlabel('波数 (cm⁻¹)', fontproperties=CHINESE_FONT, fontsize=12)
ax_residual_line.set_ylabel('残差 (%)', fontproperties=CHINESE_FONT, fontsize=12)
ax_residual_line.set_title(f'拟合残差 (RMSE={result["rmse"]:.4f})',
                           fontproperties=CHINESE_FONT, fontsize=12)
ax_residual_line.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)
ax_residual_line.set_facecolor('#FAFAFA')

# 右侧：残差直方图与正态拟合
n, bins, patches = ax_resid_hist.hist(residuals_pct, bins=35, color=colors['hist'],
                                      alpha=0.7, density=True, edgecolor='white', linewidth=0.5)
ax_resid_hist.set_title('残差分布', fontproperties=CHINESE_FONT, fontsize=12)
ax_resid_hist.set_xlabel('残差 (%)', fontproperties=CHINESE_FONT, fontsize=11)
ax_resid_hist.set_ylabel('概率密度', fontproperties=CHINESE_FONT, fontsize=11)
ax_resid_hist.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)
ax_resid_hist.set_facecolor('#FAFAFA')
```

```
        # 添加正态分布拟合曲线
        mu, sigma = np.mean(residuals_pct), np.std(residuals_pct)
        x_norm = np.linspace(residuals_pct.min(), residuals_pct.max(), 100)
        y_norm = (1/(sigma * np.sqrt(2*np.pi))) * np.exp(-0.5*((x_norm - mu)/sigma)**2)
        ax_resid_hist.plot(x_norm, y_norm, color='red', linewidth=2, linestyle='--',
                           alpha=0.8, label='正态拟合')
        ax_resid_hist.legend(prop=CHINESE_FONT, frameon=True, framealpha=0.9)

        # 参数文本框已移除以避免遮挡曲线图

        plt.tight_layout()

        # 保存图形到 data/figures，并分别导出单独面板以便检视
        try:
            import os
            out_dir = os.path.join(os.path.dirname(__file__), 'data', 'figures')
            os.makedirs(out_dir, exist_ok=True)
            base_name = os.path.splitext(os.path.basename(args.file))[0]

            out_path = os.path.join(out_dir, f'{base_name}_fit_summary.png')
            fig.savefig(out_path, dpi=300, bbox_inches='tight')
            print(f"已保存拟合图像: {out_path}")

            # 分别保存各子图，便于单独查看中文/负号问题
            try:
                ax_main.figure = fig
                main_path = os.path.join(out_dir, f'{base_name}_main.png')
                bbox_main = ax_main.get_tightbbox(fig.canvas.get_renderer())
                fig.savefig(main_path, dpi=300, bbox_inches=bbox_main.expanded(1.1, 1.05))
                print(f"已保存主图: {main_path}")
            except Exception:
                # 退回到整图截取
                pass

            try:
                resid_line_path = os.path.join(out_dir, f'{base_name}_residuals_line.png')
                bbox_line = ax_residual_line.get_tightbbox(fig.canvas.get_renderer())
                fig.savefig(resid_line_path, dpi=300, bbox_inches=bbox_line.expanded(1.1, 1.05))
                print(f"已保存残差折线图: {resid_line_path}")
            except Exception:
                pass

            try:
                resid_hist_path = os.path.join(out_dir, f'{base_name}_residuals_hist.png')
                bbox_hist = ax_resid_hist.get_tightbbox(fig.canvas.get_renderer())
                fig.savefig(resid_hist_path, dpi=300, bbox_inches=bbox_hist.expanded(1.1, 1.05))
                print(f"已保存残差直方图: {resid_hist_path}")
            except Exception:
                pass

        except Exception as e:
            warnings.warn(f"保存图像失败: {e}")

        plt.show()

if __name__ == '__main__':
```

```
    main()
```

附录六

介绍：将 Excel 转化为 CSV 文件的代码（xlsx_to_csv.py）

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
xlsx_to_csv.py -  将 Excel XLSX 文件转换为 CSV 格式
"""

import pandas as pd
import os
import argparse
import sys
from tqdm import tqdm

def xlsx_to_csv(input_file, output_file=None, sheet_name=0, encoding='utf-8', sep=',', index=False):
    """
    将 XLSX 文件转换为 CSV 格式

    参数:
        input_file (str):  输入的 XLSX 文件路径
        output_file (str):  输出的 CSV 文件路径，默认为与输入文件同名但后缀为.csv
        sheet_name (str|int):  要转换的表格名称或索引，默认为第一个表格
        encoding (str):  输出 CSV 文件的编码，默认为 utf-8
        sep (str): CSV 文件的分隔符，默认为逗号
        index (bool):  是否保留行索引，默认为 False

    返回:
        bool:  转换成功返回 True，失败返回 False
    """
    try:
        # 如果未指定输出文件，则使用与输入文件相同的名称但更改扩展名
        if output_file is None:
            base_name = os.path.splitext(input_file)[0]
            output_file = f"{base_name}.csv"

        # 读取 Excel 文件
        print(f"正在读取文件: {input_file}")

        # 检查文件是否存在
        if not os.path.exists(input_file):
            print(f"错误:  文件  '{input_file}'  不存在")
            return False

        # 获取所有表格名称
        xl = pd.ExcelFile(input_file)
        sheets = xl.sheet_names

        if isinstance(sheet_name, int) and (sheet_name >= len(sheets) or sheet_name < 0):
            print(f"错误:  表格索引  {sheet_name}  超出范围，文件包含  {len(sheets)}  个表格")
            print(f"可用的表格: {sheets}")
            return False
```

```
        elif isinstance(sheet_name, str) and sheet_name not in sheets:
            print(f"错误: 表格名称 '{sheet_name}' 不存在")
            print(f"可用的表格: {sheets}")
            return False

        # 读取指定的表格
        print(f"读取表格: {sheet_name}")
        df = pd.read_excel(input_file, sheet_name=sheet_name)

        # 转换为CSV
        print(f"将数据转换为 CSV 并保存到: {output_file}")
        df.to_csv(output_file, encoding=encoding, sep=sep, index=index)

        print(f"转换完成! 共处理 {len(df)} 行数据")
        return True

    except Exception as e:
        print(f"转换过程中发生错误: {str(e)}")
        return False

def main():
    # 创建命令行参数解析器
    parser = argparse.ArgumentParser(description='将 Excel XLSX 文件转换为 CSV 格式')
    parser.add_argument('input_file', help='输入的 XLSX 文件路径')
    parser.add_argument('-o', '--output', help='输出的 CSV 文件路径 (默认: 与输入文件同名但后缀为.csv)')
    parser.add_argument('-s', '--sheet', default=0, help='要转换的表格名称或索引 (默认: 0，第一个表格)')
    parser.add_argument('-e', '--encoding', default='utf-8', help='输出 CSV 文件的编码 (默认: utf-8)')
    parser.add_argument('-d', '--delimiter', default=',', help='CSV 文件的分隔符 (默认: 逗号)')
    parser.add_argument('--with-index', action='store_true', help='保留行索引 (默认: 不保留)')
    parser.add_argument('-l', '--list', action='store_true', help='列出 XLSX 文件中的所有表格并退出')

    args = parser.parse_args()

    # 如果只需列出表格
    if args.list:
        try:
            if not os.path.exists(args.input_file):
                print(f"错误: 文件 '{args.input_file}' 不存在")
                return 1

            xl = pd.ExcelFile(args.input_file)
            sheets = xl.sheet_names
            print(f"文件 '{args.input_file}' 包含以下表格:")
            for i, sheet in enumerate(sheets):
                print(f"  {i}: {sheet}")
            return 0
        except Exception as e:
            print(f"列出表格时出错: {str(e)}")
            return 1

    # 尝试将 sheet 参数转换为整数
    sheet_name = args.sheet
```

```
    try:
        sheet_name = int(args.sheet)
    except ValueError:
        # 如果转换失败，保持原始字符串
        pass

    # 执行转换
    success = xlsx_to_csv(
        args.input_file,
        args.output,
        sheet_name=sheet_name,
        encoding=args.encoding,
        sep=args.delimiter,
        index=args.with_index
    )

    return 0 if success else 1

if __name__ == "__main__":
    sys.exit(main())
```