红外干涉法测量半导体外延层厚度

摘 要

针对半导体材料外延层厚度的测量问题，本文利用红外干涉法，结合材料特性建立数学模型并得出结果，精确测量半导体材料外延层的厚度。

针对问题一，要求在外延层与衬底仅有一次反射和投射的前提下所产生的干涉条纹情形下，建立确定外延层厚度的数学模型。**该问题可归结为如何利用反射率与光的干涉波动方程均存在周期性变化的特点，构建光干涉增强点和相位差之间的关系。**为此，首先**建立光干涉模型**，结合**相位与波数的物理特性**，构造二者之间的数学关系；然后**依据光的波动方程、振幅平方公式，结合光强与振幅平方关系，得到干涉光强与振幅平方相关联**；最后基于**相位差和光程差之间的关系**，得到外延层厚度数学模型。

针对问题二，问题可归结为如何利用双光束干涉的薄膜干涉估算外延层厚度问题。首先根据反射率在波数中呈现出的周期性变化建立**波数-折射率-厚度函数**，并基于问题一建立的光程差-外延层厚度几何关系确定外延层厚度；然后利用**斯涅尔定律、菲涅尔公式、光程差-干涉方程**，分别构建**基于相位重构的 Kramers-Kronig 关系近似算法**和**无相位信息的菲涅尔近似算法**；接着结合外延层反射率的周期性变化规律得出波数周期，应用两个算法得出结果，并进行**柯西色散拟合**以确定最优结果。最后结合附件一和附件二可得外延层厚度为 7.6536μm。

针对问题三，问题旨在考查多光束干涉条件的分析与误差处理方法。首先根据光的能量衰减过程得到**消光系数，反射率和表面光滑度**三类必要条件。然后对反光率使用 **FFT 完成对各个干涉曲线的解耦**，利用内部解耦的正弦函数计算理想波数周期；接着对附件三和四的反射率数据进行 **K-K 关系近似**与**菲涅尔近似**以求得波长和折射率的关系；最后利用问题二中的外延层厚度算法，结合理想波数周期和折射率关系式，解得附件三和四对应的硅晶圆片的外延层厚度，并验证所得折射率的可靠性。得到硅晶圆片外延层厚度为 3.3715μm，重新计算问题二，结果为 7.8714μm。

本文通过信号处理，物理公式以及理想假设，构建了包括测算与估计在内的多层次计算机制，分别确定了折射率和吸光系数，为红外干涉厚度测量提供了科学可靠方案。

关键词：薄膜干涉；信号处理；物理光学；柯西色散；K-K 关系

一、问题重述

1.1 问题背景

碳化硅半导体材料是新一代半导体材料，而外延层厚度是外延材料的关键参数，对于器件的性能影响极其重要。那么制定一套科学准确的外延层厚度测量算法就是重中之重。

1.2 问题提出

1.2.1 问题一：

在外延层与衬底仅有一次反射和投射的前提下所产生的干涉条纹情形，构建一个外延层厚度计算算法。

1.2.2 问题二：

根据问题一的计算算法与附件一附件二相结合，根据附件一和附件二给出的数据测算碳化硅外延层的厚度。

1.2.3 问题三：

面对多光束干涉，他的必要条件是什么以及在多干涉条件下我们如何较为精准算出外延层厚度。

二、问题分析

2.1 问题一的分析

问题一要求在外延层与衬底仅有一次反射和投射的前提下所产生的干涉条纹情形下，建立确定外延层厚度的数学模型。**该问题可归结为如何利用反射率与光的干涉波动方程均存在周期性变化的特点，构建光干涉增强点和相位差之间的关系。**为此，首先根据**相位与波数的物理特性**，构造二者之间的数学关系；然后**根据光的波动方程计算两反射光之间的相位差**；接着利用**外延层厚度与折射角的几何关系**计算两反射光之间的光程差；最后基于**相位差和光程差之间的关系**，得到外延层厚度数学模型。

2.2 问题二的分析

问题二要求在问题一的基础上，在已知波数与对应反射率的情况下，构建一套精确的外延层厚度计算模型。问题二继承了问题一的外延层与衬底仅有一次反射和投射的前提，因此可将问题场景总结为一个**双光束干涉模型**。在双光束干涉的前提下，可**利用光程差所导致的“波数周期”与折射率构建一个联系厚度 d 的数学等式**。接着利用 **K-K 关系与透明介质近似**构建反射率与折射率的数学关系。最后利用“波数周期”以及此点的折射率进行计算，得出对应的外延层厚度。最后利用**柯西色散公式**对模型进行检验衡量 ，构建一个多层次运算体系。

2.3 问题三的分析

问题三要求考虑多光束干涉情况，研究产生它的必要条件并换少它对计算结果

的影响。对于必要条件问题，该问题从光在路途中的能量消耗入手，**构建折射反射-光强的物理关系**，以此判定其必要条件。而对于厚度计算问题，该问题可以通过 **FFT 转化成与第二问相同的双光束干涉模型**。通过 FFT 可以将组成反射率的各种干涉曲线分离，按照振幅大小判断能量多少进而判断**干涉优先程度**，将问题转化为双光束干涉模型。利用 **FFT 解耦的正弦函数周期作为理想波数差**，然后结合第二问公式得出结果。

三、模型假设

假设 1：光线仅在外延层与空气的交界处表面折射。
假设 2：外延层外部为理想环境真空环境，折射率为 1。
假设 3：10°和 15°入射角可以认为是垂直射入从而简化运算。

四、符号说明

符号	含义	单位
n_i	第 i 个反射率极值处的折射率	/
v_i	第 i 个反射率极值处的波数	cm^{-1}
λ_i	第 i 个反射率极值处的波长	μm
θ	光线与法线夹角	rad
$\mathcal{H}$	希尔伯特变换	/
l	在外延层内一次反射的一半路程	μm
d	外延层厚度	μm
$R/R(\omega)$	反射率	%
$n(\omega)$	k-k 变换得出的折射率	/
$\kappa(\omega)$	k-k 变换得出的吸光系数	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 外延层厚度计算模型的思考

问题一要求在光线外延层和衬底界面仅仅发生一次反射、透射的情况下构建一个外延层厚度数学模型。外延层可被视作一个薄膜，由于薄膜足够薄，因此光线在射入和射出时的位移差极小，中间重叠的部分便是干涉区域。由于干涉时一定存在一定的相位差θ，而θ又对应了一定的光程差，光程差也可以通过波数进行计算，由此通过对干涉条件的定量分析可以搭建一个完整的数学模型。也正因为光线在延层和衬底界面仅仅发生一次反射、透射，所以仅仅有两束光线发生了干涉，随意这是一个双光束干涉模型，这里从**光强，相位差，光程差三方面入手**构建 SiC 外延层厚度公式。

5.1.2 双光束干涉数学模型的建立

由于经过不同光程，两组光束存在一定的相位差，假定反射光 1 反射光 2 的振幅分别是A_1, A_2，相位为θ_1, θ_2,此时他们的波动方程为：

$$y_1(x,t) = A_1 \cos(kx - \omega t + \theta_1) \tag{1}$$

$$y_2(x,t) = A_2 \cos(kx - \omega t + \theta_2)$$

根据光的合振动的振幅平方公式得出：

$$A^2 = A_1^{\ 2} + A_2^{\ 2} + 2A_1A_2\cos(\theta_2 - \theta_1) \tag{2}$$

因为光学中，光强I与振幅平方成正比，因此由振幅平方公式可得：

$$I = I_1 + I_2 + 2\sqrt{I_1 I_2 \cos(\theta_2 - \theta_1)}$$

由于光程差δ与角度差存在关系：

$$\delta = \frac{2\Pi}{\theta_1 - \theta_2}$$

因此最终干涉光强与光程差的关系为：

$$I = I_1 + I_2 + 2\sqrt{I_1 I_2 \cos(\frac{2\pi}{\lambda}\delta)}$$

由于存在半波损失，所以光程差为整波长时为暗条纹，光程差为半波长奇数倍的时候为亮条纹，由此计算出了光程差。假设此时为暗条纹，光程差为$\delta = m\lambda$，则光在外延层内部传播的路程 $2nl = m\lambda$，又由于厚度 d 与l存在折射角几何关系，因此可以使用l计算外延层厚度。

假设折射角为θ。结合公式，推导出厚度公式为：

$$d = \frac{m\lambda\cos\theta}{2n} \tag{3}$$

如果用相位差表示，由于结合以上公式得

$$m = \frac{d \cdot 2n}{\lambda \cos\theta} \tag{4}$$

由此得相位差厚度公式为：

$$d = \frac{\Delta\theta \cdot \lambda \cdot \cos\theta}{4\pi n} \tag{5}$$

将此公式代入光强公式中反解可得到余弦项为：

$$\cos\left(\frac{2\pi}{\lambda}\delta\right) = \frac{I - (I_1 + I_2)}{2\sqrt{I_1 I_2}} \tag{6}$$

再把相位差厚度公式代入得：

$$d = \frac{\cos\theta}{2n} \cdot \left(\frac{\lambda}{2\pi} \cdot \arccos\left[\frac{I - (I_1 + I_2)}{2\sqrt{I_1 I_2}}\right]\right) \tag{7}$$

出现的是亮条纹，则刚好存在半个波长的有效差，对于光程差来说光程差与厚

度的关系式变成了：

$$(2m+1)\frac{\lambda}{2}=\frac{2nd}{\cos\theta} \tag{8}$$

解得厚度公式为：

$$d=\frac{(2m+1)\lambda\cos\theta}{4n} \tag{9}$$

以此为例，相位差厚度公式为：

$$d=\frac{\Delta\theta\cdot\lambda\cdot\cos\theta}{4\pi n}$$

光强厚度公式为：

$$d=\frac{\cos\theta}{2n}\cdot\left(\frac{\lambda}{2\pi}\cdot\arccos\left[\frac{I-(I_1+I_2)}{2\sqrt{I_1I_2}}\right]\right)$$

5.1.3 各外延层厚度公式分析

对于亮暗条纹两种情况，在发生亮暗转换的时候表面上仅有光程差算法公式发生了转换，而其他公式没有发生变化。这是因为他们在不同干涉条件下的变化的物理变量不同。由于波长为常数，所以需要用倍数来表示光程差大小。而相位差与光强本身就是变量，只需要利用对应的数据便可以计算结果。

5.2 问题二的模型建立与求解

5.2.1 外延层厚度模型的完善

问题二要求在问题一的基础上进行计算，根据附件一附件二的内容计算出外延层的厚度。由于附件一附件二中的反射率存在干涉导致的数据异常，因此我们需要对此进行滤波保证数据质量。由于波数-反射率中反射率的周期性变化体现了光的干涉原理，可以用来构建厚度公式，因此在滤波后根据折射率与波数周期推导得出的厚度-波数公式计算结果。对于重要参数折射率采用 k-k 关系近似与透明介质近似两种算法独立求解，对结果进行差值比较检查可信度。但是由于已知光学性质不足，较难衡量结果的可信程度，因此我们使用柯西色散方程进行最后的结果检验。引入 SiC 的经验参数弥补光学性质的不足，从而衡量模型是否可信，依照得出的拟合结果提取最优结果。

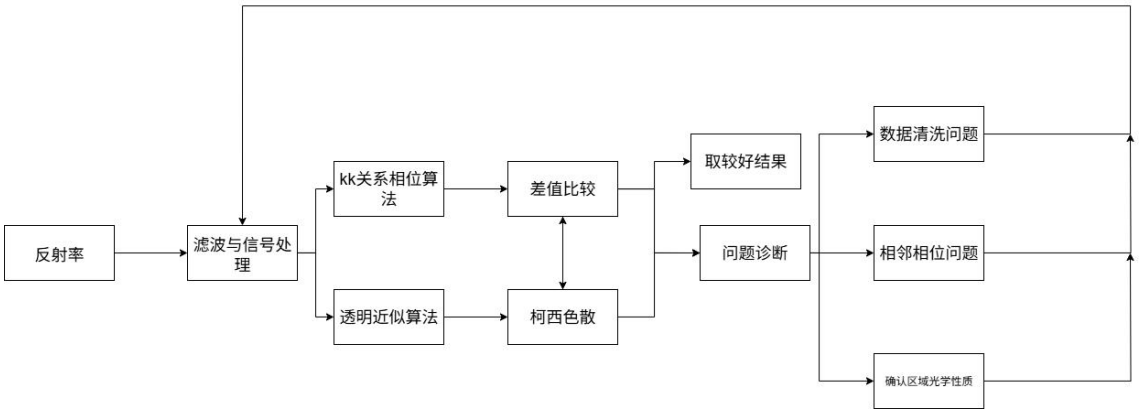

图 2 问题二外延层厚度算法流程图

5.2.2 K-K 关系近似与菲涅尔反射算法

K-K 关系（Kramers-Kronig 关系）是用来描述线性响应系统之间“实部”与

“虚部”之间的关系：在研究线性响应系统（像光学里光和物质相互作用这类系统）时，会涉及到一些物理量的“复表示”，也就是把一个物理量拆成“实部”和 “虚部” 两部分。而对于这个问题**光学的复折射率公式中实部$n(\omega)$描述折射率，虚部$\kappa(\omega)$描述吸光系数**，这里携带ω是为了表述两者与波长之间的紧密联系。其公式如下：

$$n(\omega)-1=\frac{1}{\pi}\mathcal{P}\int_{-\infty}^{\infty}\frac{\kappa(\omega')}{\omega'-\omega}\,d\omega'$$

$$\kappa(\omega)=-\frac{1}{\pi}\mathcal{P}\int_{-\infty}^{\infty}\frac{n(\omega')-1}{\omega'-\omega}\,d\omega' \tag{10}$$

应用在光学的复折射率公式为：

$$\tilde{n}(\omega)=n(\omega)+i\kappa(\omega) \tag{11}$$

但是由于 **K-K 公式的原始积分需覆盖“0 到无穷大”的全频率**，但红外干涉法只能测量有限频率，若想应用 K-K 原始公式，需对未测频段做复杂近似，但这样会导致较大误差，计算复杂度也较高。此外附件给的是离散数据，原始公式很难处理。因此我们使用**希尔伯特近似**，公式如下：

$$\theta(\omega)=-\mathcal{H}\left[\ln\sqrt{R(\omega)}\right] \tag{12}$$

在这里希尔伯特求出$\theta(\omega)$是入射光的相位偏移量，再用**菲涅尔反射公式**反推折射率与吸光系数，公式如下：

$$n(\omega)=\frac{1-R(\omega)}{1+R(\omega)-2\sqrt{R(\omega)}\cos\theta\,(\omega)} \tag{13}$$

$$\kappa(\omega)=\frac{2\sqrt{R(\omega)}\sin\theta\,(\omega)}{1+R(\omega)-2\sqrt{R(\omega)}\cos\theta\,(\omega)} \tag{14}$$

最后我们可以得出当前条件的寻常折射率$n(\omega)$与吸光系数$\kappa(\omega)$。

5.2.3 厚度-波数的公式推导

根据问题一解答中的光程差-厚度模型求解，首先假设干涉时完全抵消，由于存在半波损失且**折射率不为常数**，因此光程差 OPD 符合如下公式：

$$\mathrm{OPD}=\mathrm{m}\lambda_{\mathrm{i}},\mathrm{OPD}=2\mathrm{n_i l} \tag{15}$$

其中λ_{i}与$\mathrm{n_i}$分别为第 i 个反射率极值处的波长与折射率

干涉级数关系：

$$\mathrm{m}=\frac{2\mathrm{n_i}l}{\lambda_{\mathrm{i}}},m+1=\frac{2n_{i+1}l}{\lambda_{i+1}} \tag{16}$$

波长域极值作差推导干涉条件：

$$1=\frac{2\lambda_{\mathrm{i}}\mathrm{n_{i+1}}l-2\lambda_{\mathrm{i+1}}\mathrm{n_i}l}{\lambda_{\mathrm{i}}\lambda_{\mathrm{i+1}}} \tag{17}$$

折射光线一次反射的半边长：

$$l=\frac{\lambda_{\mathrm{i}}\lambda_{\mathrm{i+1}}}{2\lambda_{\mathrm{i}}\mathrm{n_{i+1}}-2\lambda_{\mathrm{i+1}}\mathrm{n_i}} \tag{18}$$

半边长与角度的关系：

$$d=lcos\theta$$

折射角路程修正：

$$\mathrm{n}\cos\theta = \sqrt{\mathrm{n}^2 - \sin^2\theta_{入射}} \tag{19}$$

最后修正角度并把波数转为波长（nm）：

$$\mathrm{d} = \frac{10^7}{2}\cdot\frac{1}{\nu_{i+1}\sqrt{\mathrm{n}_{i+1}^2 - \sin^2\theta_{入射}} - \nu_i\sqrt{\mathrm{n}_i^2 - \sin^2\theta_{入射}}} \tag{20}$$

由此我们将**厚度d转变为一个与折射率和波数的公式**，接下来的任务就是结合实际和实验数据求解折射率等参数来求解。其他例如增强型干涉推导过程类似，结果相同，这里不再赘述。

5.2.4 辅助模型的建立

即使 k-k 近似算法可以很好地给出范围数据，但是单独计算可信度不高。经过对相关资料的查询，可知一般 **SiC 材料在中红外区域的吸光系数为 10－50 cm^{-1}**，属于中低段吸收，误差可以忍受，以此为基础我们再用简化的透明近似算法（忽略吸光系数）再进行一次计算，以此进行结果双认证。

对于透明介质，反射率与折射率的关系为：

$$R = \left(\frac{n-1}{n+1}\right)^2 \tag{21}$$

解方程得到：

$$n = \frac{1+\sqrt{R}}{1-\sqrt{R}} \tag{22}$$

然后将反射率代入计算出粗略估计值n'并与 K-K 关系近似计算出来的进行对比，误差较小便认为结果可信。

5.2.5 柯西色散方程检验法

尽管已经有两个模型进行相辅相成的计算，但是也有可能算法同时输出错误结果，因此我们需要从材料本身出发构建一个检验模型。但是已知性质甚少，所以最后设计使用柯西色散方程作为检验方法。

柯西色散方程是一种经验方程，系数经过实验得出并且可靠。尽管可能没有严格的公式推导，但是在生产生活中良好的拟合效果完全可以用于检验模型优良程度，去弥补光学性质不足从而难以构建检验算法的困境。以下是柯西色散方程的一般形式：

$$n(\lambda) = A + \frac{B}{\lambda^2} + \frac{C}{\lambda^4} + \frac{D}{\lambda^6} + \cdots \tag{23}$$

其中 A,B,C…是材料的柯西系数（由实验测定），公式直接将波长和折射率联系起来，在附件中可以对应波数的物理关系。又因为反射率与波数有物理关系，因此在上述模型中折射率同样与波数存在物理关系。所以，使用柯西色散方程来检验两个模型，取最优模型结果作为最后结果。

5.2.6 滤波器的选择与清洗

由于干涉会影响波数周期的求解，因此我们必须进行滤波去取出影响较大的区

域，对反射率的可视化如下图：

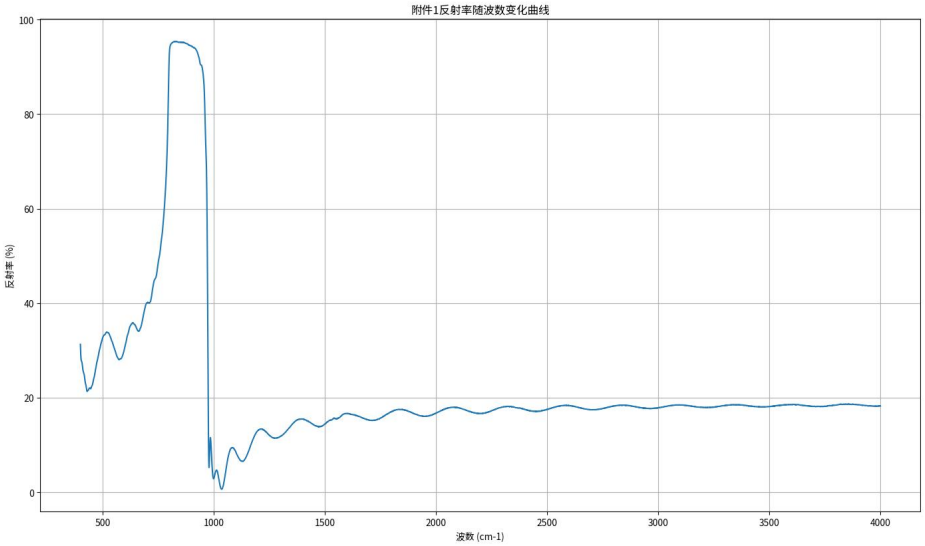

图 3 附件 1 的波数-反射率可视化

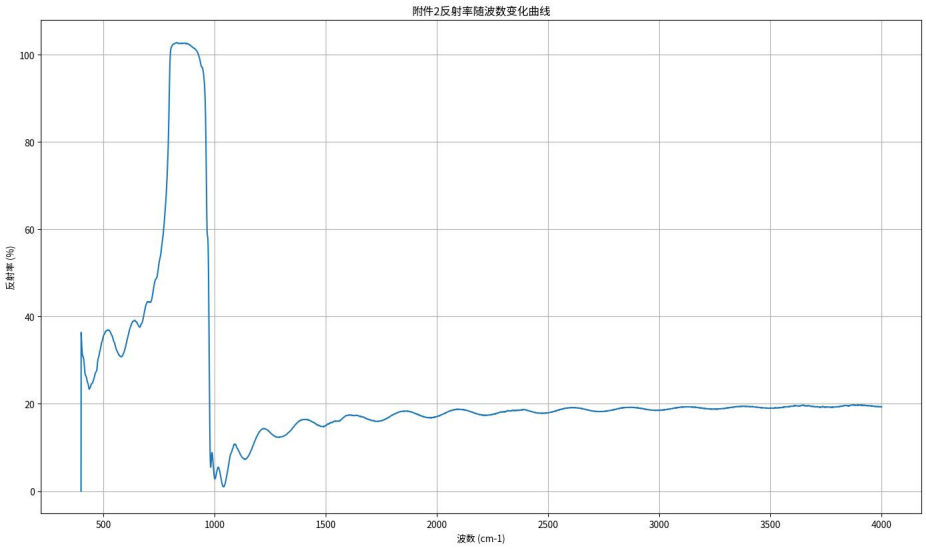

图 4 附件 2 的波数-反射率可视化

通过对附件一的可视化（图 3）与附件二的可视化（图 4），我们可以很明显的看到由于干涉导致的异常峰值，在这一段的反射率极其高，附件二甚至达到了100%以上，所以我们选择**高通滤波**进行滤波，将峰值部分全部过滤仅仅留下低频率部分并提取极值周期最后求解。

5.2.7 外延层厚度求解

经过滤波器的反射率数据分别应用于公式(4)(5)和(14)，分别计算出各自的折射率函数，其各自结果如下图：

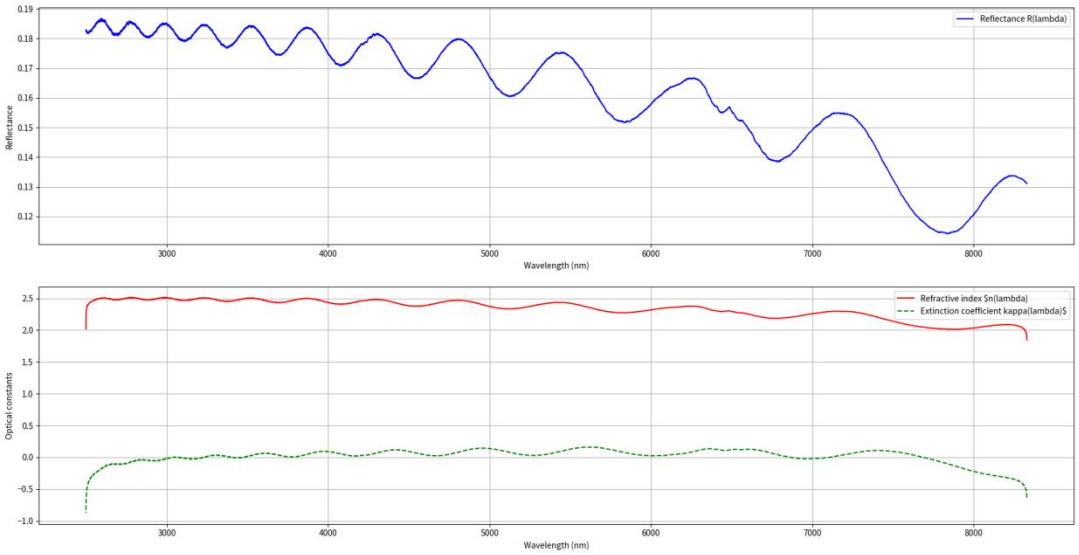

图 5 附件一解算折射率可视化

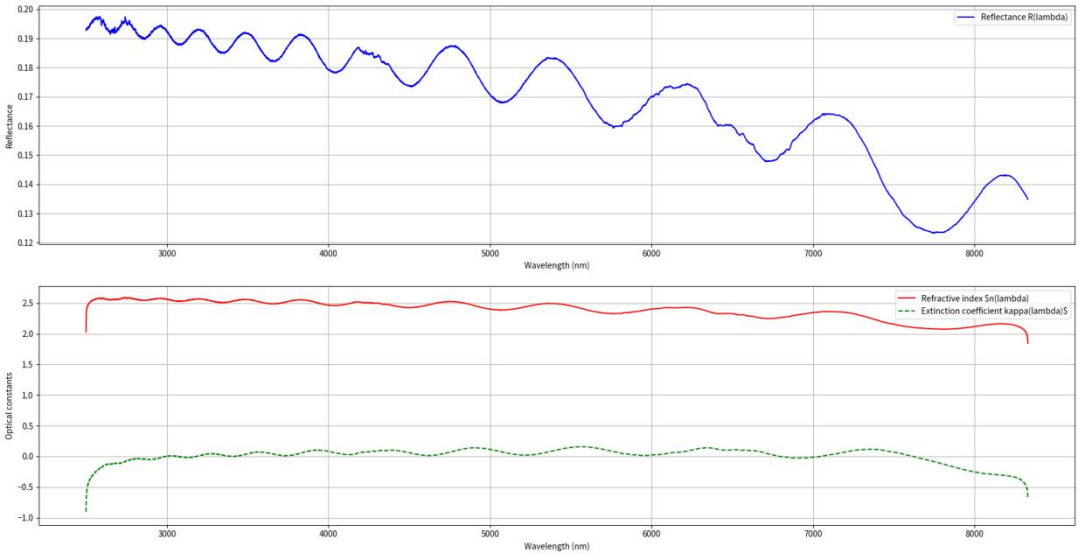

图 6 附件二解算折射率可视化

由图 5 和图 6 得知，在低频段稳定区域中，除去边缘效应导致的结果异常，大体上附件一二都保持了一个稳定的折射率与吸光系数，**吸光系数较低，在 0 左右**。而相对来说**折射率便高得多，大体在 2.0 到 2.5 之间**，且滤波后的数据呈现一个周期态势，表现良好。

因为折射率非常数，所以采取两两计算，即公式（12），最后两两计算后去除最大值最小值取得平均值。最后两个算法的四个结果如下表：

算法 \ 附件	附件一	附件二
K-K 关系近似	7.8738μm	7.6554μm
透明介质近似	7.9406μm	7.7289μm

两个模型各自算得出的厚度值相差极小，在差值检测中认为可信，再经过柯西色散检验，对每个有关于折射率的表达式进行检验拟合计算R^2，按照拟合效果决定最后结果 。

5.2.8 外延层厚度模型的检验与答案确定

根据资料，最常见的 4H-SiC 的经验参数分别为 A = 2.5-2.6，B = 0.01-0.015，C = 0.001-0.003。因此我们建立两参数和三参数的柯西色散方程并对两个模型进行拟合检验，其可视化如下 ：

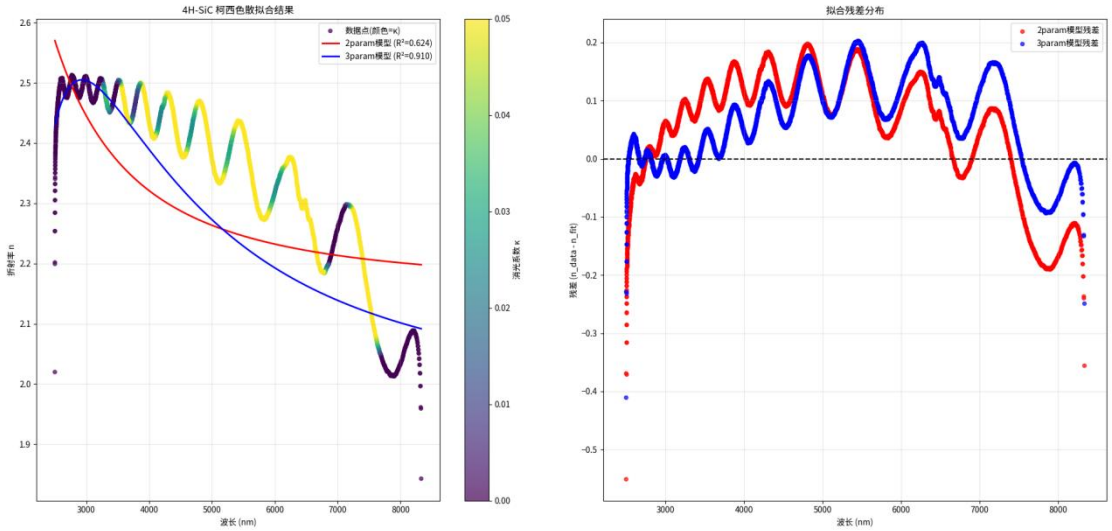

图 7 K-K 近似算法柯西色散检验

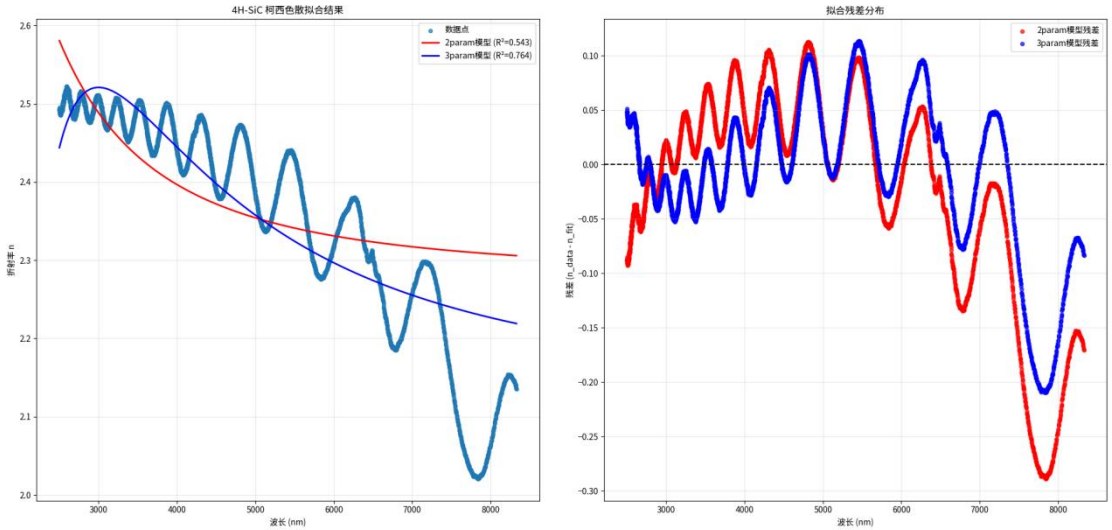

图 7 透明介质近似算法柯西色散检验

由图可知，K-K 算法近似在整体上比透明介质近似拟合程度更高，尤其在两端数据，几乎完美契合了经验公式的重建数据，但是透明介质近似算法就相对较差。但是数据中最差的部分都是中心部分，推测是因为材料的内部杂质导致了在低频段中保有了一定的吸光率导致中心频段处误差较大，但是在两端却可以得到较好拟合结果。

最后拟合结果为：K-K 关系近似R^2=0.912042，透明近似算法R^2=0.784129。一般在检验中R^2>0.9 为可接受，因此 K-K 关系结果可信度最高，对两个附件取得平均值作为最后结果，其结果为 d=7.7646μm

5.2.9 外延层厚度算法结果分析

透明近似算法效果不佳有可能是因为 SiC 的吸光效应，即使在中红外光仅仅属于中低段的吸收系数也会产生极大的影响。根据 K-K 算法结果，$\kappa(\omega)$的范围处在 0.5 左右。忽略这些会导致折射率偏大从而导致整体偏大。又由于采取了简化公式，导致误差进一步加大最后导致R^2偏低，结果误差较大。而 K-K 近似考虑了更多的因素，效果更好，可信度更高。

5.3 问题三的模型建立与求解

5.3.1 多光束干涉的思考与构建

问题三要求对半导体外延层进行多光束干涉的必要条件评估，同时分析对结果的影响并进行修正。对于这个必要条件评估问题，需要对外延层每一次反射投射进行计算前后的能量比值，最后归纳出公式并通过设置不同光学参数作为不同条件进行**迭代模拟**。设置一个能量阈值，如低于阈值则认为光线消散。所以最后目标转化为了寻找能让光线迭代第二次就消失的光学参数。同时对于结果影响问题，主要分析多光束干涉的数据会影响什么，用什么方法可以避免。最后重新计算问题二并给出结果。

5.3.2 多光束干涉仿真迭代算法的建立

多光束干涉的物理示意图如图所示：

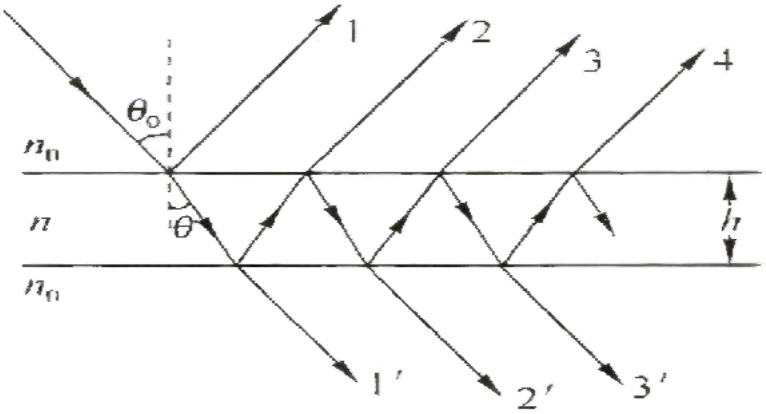

图 8 多光束干涉示意图

5.3.3 多折射光强公式推导

假设底面没有出射光线且外延层的反射率为 R，忽略路途上的能量损失，上下面距离为 d。在光线首次入射的时候，首次反射的光线光强为：

$$I_{r1} = I_i R \tag{24}$$

反射后折射光线光强：

$$I_{n1} = (1 - R) I_{\mathrm{i}} \tag{25}$$

回到上底面的干涉：

$$I_{r2} = (I_i(1 - R))(1 - R) = I_i(1 - \mathrm{R})^2 \tag{26}$$

第二束折射完成，光的光强：

$$I_{r3} = I_i(1-R)R^2(1-R) = I_i(1-R)^2R^2 \quad (27)$$

第三束折射完成，光的光强：

$$I_{r4} = I_i(1-R)^2R^4 \quad (28)$$

一直到第 m 束干涉光光强：

$$I_{r(m)} = I_i(1-R)^2R^{2m} \quad (29)$$

由此可知，R 必须要足够大，如果 R 过小则导致第二束干涉光的光强过低，被认定为消失而并非参与干涉。因此反射率足够大则为发生多光束干涉的必要条件。

5.3.4 多光束干涉必要条件总结与判定

对多光束干涉分析发现，如果要存在多光线干涉，最终的目标只有一个：保证第二束折射光的光强不过小。由多折射光强公式的结论得知反射率 R 必须足够大才能保证多光束干涉，所以其他可以保证第二束折射光的光强不过的条件也可以纳入基础条件之中，因此总结条件为：

1. **消光系数低**，在外延层传输的能量损耗较小。
2. **反射率足够大**，保证第二次干涉有足够的光强。
3. **表面光滑**，不光滑的表面难以形成有向光束产生衍射。

如果要实现多光束干涉，以上三条作为基本条件必须达成，缺一不可。

根据附件三四中的反射率数据以及拟合出的吸光系数进行估计。根据下图 9 与下图 11，可以得出**吸光率几乎为 0 而 R=48%**，在第二次的光强度为原有的四分之一左右，存在多光束干涉。对附件一二计算结果相同，同样判定存在多光束干涉，因此需要重新计算。

5.3.5 快速傅里叶变换与反射率分解

在信号处理中快速傅里叶变换（FFT）是将信号从时域转为频域的高效计算方法。这里由于最后的反射率图像为多条光线干涉图像叠加的结果，而且每个图像理想状态均符合正态分布，所以这里对反射率运用 FFT 可以**将图像从“叠加变化”变为有关于波数周期的“独立光线干涉”**，其公式如下：

$$X[k] = G[k] + W_N^k H[k] \quad X\left[k + \frac{N}{2}\right] = G[k] - W_N^k H[k] \quad (30)$$

$G[k]$是偶数项子序列

$H[k]$是奇数项子序列

W_N^k是旋转因子

通过利用 FFT 可以将各个干涉光线的周期提取出来，通过判断振幅大小（振幅越大能量越大，在介质中能量损耗越小，在外延层的的光程也越短）可以直接提取到精确的波数周期用于计算外延层厚度。

5.3.6 多光束干涉干扰结果因素分析

问题二的求解是在滤波后直接在波数-反射率中提取的波数周期进行计算，但是原先的问题二这是建立在双光束干涉模型的基础上提取的波数周期。在多光束模型干涉中往往参与干涉的不只是两束光，在这个基础上在介质内反射 3 次才发生干涉甚至更多次数的也会叠加在最后的反射率图谱上，我们可以通过 FFT 直观

了解：

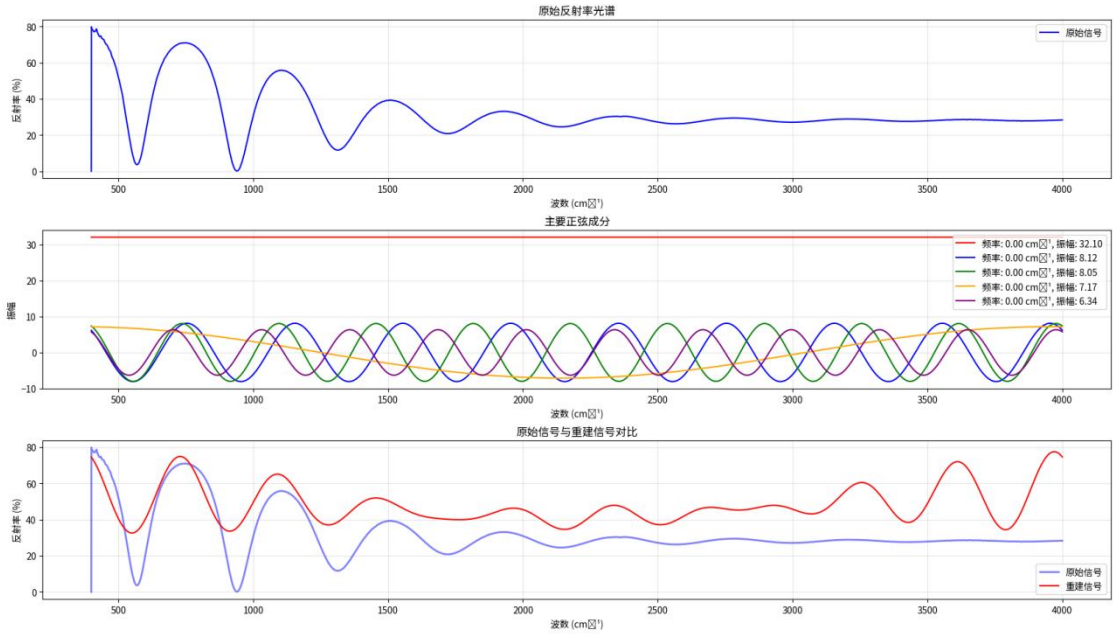

图 9 波数-反射率 FFT 解算示意图

FFT 后取最关键的五个分量可视化并分析的计算结果，我们可以发现它存在一个直流分量和四个交流分量（正弦波）。其中直流分量代表的是不发生任何干涉时外延层的反射率，其余四个交流分量除去第四个可能是受衬底的影响之外，其余三个周期相差不大，均有可能是干涉导致。但是他们又存在一定的相位差 ，由此导致利用双光束干涉提取周期的误差加大。同时由于多光束干涉的叠加效应，反射率波动更加明显且尖锐，导致折射率测算不准确。

综上所述，多光束干涉会将原始数据的波数周期扰乱，同时尖锐化数据，让折射率的测算误差变大，最后导致外延层厚度计算极其不准确。

5.3.7 多干涉光强模型的构建

对于附件三四的总反射率，可在能量层面推导出反射光与相位差等价的关系式。设A_1 ~A_n为从 1 到 n 的反射光，r 为单次反射率，以下是公式推导：

$$A_1 = Ar_1$$
$$A_2 = At^2r_2$$
$$A_3 = At^2r^2r_1 \qquad (31)$$
$$\cdots\cdots$$
$$A_n = At^2r_2^{n-1}r_1^{n-2}$$

设$\widetilde{U_R}$为复振幅，可得：

$$\widetilde{U}_R = \sum_{j=1}^{\infty} \widetilde{U}_j = A\,r_1 + A\,t^2\,r_2\,e^{i\delta} + A\,t^2\,r_2^2\,r_1\,e^{2i\delta} + \cdots \qquad (32)$$

$$\widetilde{U}_R = \left[r_1 + \frac{t^2 r_2 e^{i\delta}}{1 - r_2 r_1 e^{i\delta}}\right] A \qquad (33)$$

由此将光强作为复平面展开得：

$$I_i = \left[r_1 + r_1^3 r_2^2 + r_2\left(1 - 3r_1^2\right)\cos\theta\right]^2 \tag{34}$$

$$I_j = r_2^2\left(1 - r_1^2\right)^2 \sin^2\delta$$

由此构建了相位差-单次反射率-光强公式。

由于相位差公式为：

$$\delta = \frac{2\pi}{\lambda}\Delta L = \frac{4\pi n d\cos\theta}{\lambda} \tag{35}$$

结合能量守恒定律，进入衬底的光子能量与出射的光子能量平方和恒为 1.

最后由光强与复振幅关系可得：

$$I_R = \tilde{U}_R\tilde{U}_R^* = \frac{I_i + I_j}{(1 + r_1 r_2 - 2r_1 r_2\cos\theta)^2} I_\lambda \tag{36}$$

5.3.8 对单晶硅片外延层厚度的求解

1. 确定可靠波数周期

多光束干涉导致的图像周期偏差的原因是多种干涉图像叠加的结果，但是理想状态下各个干涉图像均为近似的正弦函数，正如图 9 所示。由此在 FFT 操作后进行找到振幅最大的，最稳定的正弦函数作为第一次干涉所形成的周期，这样问题又**回到了类似于双光束干涉的问题**，而这个正弦函数的周期就是我们的理想波数周期，可视化如下图：

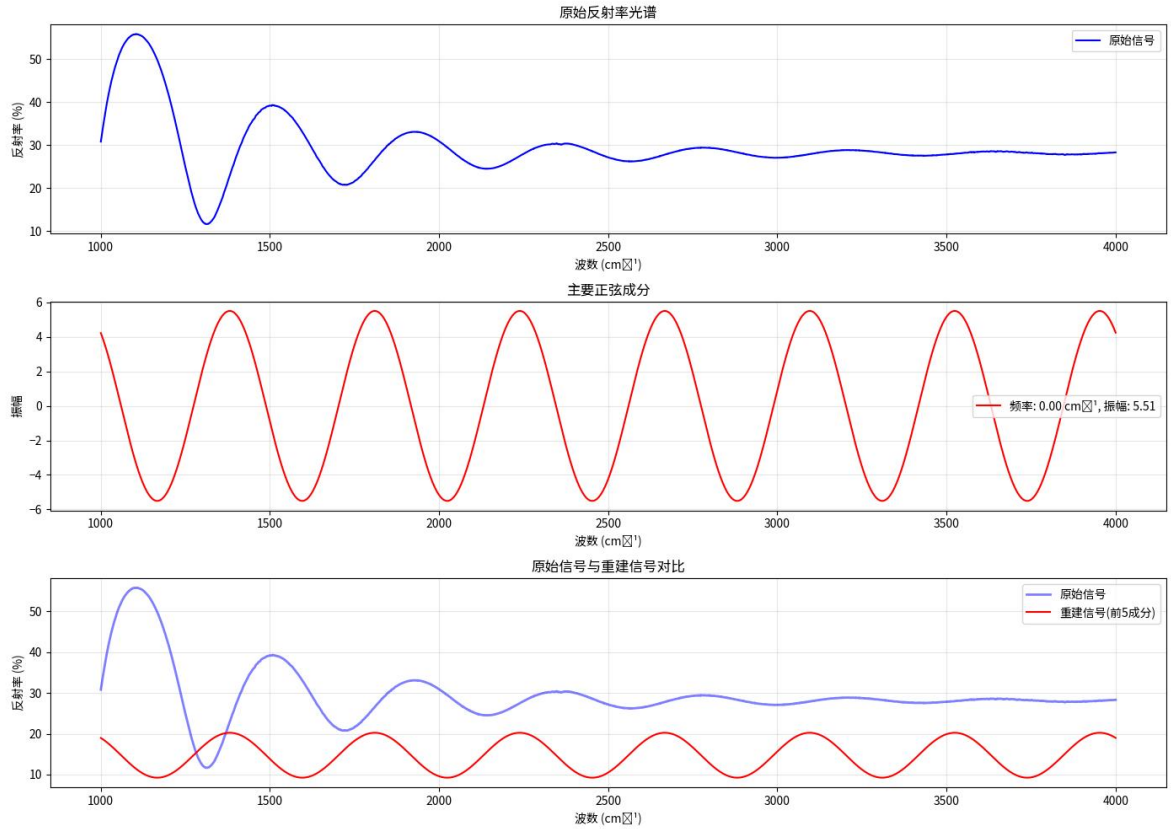

图 10 滤波后的 FFT 及其提取主要交流信号

由此我们确定理想波数周期为 428.6017 cm^{-1}

2. 确定折射率 n

对已知的附件三附件四反射率运用问题二构建的 **K-K 关系近似与透明介质近似算法**，以全局反射率运用 5 计算当前波长的折射率 n。观察分析折射率 n 的变化趋势从而确定滤波范围。经过分析最终选择高通滤波，滤过波数小于 1000 的数据，仅仅考虑大于 1000 的平稳数据，其解算结果可视化如下：

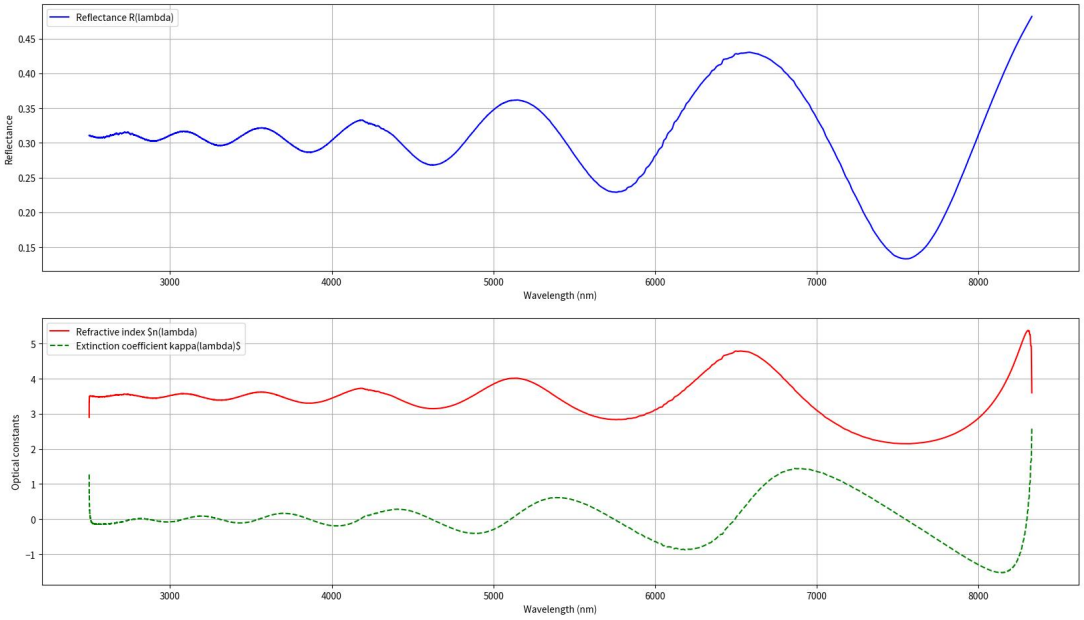

图 11 附件三四解算折射率与吸光系数

根据左边平稳曲线与其他频段均值，由此我们确定理想的折射率范围为 3.5 左右。

3. 确定晶硅片外延层厚度

根据公式（20），将理想正弦函数极值与对应的各点的折射率代入公式（20）求取结果，其结果如下表所示：

算法 \ 附件	附件三	附件四
K-K 关系近似	3.445μm	3.100μm
透明介质近似	3.553μm	3.190μm

根据已知的单晶硅折射率在 3.5 左右，根据计算 K-K 近似在稳定区域近似在 3.5，而透明介质近似在较不稳定区域也同样稳定在了 3.5 左右，由此我们原则透明近似算法作为最后结果，取得两平均值为 3.3715μm。

4. 重新计算问题二

对附件一附件二进行 FFT 提取理想波程差，再基于第二问的算法步骤重新求解，与其余与步骤 3 相同，输出的结果为：

算法 \ 附件	附件一	附件二

K-K 关系近似	7.8535μm	7.6113μm
透明介质近似	7.8907μm	7.7004μm

同样根据柯西色散检验，K-K 关系近似优势不变，依然用 K-K 近似算法答案取平均值，取得结果厚度为 7.8714μm。

5.3.9 问题三模型结果检验

1. 差值检验

通过对问题三模型结果进行对比与差值检验，在 10 与 15 的情况下，两种算法结果误差均不超过 2.5%。代表两种算法捕捉了最广泛最明显的特征区域用来计算折射率和吸光系数，对应效果好。

2. 物理值检验

对于单晶硅外延层在中红外波段的高吸收高扰动，这里直接对单晶硅的折射率进行对应。根据资料，单晶硅的折射率为 3.5 左右，图 11 平稳段的折射率上下波动在 3.2 到 3.9 之间，实际效果良好。

六、模型优缺点

6.1 模型优点

1.该模型不依靠单独的算法，利用多层次检验且与经验公式结合，保证结果的可信度

2.该模型针对红外干涉光周期性变化，建立波数-折射率-厚度函数，可以精准匹配半导体外延层在红外波段的特点

3. 该模型主要基于各类物理公式的推导与延伸来构建，先验较少，泛用性佳。

6.2 模型缺点

1.本文模型高度依赖附件中的数据反射率，若反射率变化周期过短或峰值过低会提高误差。

2.未讨论温度等其他外界干扰因素，并未留下修正机制。

参考文献

[1] 赵凯华. 《新概念物理教程》光学[M]. 北京: 高等教育出版社, 2005: 4.
[2]陈长青, 毛旭, 周祯来, 等. SiC 薄膜红外及紫外-可见光谱的研究[J]. 红外技术, 2005, 27(4): 314-318.]
[3]马科斯·玻恩, 埃米尔·沃耳夫. 光学原理[M]. 杨葭荪,译. 北京: 电子工业出版社, 2009.

本参赛队未使用任何 AI 工具。

附录

附录 1：支撑材料清单：

Q1.py	原始波数差代码
Q_2.py	折射率计算与柯西色散检验代码
Q_3.py	FFT 提取理想波数差
Q_4.py	Sellmeier 色散模型检验代码
其他文件	参考文献与可视化图片

主要代码汇总：

Q_1 ：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
import zhplot

# 加载数据
data = pd.read_excel('B 题\B 题\附件\附件 2.xlsx')
lmd = data['波数  (cm-1)'].values
rsl1 = data['反射率  (%)'].values

# 选择分析范围
# mask = (lmd >= 1000) & (lmd <= lmd.max())
# # mask = (lmd <= 2500)
# lmd = lmd[mask]
# rsl1 = rsl[mask]
```

```
rsl11 = gaussian_filter1d(rsl1, sigma=50)

deepdick = []
deepdick_ = []
window_size = 100
for i in range(window_size, len(rsl1)-window_size):
    is_peak = True
    for j in range(1, window_size+1):
        if rsl1[i] <= rsl1[i-j] or rsl1[i] <= rsl1[i+j]:
            is_peak = False
            break
    if is_peak:
        deepdick.append(lmd[i])
        print(i);
for i in range(window_size, len(rsl1)-window_size):
    is_peak = True
    for j in range(1, window_size+1):
        if rsl1[i] >= rsl1[i-j] or rsl1[i] >= rsl1[i+j]:
            is_peak = False
            break
    if is_peak:
        deepdick_.append(lmd[i])
        print(i);
# 计算相邻峰值的间隔
if len(deepdick) >= 2:
    intervals = np.diff(deepdick)  # 计算相邻峰值的波数差
    avg_interval = np.mean(intervals)      # 计算平均间隔
    std_interval = np.std(intervals)       # 计算标准差
    print(f"峰值波数(cm⁻¹):", ["%.2f" % x for x in deepdick])
    print(f"检测到 {len(deepdick)} 个峰值")
    print("相邻峰值间隔(cm⁻¹):", ["%.2f" % x for x in intervals])
    print(f"平均间隔: {avg_interval:.2f} ± {std_interval:.2f} cm⁻¹")


    n = 2.1
    thickness = 1 / (2 * n * avg_interval) * 1e7
    print(f"计算得到的薄膜厚度: {thickness:.2f} nm")
else:
    print("检测到的峰值不足，无法计算间隔")

# 可视化
plt.figure(figsize=(12, 6))
plt.plot(lmd, rsl1, 'b-', label='原始反射率')
plt.plot(deepdick, rsl1[np.isin(lmd, deepdick)], 'ro', label='检测到的峰值')

# 标注间隔
if len(deepdick) >= 2:
```

```
    for i in range(len(deepdick)-1):
        mid = (deepdick[i] + deepdick[i+1])/2
        plt.annotate(f'Δ={intervals[i]:.1f}',
                     xy=(mid, np.mean(rsl1[np.isin(lmd, deepdick)])),
                     ha='center', va='bottom', fontsize=8)

plt.xlabel('波数 (cm-1)')
plt.ylabel('反射率 (%)')
plt.title(f'峰值间隔分析 (平均间隔: {avg_interval:.2f} cm⁻¹)' if len(deepdick)>=2 else '
峰值检测')
plt.legend()
plt.grid(True)
plt.show()

# 计算相邻峰值的间隔
if len(deepdick_) >= 2:
    intervals = np.diff(deepdick_)  # 计算相邻峰值的波数差
    avg_interval = np.mean(intervals)       # 计算平均间隔
    std_interval = np.std(intervals)        # 计算标准差
    print(f"峰值波数(cm⁻¹):", ["%.2f" % x for x in deepdick_])
    print(f"检测到 {len(deepdick_)} 个峰值")
    print("相邻峰值间隔(cm⁻¹):", ["%.2f" % x for x in intervals])
    print(f"平均间隔: {avg_interval:.2f} ± {std_interval:.2f} cm⁻¹")


    n = 2.1
    thickness = 1 / (2 * n * avg_interval) * 1e7
    print(f"计算得到的薄膜厚度: {thickness:.2f} nm")
else:
    print("检测到的峰值不足，无法计算间隔")

# 可视化
plt.figure(figsize=(12, 6))
plt.plot(lmd, rsl1, 'b-', label='原始反射率')
plt.plot(deepdick_, rsl1[np.isin(lmd, deepdick_)], 'ro', label='检测到的峰值')

# 标注间隔
if len(deepdick_) >= 2:
    for i in range(len(deepdick_)-1):
        mid = (deepdick_[i] + deepdick_[i+1])/2
        plt.annotate(f'Δ={intervals[i]:.1f}',
                     xy=(mid, np.mean(rsl1[np.isin(lmd, deepdick_)])),
                     ha='center', va='bottom', fontsize=8)

plt.xlabel('波数 (cm-1)')
plt.ylabel('反射率 (%)')
plt.title(f'峰值间隔分析 (平均间隔: {avg_interval:.2f} cm⁻¹)' if len(deepdick_)>=2 else
```

```
'峰值检测')
plt.legend()
plt.grid(True)
plt.show()
```

Q_2：

```
import numpy as np
from scipy.integrate import quad
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import pandas as pd
import scipy.signal
import math
import zhplot
from scipy.optimize import curve_fit
# 定义入射角度（度）
哈气 = 15.0
耄耋 = math.radians(哈气)
入射平方 = math.sin(耄耋)**2

# 数据加载与预处理
data = pd.read_excel('B 题\B 题\附件\附件 3.xlsx')
lmd = data['波数 (cm-1)'].values
rsl = data['反射率 (%)'].values
mask = (lmd >= 1200) & (lmd <= lmd.max())
lmd = lmd[mask]
rsl = rsl[mask]
# 选择分析范围并归一化反射率
R = np.clip(rsl / 100, 1e-6, 0.999)  # 确保 R 在(0,1)范围内

# 单位转换：波数(cm⁻¹) → 波长(nm) → 角频率(rad/s)
CCB = 1e7 / lmd  # 波数(cm⁻¹)转波长(nm): λ = 1e7/ν̃
c = 2.99792458e8  # 光速(m/s)
omg = 2 * np.pi * c / (CCB * 1e-9)  # ω = 2πc/λ

# 按ω升序重新排序所有数据
sort_idx = np.argsort(omg)
omg = omg[sort_idx]
R = R[sort_idx]
CCB = CCB[sort_idx]

# 创建反射率的插值函数（用于 KK 积分）
diangun = interp1d(omg, R, kind='cubic', bounds_error=False, fill_value=0)

# 快速 KK 积分计算相位θ(ω)（使用 Hilbert 变换近似）
luguan = np.log(np.sqrt(R))
theta = -np.imag(scipy.signal.hilbert(luguan))  # 希尔伯特变换
print(theta)
```

```
# 计算复折射率 n + iκ
sqrt_R = np.sqrt(R)
cos_theta = np.cos(theta)

n1 = (1 - R) / (1 + R - 2 * sqrt_R * cos_theta)
n2 = (1 + sqrt_R) / (1 - sqrt_R)

kappa = 2 * sqrt_R * np.sin(theta) / (1 + R - 2 * sqrt_R * cos_theta)
x = ['1393.80','1822.40','2251.00','2679.60','3108.21','3536.64','3950.46']
# x = ['1275.68', '1474.31', '1713.44', '1948.72', '2200.86', '2451.08', '2705.64', '2965.50',
'3216.20', '3466.90', '3709.41']
for i in range(len(x)):
    x[i] = eval(x[i])

# 使用 n1 计算厚度
nm = [0.0] * len(x)
for i, dingdongji in enumerate(x):
    dagoujiao = 1e7 / dingdongji
    idx = np.argmin(np.abs(CCB - dagoujiao))
    nm[i] = n1[idx]
    print(f"波长 {dagoujiao:.2f} nm (波数 {dingdongji} cm⁻¹) 的折射率 n =
{nm[i]:.4f}")

value_pro = 0
min_val = float('inf')
max_val = float('-inf')
for i in range(0, len(x) - 1):
    daishuji = nm[i+1]**2 - 入射平方
    jiankangma = nm[i]**2 - 入射平方

    if daishuji < 0 or jiankangma < 0:
        print(f"跳过无效计算（i={i}）：daishuji={daishuji:.4f},
jiankangma={jiankangma:.4f}")
        continue

    value = 1e7 * (1/2) * 1 / (x[i+1] * math.sqrt(daishuji) - x[i] *
math.sqrt(jiankangma))
    if value < min_val:
        min_val = value
    if value > max_val:
        max_val = value
    value_pro += value
    print(f'x[{i}]={x[i]}, x[{i+1}]={x[i+1]} -> 计算值 = {value:.4f}")
    print(f'计算值 (i={i}): {value:.4f}")

num_valid = len(x) - 1 - 2
if num_valid > 0:
```

```
        final_thickness = (value_pro - min_val - max_val) / num_valid
        print(f"最终计算的薄膜厚度: {final_thickness:.2f} nm")
else:
        print("无有效计算值")

# 可视化
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(CCB, R, 'b-', label='Reflectance R(lambda)')
ax1.set_xlabel('Wavelength (nm)')
ax1.set_ylabel('Reflectance')
ax1.legend()
ax1.grid(True)

ax2.plot(CCB, n1, 'r-', label='Refractive index $n(lambda)')
ax2.plot(CCB, kappa, 'g--', label='Extinction coefficient kappa(lambda)$')
ax2.set_xlabel('Wavelength (nm)')
ax2.set_ylabel('Optical constants')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# 使用 n2 计算厚度
nm = [0.0] * len(x)
for i, dingdongji in enumerate(x):
        dagoujiao = 1e7 / dingdongji
        idx = np.argmin(np.abs(CCB - dagoujiao))
        nm[i] = n2[idx]
        print(f"波长 {dagoujiao:.2f} nm (波数 {dingdongji} cm⁻¹) 的折射率 n = {nm[i]:.4f}")

value_pro = 0
min_val = float('inf')
max_val = float('-inf')
for i in range(0, len(x) - 1):
        daishuji = nm[i+1]**2 - 入射平方
        jiankangma = nm[i]**2 - 入射平方

        if daishuji < 0 or jiankangma < 0:
                print(f"跳过无效计算（i={i}）：daishuji={daishuji:.4f}, jiankangma={jiankangma:.4f}")
                continue

        value = 1e7 * (1/2) * 1 / (x[i+1] * math.sqrt(daishuji) - x[i] * math.sqrt(jiankangma))
        if value < min_val:
                min_val = value
```

```
        if value > max_val:
            max_val = value
        value_pro += value
        print(f"x[{i}]={x[i]}, x[{i+1}]={x[i+1]} ->  计算值  = {value:.4f}")
        print(f"计算值  (i={i}): {value:.4f}")

num_valid = len(x) - 1 - 2
if num_valid > 0:
        final_thickness = (value_pro - min_val - max_val) / num_valid
        print(f"最终计算的薄膜厚度: {final_thickness:.2f} nm")
else:
        print("无有效计算值")

# 可视化
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(CCB, R, 'b-', label='Reflectance R(lambda)')
ax1.set_xlabel('Wavelength (nm)')
ax1.set_ylabel('Reflectance')
ax1.legend()
ax1.grid(True)

ax2.plot(CCB, n2, 'r-', label='Refractive index $n(lambda)')
ax2.plot(CCB, kappa, 'g--', label='Extinction coefficient kappa(lambda)$')
ax2.set_xlabel('Wavelength (nm)')
ax2.set_ylabel('Optical constants')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# 柯西色散验证部分
class CauchyDispersionValidator:

        # 柯西色散关系完整验证工具
        # 支持 2 参数(A,B)、3 参数(A,B,C)和 4 参数(A,B,C,D)模型
        # 针对碳化硅(4H-SiC)优化

        def __init__(self, wavelength_nm, refractive_index, kappa=None):
                self.wavelength_nm = np.array(wavelength_nm)
                self.refractive_index = np.array(refractive_index)
                self.kappa = np.array(kappa) if kappa is not None else None
                self.wavelength_um = self.wavelength_nm / 1000   # 转换为微米

                self.results = {
                        '2param': None,
                        '3param': None,
                        '4param': None
```

```
        }

    @staticmethod
    def _cauchy_2param(λ, A, B):
        return A + B / (λ**2 * 1e-6)

    @staticmethod
    def _cauchy_3param(λ, A, B, C):
        return A + B / (λ**2 * 1e-6) + C / (λ**4 * 1e-12)

    @staticmethod
    def _cauchy_4param(λ, A, B, C, D):
        return A + B / (λ**2 * 1e-6) + C / (λ**4 * 1e-12) + D / (λ**6 * 1e-18)

    def fit(self, max_order=4, transparency_threshold=0.01):
        if self.kappa is not None:
            mask = self.kappa < transparency_threshold
            λ_fit = self.wavelength_um[mask]
            n_fit = self.refractive_index[mask]
            print(f"使用透明区域数据点：{sum(mask)}/{len(mask)} (κ<{transparency_threshold})")
        else:
            λ_fit = self.wavelength_um
            n_fit = self.refractive_index

        # p0_2param = [2.553, 0.00735]  # 4H-SiC 初始值
        # p0_3param = [2.553, 0.00735, 0.0]
        # p0_4param = [2.553, 0.00735, 0.0, 0.0]
        p0_2param = [3.4228, 0.011078]  # 初始值
        p0_3param = [3.4228, 0.011078,0.00008]
        p0_4param = [3.4228, 0.011078,0.00008, 0.0]
        if max_order >= 2:
            try:
                popt, pcov = curve_fit(self._cauchy_2param, λ_fit, n_fit, p0=p0_2param,
                                       bounds=([1.0, 0.0], [5.0, 0.1]))
                perr = np.sqrt(np.diag(pcov))
                self.results['2param'] = {'coeffs': popt, 'errors': perr, 'model': self._cauchy_2param}
            except Exception as e:
                print(f"两参数拟合失败: {e}")

        if max_order >= 3:
            try:
                popt, pcov = curve_fit(self._cauchy_3param, λ_fit, n_fit, p0=p0_3param,
                                       bounds=([1.0, 0.0, -0.01], [5.0, 0.1, 0.01]))
                perr = np.sqrt(np.diag(pcov))
                self.results['3param'] = {'coeffs': popt, 'errors': perr, 'model':
```

```
self._cauchy_3param}
                except Exception as e:
                    print(f"三参数拟合失败: {e}")

            if max_order >= 4:
                try:
                    popt, pcov = curve_fit(self._cauchy_4param, λ_fit, n_fit,
p0=p0_4param,
                                           bounds=([1.0, 0.0, -0.01, -0.01], [3.0, 0.1,
0.01, 0.01]))
                    perr = np.sqrt(np.diag(pcov))
                    self.results['4param'] = {'coeffs': popt, 'errors': perr, 'model':
self._cauchy_4param}
                except Exception as e:
                    print(f"四参数拟合失败: {e}")

        def evaluate(self):
            if self.kappa is not None:
                mask = self.kappa < 0.01
                λ_eval = self.wavelength_um[mask]
                n_eval = self.refractive_index[mask]
            else:
                λ_eval = self.wavelength_um
                n_eval = self.refractive_index

            for order, result in self.results.items():
                if result is None:
                    continue
                n_pred = result['model'](λ_eval, *result['coeffs'])
                ss_res = np.sum((n_eval - n_pred)**2)
                ss_tot = np.sum((n_eval - np.mean(n_eval))**2)
                r_squared = 1 - (ss_res / ss_tot)
                residuals = n_eval - n_pred
                std_dev = np.std(residuals)
                result['r_squared'] = r_squared
                result['residual_std'] = std_dev

                print(f"\n{order}模型结果:")
                print(f"柯西系数: {result['coeffs']}")
                print(f"系数误差: {result['errors']}")
                print(f"R² = {r_squared:.6f}")
                print(f"残差标准差 = {std_dev:.6f}")

                # 模型好坏判断
                if r_squared > 0.95:
                    print("模型质量: 优秀 (R² > 0.95)")
                elif r_squared > 0.9:
                    print("模型质量: 良好 (0.9 ≤ R² ≤ 0.95)")
```

```
            elif r_squared > 0.8:
                print("模型质量: 一般 (0.8 ≤ R² < 0.9)")
            else:
                print("模型质量: 较差 (R² < 0.8)，建议检查数据或模型")

    def plot_results(self):
        plt.figure(figsize=(14, 6))
        ax1 = plt.subplot(1, 2, 1)
        if self.kappa is not None:
            sc = ax1.scatter(self.wavelength_nm, self.refractive_index,
                             c=self.kappa, cmap='viridis', vmin=0, vmax=0.05,
                             s=20, alpha=0.7, label='数据点(颜色=κ)')
            plt.colorbar(sc, label='消光系数 κ')
        else:
            ax1.scatter(self.wavelength_nm, self.refractive_index,
                        s=20, alpha=0.7, label='数据点')

        colors = ['red', 'blue', 'green']
        for i, (order, result) in enumerate(self.results.items()):
            if result is None:
                continue
            n_fit = result['model'](self.wavelength_um, *result['coeffs'])
            ax1.plot(self.wavelength_nm, n_fit,
                     color=colors[i], linewidth=2,
                     label=f'{order}模型 (R²={result["r_squared"]:.3f})')

        ax1.set_xlabel('波长 (nm)')
        ax1.set_ylabel('折射率 n')
        ax1.set_title('4H-SiC 柯西色散拟合结果')
        ax1.legend()
        ax1.grid(True, alpha=0.3)

        ax2 = plt.subplot(1, 2, 2)
        for i, (order, result) in enumerate(self.results.items()):
            if result is None:
                continue
            n_pred = result['model'](self.wavelength_um, *result['coeffs'])
            residuals = self.refractive_index - n_pred
            ax2.scatter(self.wavelength_nm, residuals,
                        color=colors[i], s=20, alpha=0.7,
                        label=f'{order}模型残差')

        ax2.axhline(0, color='black', linestyle='--')
        ax2.set_xlabel('波长 (nm)')
        ax2.set_ylabel('残差 (n_data - n_fit)')
        ax2.set_title('拟合残差分布')
        ax2.legend()
        ax2.grid(True, alpha=0.3)
```

```
        plt.tight_layout()
        plt.show()

validator1 = CauchyDispersionValidator(CCB, n1, kappa)
validator1.fit(max_order=3)
validator1.evaluate()
validator1.plot_results()

validator2 = CauchyDispersionValidator(CCB, n2)
validator2.fit(max_order=3)
validator2.evaluate()
validator2.plot_results()

if validator1.results['3param'] and validator2.results['3param']:
    coeffs1 = validator1.results['3param']['coeffs']
    coeffs2 = validator2.results['3param']['coeffs']
    r2_1 = validator1.results['3param']['r_squared']
    r2_2 = validator2.results['3param']['r_squared']

    diff_A = abs(coeffs1[0] - coeffs2[0]) / coeffs1[0] * 100 if coeffs1[0] != 0 else 0
    diff_B = abs(coeffs1[1] - coeffs2[1]) / abs(coeffs1[1]) * 100 if coeffs1[1] != 0 else 0
    print(f"A 系数相对差异: {diff_A:.2f}%")
    print(f"B 系数相对差异: {diff_B:.2f}%")
```

Q_3：

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import zhplot
# 加载数据
data = pd.read_excel('B 题\B 题\附件\附件 3.xlsx')
lmd = data['波数  (cm-1)'].values
rsl = data['反射率  (%)'].values
# mask = (lmd <= 2500)
mask = (lmd >= 1000) & (lmd <= lmd.max())
lmd = lmd[mask]
rsl = rsl[mask]

# 计算波数间隔
delta_lmd = np.mean(np.diff(lmd))
N = len(lmd)

# 执行 FFT
fft_result = np.fft.fft(rsl)
fft_dagoujiao5 = np.fft.fftdagoujiao5(N, d=delta_lmd)  # 频率单位：cm⁻¹

# 计算振幅和相位
i_love_you_so = np.abs(fft_result) / N * 2
```

```
i_love_you_so[0] /= 2   # DC 分量
phase = np.angle(fft_result)

# 选择前 5 个主要频率成分（按振幅排序）
dagoujiao1 = np.argsort(i_love_you_so[:N//2])[::-1]
top5_indices = dagoujiao1[1:2]

# 计算各成分的波数周期
print("前 5 个主要正弦成分的波数周期：")
print("=" * 60)
for i, idx in enumerate(top5_indices, 1):
    dagoujiao5 = abs(fft_dagoujiao5[idx])   # 频率 (cm⁻¹)
    period = 1 / dagoujiao5 if dagoujiao5 != 0 else np.inf   # 波数周期

    print(f"成分 {i}:")
    print(f"   频率: {dagoujiao5:.4f} cm⁻¹")
    print(f"   波数周期: {period:.4f} cm⁻¹")
    print(f"   振幅: {i_love_you_so[idx]:.4f}")
    print(f"   相位: {phase[idx]:.4f} rad")
    print("-" * 40)

# 重建各正弦成分
reconstructed_components = []
for i in top5_indices:
    dagoujiao5 = fft_dagoujiao5[i]
    dagoujiao4 = i_love_you_so[i]
    dagoujiao3 = phase[i]
   dagoujiao2 = dagoujiao4 * np.cos(2 * np.pi * dagoujiao5 * lmd + dagoujiao3)
    reconstructed_components.append((dagoujiao5, dagoujiao4,
dagoujiao3,dagoujiao2))

# 绘制 FFT 分析结果
plt.figure(figsize=(14, 10))

# 1. 原始反射率光谱
plt.subplot(3, 1, 1)
plt.plot(lmd, rsl, 'b-', label='原始信号', linewidth=1.5)
plt.title('原始反射率光谱')
plt.xlabel('波数 (cm⁻¹)')
plt.ylabel('反射率 (%)')
plt.grid(True, alpha=0.3)
plt.legend()

# 2. 主要正弦成分（前 5 个）
plt.subplot(3, 1, 2)
colors = ['red', 'blue', 'green', 'orange', 'purple']
for i, (dagoujiao5, dagoujiao4, dagoujiao3,dagoujiao2) in
```

```
enumerate(reconstructed_components):
    plt.plot(lmd,dagoujiao2, color=colors[i],
             label=f'频率: {abs(dagoujiao5):.2f} cm⁻¹, 振幅: {dagoujiao4:.2f}')
plt.title('主要正弦成分')
plt.xlabel('波数 (cm⁻¹)')
plt.ylabel('振幅')
plt.grid(True, alpha=0.3)
plt.legend()

# 3. 重建信号对比
plt.subplot(3, 1, 3)
reconstructed_signal = i_love_you_so[0]/2  # DC 分量
for _, _, _,dagoujiao2 in reconstructed_components:
    reconstructed_signal +=dagoujiao2

plt.plot(lmd, rsl, 'b-', alpha=0.5, label='原始信号', linewidth=2)
plt.plot(lmd, reconstructed_signal, 'r-', label='重建信号(前 5 成分)', linewidth=1.5)
plt.title('原始信号与重建信号对比')
plt.xlabel('波数 (cm⁻¹)')
plt.ylabel('反射率 (%)')
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()
plt.show()

for i, (dagoujiao5, dagoujiao4, dagoujiao3,dagoujiao2) in
enumerate(reconstructed_components, 1):
    # 计算导数寻找极值点
    derivative = -dagoujiao4 * 2 * np.pi * dagoujiao5 * np.sin(2 * np.pi * dagoujiao5 *
lmd + dagoujiao3)

    # 寻找导数为零的点（极值点）
    zero_c= np.where(np.diff(np.sign(derivative)))[0]

    # 筛选极大值点（导数从正变负）
    maxima_indices = []
    for idx in zero_crossings:
        if derivative[idx] > 0 and derivative[idx+1] < 0:
            maxima_indices.append(idx)

    # 获取极大值点的波数和振幅
    maxima_wavenumbers = lmd[maxima_indices]
    maxima_values =dagoujiao2[maxima_indices]

    print(f"\n 成分 {i} (频率={abs(dagoujiao5):.2f} cm⁻¹)的极大值点：")
    for j, (wavenumber, value) in enumerate(zip(maxima_wavenumbers,
```

```
maxima_values), 1):
            print(f"   极大值 {j}: 波数 = {wavenumber:.2f} cm⁻¹, 振幅 = {value:.4f}")

# 可视化各正弦成分的极大值点
plt.figure(figsize=(14, 8))
for i, (dagoujiao5, dagoujiao4, dagoujiao3,dagoujiao2) in
enumerate(reconstructed_components):
    # 重新计算极大值点
    derivative = -dagoujiao4 * 2 * np.pi * dagoujiao5 * np.sin(2 * np.pi * dagoujiao5 *
lmd + dagoujiao3)
    zero_crossings = np.where(np.diff(np.sign(derivative)))[0]
    maxima_indices = [idx for idx in zero_crossings
                      if derivative[idx] > 0 and derivative[idx+1] < 0]
    maxima_wavenumbers = lmd[maxima_indices]
    maxima_values =dagoujiao2[maxima_indices]

    # 绘制正弦曲线
    plt.plot(lmd,dagoujiao2, color=colors[i],
             label=f'成分 {i}: {abs(dagoujiao5):.2f} cm⁻¹', alpha=0.7)

    # 标记极大值点
    plt.scatter(maxima_wavenumbers, maxima_values, color=colors[i],
                s=80, marker='o', zorder=5)

    # 标注波数值
    for wavenumber, value in zip(maxima_wavenumbers, maxima_values):
        plt.annotate(f'{wavenumber:.1f}',
                     xy=(wavenumber, value),
                     xytext=(5, 5), textcoords='offset points',
                     fontsize=9, color=colors[i])

plt.title('正弦成分的极大值点标注')
plt.xlabel('波数 (cm⁻¹)')
plt.ylabel('振幅')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```

Q4：

```
import numpy as np
from scipy.integrate import quad
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import pandas as pd
import scipy.signal
import math
import zhplot
from scipy.optimize import curve_fit
```

```
# 定义入射角度（度）
哈气 = 15.0
耄耋 = math.radians(哈气)
入射平方 = math.sin(耄耋)**2

# 数据加载与预处理
data = pd.read_excel('B 题\B 题\附件\附件 3.xlsx')
lmd = data['波数 (cm-1)'].values
rsl = data['反射率 (%)'].values
mask = (lmd >= 1200) & (lmd <= lmd.max())
lmd = lmd[mask]
rsl = rsl[mask]
# 选择分析范围并归一化反射率
R = np.clip(rsl / 100, 1e-6, 0.999)  # 确保 R 在(0,1)范围内

# 单位转换：波数(cm⁻¹) → 波长(nm) → 角频率(rad/s)
CCB = 1e7 / lmd  # 波数(cm⁻¹)转波长(nm): λ = 1e7/ν̃
c = 2.99792458e8  # 光速(m/s)
omg = 2 * np.pi * c / (CCB * 1e-9)  # ω = 2πc/λ

# 按ω升序重新排序所有数据
sort_idx = np.argsort(omg)
omg = omg[sort_idx]
R = R[sort_idx]
CCB = CCB[sort_idx]

# 创建反射率的插值函数（用于 KK 积分）
diangun = interp1d(omg, R, kind='cubic', bounds_error=False, fill_value=0)

# 快速 KK 积分计算相位θ(ω)（使用 Hilbert 变换近似）
luguan = np.log(np.sqrt(R))
theta = -np.imag(scipy.signal.hilbert(luguan))  # 希尔伯特变换
print(theta)

# 计算复折射率 n + iκ
sqrt_R = np.sqrt(R)
cos_theta = np.cos(theta)

n1 = (1 - R) / (1 + R - 2 * sqrt_R * cos_theta)
n2 = (1 + sqrt_R) / (1 - sqrt_R)

kappa = 2 * sqrt_R * np.sin(theta) / (1 + R - 2 * sqrt_R * cos_theta)
x = ['1393.80','1822.40','2251.00','2679.60','3108.21','3536.64','3950.46']
# x = ['1275.68', '1474.31', '1713.44', '1948.72', '2200.86', '2451.08', '2705.64', '2965.50',
'3216.20', '3466.90', '3709.41']
for i in range(len(x)):
    x[i] = eval(x[i])
```

```
# 使用 n1 计算厚度
nm = [0.0] * len(x)
for i, dingdongji in enumerate(x):
    dagoujiao = 1e7 / dingdongji
    idx = np.argmin(np.abs(CCB - dagoujiao))
    nm[i] = n1[idx]
    print(f"波长 {dagoujiao:.2f} nm (波数 {dingdongji} cm⁻¹) 的折射率 n = {nm[i]:.4f}")

value_pro = 0
min_val = float('inf')
max_val = float('-inf')
for i in range(0, len(x) - 1):
    daishuji = nm[i+1]**2 - 入射平方
    jiankangma = nm[i]**2 - 入射平方

    if daishuji < 0 or jiankangma < 0:
        print(f"跳过无效计算（i={i}）：daishuji={daishuji:.4f}, jiankangma={jiankangma:.4f}")
        continue

    value = 1e7 * (1/2) * 1 / (x[i+1] * math.sqrt(daishuji) - x[i] * math.sqrt(jiankangma))
    if value < min_val:
        min_val = value
    if value > max_val:
        max_val = value
    value_pro += value
    print(f"x[{i}]={x[i]}, x[{i+1}]={x[i+1]} -> 计算值 = {value:.4f}")
    print(f"计算值 (i={i}): {value:.4f}")

num_valid = len(x) - 1 - 2
if num_valid > 0:
    final_thickness = (value_pro - min_val - max_val) / num_valid
    print(f"最终计算的薄膜厚度: {final_thickness:.2f} nm")
else:
    print("无有效计算值")

# 可视化
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(CCB, R, 'b-', label='Reflectance R(lambda)')
ax1.set_xlabel('Wavelength (nm)')
ax1.set_ylabel('Reflectance')
ax1.legend()
ax1.grid(True)

ax2.plot(CCB, n1, 'r-', label='Refractive index $n(lambda)')
```

```
ax2.plot(CCB, kappa, 'g--', label='Extinction coefficient kappa(lambda)$')
ax2.set_xlabel('Wavelength (nm)')
ax2.set_ylabel('Optical constants')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# 使用 n2 计算厚度
nm = [0.0] * len(x)
for i, dingdongji in enumerate(x):
    dagoujiao = 1e7 / dingdongji
    idx = np.argmin(np.abs(CCB - dagoujiao))
    nm[i] = n2[idx]
    print(f"波长 {dagoujiao:.2f} nm (波数 {dingdongji} cm⁻¹) 的折射率 n = {nm[i]:.4f}")

value_pro = 0
min_val = float('inf')
max_val = float('-inf')
for i in range(0, len(x) - 1):
    daishuji = nm[i+1]**2 - 入射平方
    jiankangma = nm[i]**2 - 入射平方

    if daishuji < 0 or jiankangma < 0:
        print(f"跳过无效计算（i={i}）：daishuji={daishuji:.4f}, jiankangma={jiankangma:.4f}")
        continue

    value = 1e7 * (1/2) * 1 / (x[i+1] * math.sqrt(daishuji) - x[i] * math.sqrt(jiankangma))
    if value < min_val:
        min_val = value
    if value > max_val:
        max_val = value
    value_pro += value
    print(f"x[{i}]={x[i]}, x[{i+1}]={x[i+1]} -> 计算值 = {value:.4f}")
    print(f"计算值 (i={i}): {value:.4f}")

num_valid = len(x) - 1 - 2
if num_valid > 0:
    final_thickness = (value_pro - min_val - max_val) / num_valid
    print(f"最终计算的薄膜厚度: {final_thickness:.2f} nm")
else:
    print("无有效计算值")

# 可视化
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(CCB, R, 'b-', label='Reflectance R(lambda)')
ax1.set_xlabel('Wavelength (nm)')
ax1.set_ylabel('Reflectance')
ax1.legend()
ax1.grid(True)

ax2.plot(CCB, n2, 'r-', label='Refractive index $n(lambda)')
ax2.plot(CCB, kappa, 'g--', label='Extinction coefficient kappa(lambda)$')
ax2.set_xlabel('Wavelength (nm)')
ax2.set_ylabel('Optical constants')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# 柯西色散验证部分
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

class SellmeierDispersionModel:
    """
    Sellmeier 色散模型 - 专门针对 4H-SiC 等半导体材料优化
    """

    def __init__(self, wavelength_nm, refractive_index):
        self.wavelength_nm = np.array(wavelength_nm)
        self.refractive_index = np.array(refractive_index)
        self.wavelength_um = self.wavelength_nm / 1000  # 转换为微米

        # 4H-SiC 的典型 Sellmeier 参数初始值（文献参考）
        self.initial_guess_3term = [6.7, 0.32, 0.109, 2.77, 46.9, 0.170]
        self.initial_guess_2term = [6.7, 0.32, 0.109, 2.77]

    @staticmethod
    def sellmeier_2term(λ, B1, C1, B2, C2):
        """
        2 项 Sellmeier 方程: n² = 1 + B1*λ²/(λ² - C1) + B2*λ²/(λ² - C2)
        """
        λ_um = λ
        return np.sqrt(1 + B1*λ_um**2/(λ_um**2 - C1) + B2*λ_um**2/(λ_um**2 -
C2))

    @staticmethod
    def sellmeier_3term(λ, B1, C1, B2, C2, B3, C3):
        """
        3 项 Sellmeier 方程: n² = 1 + B1*λ²/(λ² - C1) + B2*λ²/(λ² - C2) + B3*λ²/(λ² -
```

```
C3)
        """
        λ_um = λ
        return np.sqrt(1 + B1*λ_um**2/(λ_um**2 - C1) +
                       B2*λ_um**2/(λ_um**2 - C2) +
                       B3*λ_um**2/(λ_um**2 - C3))

    def fit(self, model_type='3term', bounds=None):
        """
        拟合 Sellmeier 方程
        model_type: '2term'  或  '3term'
        """
        if bounds is None:
            if model_type == '2term':
                bounds = ([0, 0, 0, 0], [10, 1, 10, 100])
            else:
                bounds = ([0, 0, 0, 0, 0, 0], [10, 1, 10, 100, 10, 100])

        try:
            if model_type == '2term':
                popt, pcov = curve_fit(self.sellmeier_2term, self.wavelength_um,
                                       self.refractive_index,
                                       p0=self.initial_guess_2term,
                                       bounds=bounds,
                                       maxfev=10000)
                model_func = self.sellmeier_2term
            else:
                popt, pcov = curve_fit(self.sellmeier_3term, self.wavelength_um,
                                       self.refractive_index,
                                       p0=self.initial_guess_3term,
                                       bounds=bounds,
                                       maxfev=10000)
                model_func = self.sellmeier_3term

            perr = np.sqrt(np.diag(pcov))

            # 计算拟合质量
            n_pred = model_func(self.wavelength_um, *popt)
            ss_res = np.sum((self.refractive_index - n_pred)**2)
            ss_tot = np.sum((self.refractive_index -
np.mean(self.refractive_index))**2)
            r_squared = 1 - (ss_res / ss_tot)
            residuals = self.refractive_index - n_pred
            std_dev = np.std(residuals)

            return {
                'coefficients': popt,
                'errors': perr,
                'r_squared': r_squared,
```

```
                    'residual_std': std_dev,
                    'model_func': model_func,
                    'residuals': residuals
                }

            except Exception as e:
                print(f"Sellmeier {model_type} 拟合失败: {e}")
                return None

        def evaluate_fit_quality(self, result):
            """评估拟合质量"""
            if result is None:
                return "拟合失败"

            r2 = result['r_squared']
            std = result['residual_std']

            quality = ""
            if r2 > 0.99:
                quality = "优秀"
            elif r2 > 0.95:
                quality = "良好"
            elif r2 > 0.90:
                quality = "一般"
            else:
                quality = "较差"



            # 拟合曲线
            if result_2term:
                n_fit_2term = result_2term['model_func'](self.wavelength_um,

*result_2term['coefficients'])
                ax1.plot(self.wavelength_nm, n_fit_2term, 'r-', linewidth=2,
                         label=f'2 项 Sellmeier (R²={result_2term["r_squared"]:.3f})')

            if result_3term:
                n_fit_3term = result_3term['model_func'](self.wavelength_um,

*result_3term['coefficients']
```