基于波动光学的外延层厚度计算模型

摘要

红外干涉法是一种无损伤测量方法，其原理基于外延层与衬底因掺杂浓度不同导致的折射率差异。红外光入射后，分别从外延层表面和衬底表面反射的两束光会产生干涉条纹。通过分析红外光谱的波长、外延层折射率及入射角等参数，即可确定外延层的厚度。

针对问题一，我们需建立确定外延层厚度的数学模型。首先，设出反射系数和透射系数，利用**复振幅法**计算出单光束干涉的**光强反射率**关于**反射系数**、**透射系数**、**相位差**的**函数**；接着，根据 Snell 定律、几何关系、三角函数诱导公式和光程的定义，求出**相位差**关于**入射角**、**波数**、**外延层折射率**的**函数**；最后，根据 **Fresnel 方程组**求出**反射系数**、**透射系数**关于**折射率**、**入射角**的**函数**，由 **Sellmeier 方程**给出折射率与波长的关系。将上述关系联立，最终构建出外延层厚度关于波数、外延层折射率和入射角的数学模型（下称“单光束模型”）。

针对问题二，我们将附表中的原始数据做成可视化曲线，用 Python 语言编写了基于数学模型的算法，通过调节模型算法中的参量来拟合附表数据曲线。当拟合偏差足够小时，此时的所调参量即所求数据，即 **h=11.71μm**。在结果可靠性分析方面，首先分析算法图像与附表数据的可视化图像拟合是否足够精确，即判定 R^2 是否足够接近 1。接着观察附表数据发现，在入射角分别为 10° 和 15° 时，在波数为 399.6747cm^{-1} 处**反射率均为 0**，即发生**完全相消干涉**，则两束光的振幅应相等，且可算它们的光程差，进而代入模型中计算外延层厚度。将此特殊点法的计算结果与通过模型算法得出的结果相比较。

针对问题三，由物理学中两束光干涉的必要条件推导出：在本题情境下，多光束干涉的必要条件为**光束入射角大于临界角，透射系数与反射系数不远小于 1**。仿照问题一的建模思路利用复振幅法得出多光束干涉下求外延层厚度的数学模型（下简称“多光束模型”），并与单光束模型相比较，认为多光束干涉对外延层厚度的精度影响为：**当样品表面和界面的反射率较高时，多光束干涉会使得干涉条纹的极值点（极大/极小）发生偏移**，从而导致根据单光束模型计算出的厚度产生系统误差。

将附件 3 和附件 4 中的波数和反射率分别代入单光束模型和多光束模型，得到两情境下的反射率-波数估计曲线，分别与原始数据的曲线作比较，分析得出硅晶圆片中**会**出现多光束干涉。对模型进行修正，最终得出硅外延层厚度计算的数学模型，使用与问题二相似的方法，得出硅外延层厚度为 **7.58μm**。

用同样的方法得到碳化硅外延层的多光束模型及算法，得到两情境下的反射率-波数估计曲线，分别与原始数据的曲线作比较，发现使用多光束模型得出的估计曲线与原始数据的曲线拟合程度略有提升，同时得出消除影响后的计算结果。由于精度提升效果并不显著，可知发生在碳化硅外延层上的红外光干涉主要为单光束干涉。

关键字：复振幅法　Fresnel 方程　Sellmeier 方程　单光束干涉　多光束干涉

一、 问题重述

1.1 问题背景

碳化硅作为一种新兴的第三代半导体材料，其优越的综合性能表现正在受到越来越多的关注。碳化硅外延层的厚度是外延材料的关键参数之一，因此制定一套科学、准确、可靠的碳化硅外延层厚度测试标准十分重要。

红外干涉法是外延层厚度测量的无损伤测量方法，其工作原理是，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，红外光入射到外延层后，一部分从外延层表面反射出来，另一部分从衬底表面反射回来，这两束光在一定条件下会产生干涉条纹。**可根据红外光谱的波长、外延层的折射率和红外光的入射角等参数确定外延层的厚度**。

通常外延层的折射率不是常数，它与掺杂载流子的浓度、红外光谱的波长等参数有关。

1.2 问题要求

问题 1 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，建立确定外延层厚度的数学模型。

问题 2 根据问题 1 的数学模型，设计确定外延层厚度的算法。根据提供的碳化硅晶圆片的光谱实测数据，给出计算结果，并分析结果的可靠性。

问题 3 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。请推导产生多光束干涉的必要条件，以及多光束干涉对外延层厚度计算精度可能产生的影响。

根据多光束干涉的必要条件，分析所提供硅晶圆片的测试结果是否出现多光束干涉，给出确定硅外延层厚度计算的数学模型和算法及相应计算结果。

若认为多光束干涉也会出现在碳化硅晶圆片的测试结果中，从而影响到碳化硅外延层厚度计算得到精度，设法消除其影响并给出消除影响后的计算结果。

二、 问题分析

2.1 问题一分析

对于问题一，我们认为可以建立外延层厚度关于反射率、入射角、外延层折射率的数学关系，由此解出外延层厚度。根据波动光学相关知识，光强反射率可用反射系数、透射系数和两束光的相位差表达，其中反射系数和透射系数可以通过 Fresnel 方程组由

折射率与入射角表达，相位差可由折射率、入射角、波数与外延层厚度表达。考虑到题目未提供外延层折射率随波数变化的关系式，我们引用 Sellmeier 方程，将折射率表达为关于波数和六个只与材料有关的常量的解析式，进而建立单光束模型。

2.2 问题二分析

对于问题二，考虑到附件提供数据量很大，我们用 Python 编写了基于单光束模型的算法，通过调节模型算法中的参量来拟合附表数据曲线。拟合偏差足够小时，此时所调参量即是所求数据。通过非线性最小二乘法得到反射率-波数拟合曲线与外延层厚度的计算结果，并与原始数据绘制曲线进行比较，以分析结果的可靠性。

2.3 问题三分析

对于问题三，分析题目情景，发现题中光束一定满足两束光干涉的必要条件（相位差恒定、传播方向一致、频率相同），那么出现多光束干涉的必要条件应为保证多束光的产生，即：外延层上表面的透射系数、外延层与衬底间的反射系数应足够高。为分析多光束干涉对外延层厚度计算精度可能产生的影响，分析可仿照问题一的建模思路，利用复振幅法建立多光束模型，并与单光束模型计算出的结果进行比较以分析其影响。

将附件 3 和附件 4 中数据代入多光束模型中，获得硅外延层厚度计算的数学模型。将该模型放入问题二编写的 Python 算法中可计算出硅晶圆外延层厚度。

为分析多光束干涉是否会出现在碳化硅晶圆片中，将附件 3 和附件 4 的波数和反射率据代入多光束模型进行拟合，若拟合曲线的回归系数比单光束模型拟合曲线的回归系数高，则说明碳化硅中存在多光束干涉，可用多光束模型消除其影响并给出计算结果。

三、 模型假设

为简化问题，本文做出以下假设：

- **假设一** 空气透射率足够接近 1，可近似为真空透射率。
- **假设二** 外延层折射率大于衬底折射率。
- **假设三** 红外光中 s 偏振光与 p 偏振光占比相等，可通过平方平均计算等效和光振幅透射系数、反射系数。

四、 符号说明

符号	说明	单位
h	外延层厚度	μm
$n_0 = 1$	近似空气折射率	/
n	外延层折射率	/
$\tilde{A}_0$	初振幅	/
δ	相位	rad
$e^{i\delta}$	相位因子	/
$r_{1/2}$	上/下介面振幅反射系数	/
$t_{1/2}$	上/下介面上表面振幅透射系数	/
$t'_{1/2}$	上/下介面下表面振幅透射系数	/
$R_{1/2}$	上/下介面功率反射系数	/
$\theta = 10°/15°$	l_1 的入射角	°
ϕ	l_1 的折射角	°

五、 问题一

问题一要求建立确定外延层厚度的数学模型。如图 1所示，首先根据 Fresnel 方程组，用外延层折射率和入射角表达反射系数和透射系数。接着根据 Snell 定律、几何关系、三角函数诱导公式和光程的定义，用外延层厚度、折射率、入射角、波数表达相位差。最后根据光强反射率的公式和复振幅法，用反射系数和透射系数、相位差表达光强反射率，最终得到光强反射率关于折射率、入射角和波数的函数，即单光束模型。

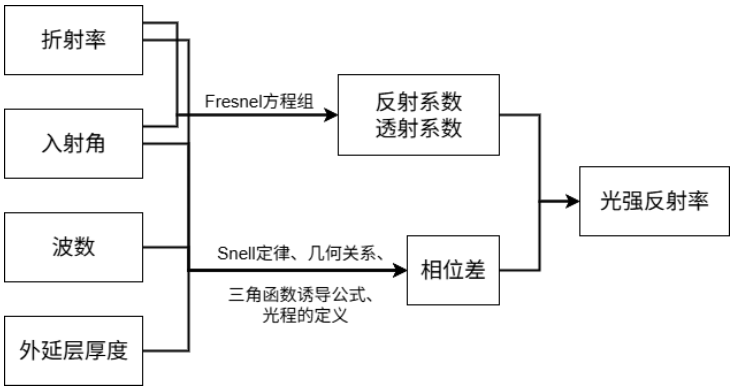


图 1　问题 1 流程图

5.1 光强反射率

仅考虑在 P 点由空气入射外延层，经过一次反射后从 P_1 处，自外延层透射入空气的光线 l_0 与 P_1 处反射光 l_1 的干涉（如图 2所示）。电磁波传播中每次透射、反射、行光，均应在复振幅上乘一对应因子。

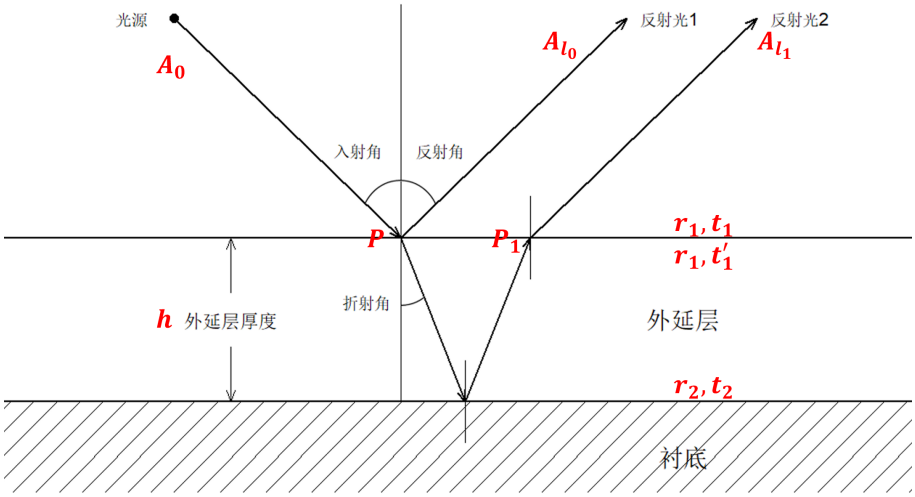


图 2　单束光干涉示意图

有 l_1 末复振幅:

$$\tilde{A}_{l_1} = \tilde{A}_0 t_1 t_1' r_2 e^{i\cdot 2\delta} \tag{1}$$

l_0 末复振幅:

$$\tilde{A}_{l_0} = \tilde{A}_0 e^{i\pi} r_1 \tag{2}$$

干涉后和振幅:

$$\tilde{A}_1 = \tilde{A}_0 \left(e^{i\pi} r_1 + t_1 t_1' r_2 e^{i\cdot 2\delta}\right) \tag{3}$$

由光强公式 $I = \tilde{A}\bar{\tilde{A}}$，得初末光强:

$$I_0 = \tilde{A}_0 \bar{\tilde{A}}_0 \tag{4}$$

$$I_1 = \tilde{A}_0 \bar{\tilde{A}}_0 \left(r_1^2 + t_1^2 t_1'^2 r_2^2 - 2 t_1 t_1' r_1 r_2 \cos(2\delta)\right) \tag{5}$$

得光强反射率:

$$R_1 = \frac{I_1}{I_0} = r_1^2 + t_1^2 t_1'^2 r_2^2 - 2 t_1 t_1' r_1 r_2 \cos(2\delta) \tag{6}$$

5.2 相位差

计算相位差 δ 与外延层厚度 h、入射角 θ、波数 ν、折射率 n 的关系，由 Snell 定律 $n_0 \sin\theta = n \sin\phi$ 及假设一:

$$\sin\phi = \frac{1}{n} \sin\theta \tag{7}$$

由几何关系及光程定义，l_1 自入射到衬底一次反射前的光程为：

$$S = \frac{nh}{\cos\phi} \tag{8}$$

引起相位差：

$$\delta = 2\pi S\nu = 2\pi\frac{nh}{\cos\phi}\nu \tag{9}$$

5.3 反射系数与透射系数

由 Fresnel 方程组，振幅反射系数、透射系数与折射率存在关系:

$$\begin{cases} r_p = \frac{n\cos\theta - n_0\cos\phi}{n\cos\theta + n_0\cos\phi} \\ t_p = \frac{2n_0\cos\theta}{n\cos\theta + n_0\cos\phi} \\ r_s = \frac{n_0\cos\theta - n\cos\phi}{n_0\cos\theta + n\cos\phi} \\ t_s = \frac{2n_0\cos\theta}{n_0\cos\theta + n\cos\phi} \end{cases} \tag{10}$$

其中 r_p, t_p 为 p 光的透射系数和反射系数，r_s, t_s 为 s 光的透射系数和反射系数。

由假设一、三：

$$\begin{cases} r_1 = \sqrt{\frac{1}{2}[\left(\frac{n\cos\theta - \cos\phi}{n\cos\theta + \cos\phi}\right)^2 + \left(\frac{\cos\theta - n\cos\phi}{\cos\theta + n\cos\phi}\right)^2]} \\ t_1 = \sqrt{\frac{1}{2}[\left(\frac{2\cos\theta}{n\cos\theta + \cos\phi}\right)^2 + \left(\frac{2\cos\theta}{\cos\theta + n\cos\phi}\right)^2]} \\ t_1' = \sqrt{\frac{1}{2}[\left(\frac{2n\cos\theta}{n\cos\theta + \cos\phi}\right)^2 + \left(\frac{2n\cos\theta}{\cos\theta + n\cos\phi}\right)^2]} \end{cases} \tag{11}$$

5.4 最终模型

联立 (6), (9), (11), 得到：

$$\begin{aligned} R_1 = & \frac{n^2\left(\cos^4\theta + \cos^4\phi\right) + \left(n^4 - 4n^2 + 1\right)\cos^2\theta\cos^2\phi}{\left(\cos\theta + n\cos\phi\right)^2\left(n\cos\theta + \cos\phi\right)^2} \\ & + 4n^2\cos^4\theta r_2^2\left[\frac{\left(1+n^2\right)\left(\cos^2\theta + \cos^2\phi\right) + 4n\cos\theta\cos\phi}{\left(\cos\theta + n\cos\phi\right)^2\left(n\cos\theta + \cos\phi\right)^2}\right]^2 \\ & + r_2\sqrt{\frac{\left(\frac{n\cos\theta - \cos\phi}{n\cos\theta + \cos\phi}\right)^2 + \left(\frac{\cos\theta - n\cos\phi}{\cos\theta + n\cos\phi}\right)^2}{2}}\cos\left(2\pi\frac{2nh}{\cos\phi}\nu\right) \\ & \cdot 4n\cos^2\theta\ \frac{\left(1+n^2\right)\left(\cos^2\theta + \cos^2\phi\right) + 4n\cos\theta\cos\phi}{\left(\cos\theta + n\cos\phi\right)^2\left(n\cos\theta + \cos\phi\right)^2} \end{aligned} \tag{12}$$

其中 $\cos\phi = \sqrt{1 - \left(\frac{1}{n}\sin\theta\right)^2}$.

由 Sellmeier 方程：

$$n^2 = 1 + \frac{B_1\lambda^2}{\lambda^2 - C_1^2} + \frac{B_2\lambda^2}{\lambda^2 - C_2^2} + \frac{B_3\lambda^2}{\lambda^2 - C_3^2} \tag{13}$$

其中 $\lambda = \frac{1}{\nu}$, $B_1, B_2, B_3, C_1, C_2, C_3$ 为仅与材料有关的常数。将 (13) 代入 (12) 后得到 $R_1(\theta, \nu)$. 根据题目提供的一系列 R_1, θ 数据可估计外延层厚度，此即确定外延层厚度的数学模型。

考虑到红外光束在到达外延层上界面时，存在光子与声子散射等损耗因素，引入损耗项 E_{lost}。损耗项只与外延层性质相关，故材料不变时，E_{lost} 应为常数。这一常量能损体现为入射波波数与反射波波数存在差值。由光子性质：

$$E = h\nu_t \tag{14}$$

可知：

$$E_{lost} = h\nu_{tlost} \tag{15}$$

考虑到（真空中）光速不变：

$$c = \lambda f = \frac{\nu_t}{\nu} \tag{16}$$

联立 (15) (16) 得：

$$\nu_{lost} = \frac{E_{lost}}{hc} \tag{17}$$

由此可知 ν_{lost} 也是常量。

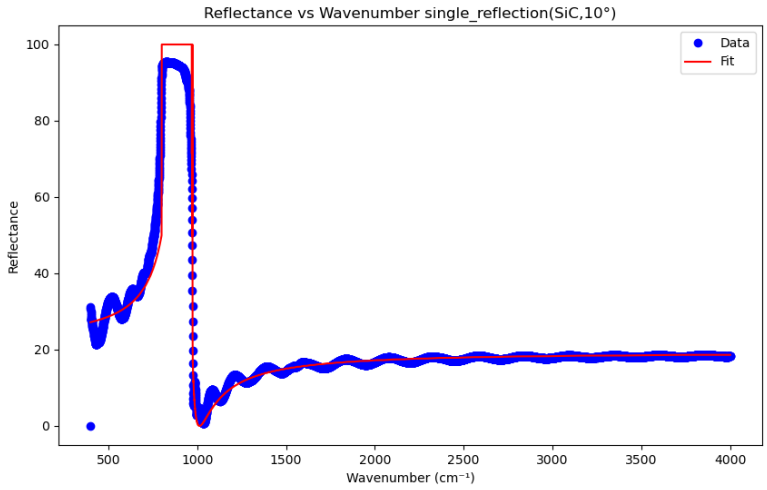
六、 问题二

6.1 基于 Python 的外延层厚度算法及结果

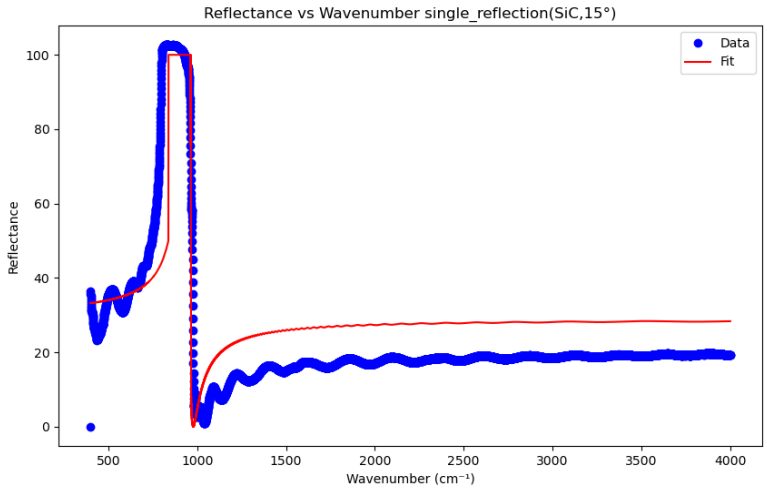
我们用 Python 编写了基于 (12) 和 (13) 的程序 `B2_fit_draw.py`，将入射角、波数与反射率的数据输入程序中，得到外延层厚度计算结果表 1及拟合曲线图 3。

表 1　单光束干涉下拟合 SiC 厚度

入射角	外延层厚度	R^2
10°	$11.7056871 \pm 0.04129513\,\mu m\,(0.35\%)$	0.95810968
15°	$9.65676289 \pm 0.07890127\,\mu m\,(0.82\%)$	0.63962779



(a) 单光束-SiC-10°



(b) 单光束-SiC-15°

图 3　单光束-SiC 拟合曲线

6.2 可靠性分析

由程序 `B2_fit_draw.py` 的运行结果可见，result1_1，result1_2 分别给出 $E_{lost} = 229.764446, E_{lost} = 228.067325$, 均在 229 左右，如分析是一常数，模型预测可靠。

分析图 3，$\theta = 10°$ 时拟合曲线与原始曲线十分贴合，$\theta = 15°$ 时拟合曲线走势与原始曲线相对一致，但在高度上与原始曲线有一定偏差。

由表 1，当 $\theta = 10°$ 时，$R^2 \approx 1$，拟合效果非常好，可靠性较强；当 $\theta = 15°$ 时 $R^2 > 0.6$，拟合效果较好，可靠性一般。据此，在单光束模型下计算出的碳化硅外延层厚度为：

$$h = 11.7056871 \pm 0.04129513\,\mu m\,(\,\frac{\Delta h}{h} \approx 0.35\%\,) \tag{18}$$

此外我们尝试不使用模型，而根据数据中的特殊点计算厚度，与模型给出厚度进行

比较和验证。

注意到附件 1 和附件 2 中，当波数 $\nu = 399.6747$ 时，$R_1 = 0$，代入 (6)：

$$R_1 = r_1^2 + t_1^2 t_1'^2 r_2^2 - 2t_1 t_1' r_1 r_2 \cos(2\delta) = 0$$

考虑到 $R_1 \geq 0$, 则 R_1 取到最小值，$\cos(2\delta) = 1$：

$$2\delta = 2k\pi (k \in Z)$$

与 (9) 联立，代入碳化硅材料估计值 $n = 2.6$, 入射角 $\theta = 10°$，得到

$$h = 9.94\,\mu m$$

所求出 h 值与算法所得出 h 值相近, 说明模型可靠。

七、 问题三的模型的建立和求解

问题三要求如下：

（1）推导产生多光束干涉的必要条件，研究多光束干涉对外延层厚度计算精度可能产生的影响。

（2）根据（1）分析硅晶圆片测试结果是否出现多光束干涉，并给出确定硅外延层厚度计算的数学模型、算法、计算结果。

（3）若认为多光束干涉也会出现在碳化硅晶圆片的测试结果中并影响其厚度计算的精度，试消除其影响并给出消除影响后的计算结果。

7.1 分析（1）

根据物理学定律，两束光相干的必要条件为：相位差恒定，频率相同，传播方向一致。分析本题多光束干涉情境，注意到这些光束的相位差和频率都是不变的，又根据反射定律和折射定律可知，它们的传播方向均一致。且为了产生干涉且效果明显，光束在外延层上表面不应发生全反射，且经多次透射和反射后出射的光强不应太小。故本题情境下产生多光束干涉的必要条件为：光的入射角大于临界角，透射系数与反射系数不远小于 1.

为分析多光束干涉对外延层厚度计算精度可能产生的影响，我们采用和问题一相同的思路，通过多光束干涉下光的反射率计算外延层厚度，并与单光束模型的计算结果进行比较。

如图 4，在多光束干涉下，设只经过空气-外延层介面一次反射的光复振幅为 A_{l_0}，从 P_n 透射出的光的复振幅为 A_{l_n}。

在 P 经一次反射的复振幅：

$$\tilde{A}_{l_0} = \tilde{A}_0 e^{i\pi} r_1 \tag{19}$$

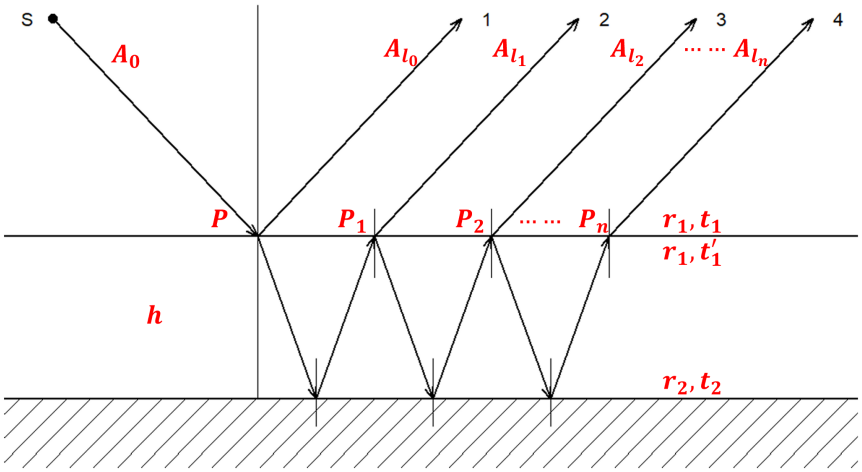


图 4　多光束干涉示意图

从 P_n 透射出的复振幅：

$$\tilde{A_{l_n}} = \tilde{A}_0 t_1 e^{i\cdot 2n\delta} r_2^n {r_1'}^{n-1} t_1' \tag{20}$$

多光束干涉后和振幅：

$$\begin{aligned}
\tilde{A}_2 &= \tilde{A}_{l_0} + \sum_{k=1}^{n} \tilde{A}_{l_k} \\
&= \tilde{A}_0 e^{i\pi} r_1 + \sum_{k=1}^{n} \tilde{A}_0 t_1 e^{i\cdot 2k\delta} r_2^k {r_1}^{k-1} t_1' \\
&= \tilde{A}_0 r_1 \left(-1 + \sum_{k=1}^{n} t_1 e^{i\cdot 2k\delta} r_2^k r_1^{k-2} t_1' \right) \\
&= \tilde{A}_0 r_1 \left(-1 + \frac{t_1 t_1'}{r_1^2} \frac{1}{1 - e^{i\cdot 2\delta} r_2 r_1} \right)
\end{aligned} \tag{21}$$

总光强：

$$I_2 = \tilde{A}_2 \bar{\tilde{A}}_2 = I_0 \cdot \frac{(t_1 t_1' - r_1^2)^2 + 2r_1^3 r_2 (t_1 t_1' - r_1^2)\cos(2\delta) + r_1^6 r_2^2}{r_1^2 (1 - 2r_1 r_2 \cos(2\delta) + r_1^2 r_2^2)} \tag{22}$$

得光强反射率：

$$R_2 = \frac{I_2}{I_0} = \frac{(t_1 t_1' - r_1^2)^2 + 2r_1^3 r_2 (t_1 t_1' - r_1^2)\cos(2\delta) + r_1^6 r_2^2}{r_1^2 (1 - 2r_1 r_2 \cos(2\delta) + r_1^2 r_2^2)} \tag{23}$$

联立 (9) (13) (23)，即得多光束模型：

$$\begin{cases}
R_2 = \frac{I_2}{I_0} = \frac{(t_1 t_1' - r_1^2)^2 + 2r_1^3 r_2 (t_1 t_1' - r_1^2)\cos(2\delta) + r_1^6 r_2^2}{r_1^2 (1 - 2r_1 r_2 \cos(2\delta) + r_1^2 r_2^2)} \\
\delta = 2\pi \frac{nh}{\cos\phi} \nu \\
n^2 = 1 + \frac{B_1 \lambda^2}{\lambda^2 - C_1^2} + \frac{B_2 \lambda^2}{\lambda^2 - C_2^2} + \frac{B_3 \lambda^2}{\lambda^2 - C_3^2}
\end{cases} \tag{24}$$

将此模型导入程序 `B3_fit.py` 中，并输入附件 3 与附件 4 的波数和反射率数据，得到拟合曲线，与附件 3 和附件 4 的原始数据曲线进行比较，以分析多光束干涉对外延层厚度计算精度可能产生的影响。

7.2 分析（2）

利用与问题二相类似的方法，将入射角与波数和反射率数据输入程序 `B3_fit.py` 中，得到外延层厚度计算结果表 2。

表 2　多光束干涉下拟合 Si 厚度

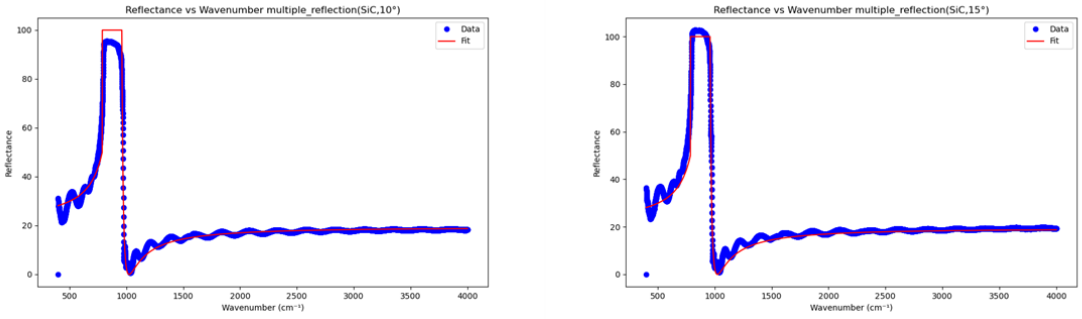
入射角	外延层厚度
10°	$7.57635469 \pm 0.06853746\,\mu m\,(0.90\%)$
15°	$6.23711871 \pm 0.05872850\,\mu m\,(0.94\%)$

7.3 分析（3）

将附件 1 和附件 2 的波数和反射率数据输入多光束模型，拟合出多光束模型下碳化硅的反射率-波数曲线。

表 3　多光束干涉下拟合 SiC 厚度

入射角	外延层厚度	R^2
10°	$9.58145550 \pm 0.02430286\,\mu m\,(0.25\%)$	0.97514798
15°	$9.34049335 \pm 0.01014973\,\mu m\,(0.11\%)$	0.98060959



(a) 多光束-SiC-10°　　**(b) 多光束-SiC-15°**

图 5　多光束-SiC 拟合曲线

为分析碳化硅中是否发生了多光束干涉，将表 1与表 3、图 3与图 5进行比较。

对比表 1与表 3中数据，我们发现，入射角相同时，多光束模型下的回归系数均略高于单光束模型下的回归系数；同时对比图 3与图 5，发现多光束模型的拟合效果更好。这说明碳化硅中存在多光束干涉，但其效应并不明显。

八、 模型的评价

8.1 模型的优点

- 直面延伸层材料的色散特性，通过引入 Sellmeier 方程, 给出了反射率-波数关系的完全模型
- 充分结合物理背景
- 模型可以较精准的拟合 SiC 不同入射角下的反射率-波数函数
- 最小二乘法拟合避免高频噪声
- 拟合值与特殊点估计值吻合
- 数据利用充分

8.2 模型的缺点

- 建模中引入较多层物理关系，反射率最终表达式较复杂
- 对 Si 材料的拟合较粗糙
- Sellmeier 方程引入了六个参数，使参数较多，存在过拟合可能

参考文献

[1] 钟锡华. 现代光学基础[M]. 北京: 北京大学出版社, 2003.

[2] 姜启源, 谢金星, 叶俊. 数学模型[M]. 第五版. 北京: 高等教育出版社, 2018.

附录 A　文件列表

文件名	功能描述
B2_fit_draw.py	问题二程序代码
B3_fit.py	问题三程序代码

附录 B　代码

B2_fit_draw.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lmfit import Model, Parameters
import os
TINY= 1e-10
def n_square(lam,B1,B2,B3,C1,C2,C3):
    return 1 + (B1 * lam**2) / (lam**2 - C1**2+TINY) + (B2 * lam**2) / (lam**2 - C2**2+TINY) + (B3 * lam**2) / (lam**2 - C3**2+TINY)

def n_val(lam,B1,B2,B3,C1,C2,C3):
    n_sq=n_square(lam,B1,B2,B3,C1,C2,C3)
    return np.maximum(n_sq,TINY)

def cos_phi(theta,n):
    inside_sqrt=1-(np.sin(theta)/n)**2
    inside_sqrt=np.maximum(inside_sqrt,TINY)
    return np.sqrt(inside_sqrt)

def delta_fun(h,n,theta):
    cos_theta = np.cos(theta)
    cos_theta = np.where(np.abs(cos_theta) < TINY, np.sign(cos_theta) * TINY, cos_theta)
    return 2 * np.pi * n * h / cos_theta

```

```
def r_p(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = n * cos_theta + phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    return (n * cos_theta - phi_term) / denominator

def r_s(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = cos_theta + n * phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    return (cos_theta-phi_term)/denominator

def r_1(n,theta):
    r_1_p=r_p(n,theta)
    r_1_s=r_s(n,theta)
    return np.sqrt((r_1_p**2+r_1_s**2)/2)

def t_p(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = n * cos_theta + phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    return (2 * cos_theta) / denominator

def t_s(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = cos_theta + n * phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
```

```
55      return (2 * cos_theta) / denominator
56
57  def t_p_prim(n,theta):
58      cos_theta = np.cos(theta)
59      phi_term = cos_phi(theta, n)
60      denominator = n * cos_theta + phi_term
61      denominator = np.where(np.abs(denominator) < TINY, np.sign
    (denominator) * TINY, denominator)
62      return (2 * n * cos_theta) / denominator
63
64  def t_s_prim(n,theta):
65      cos_theta = np.cos(theta)
66      phi_term = cos_phi(theta, n)
67      denominator = cos_theta + n * phi_term
68      denominator = np.where(np.abs(denominator) < TINY, np.sign
    (denominator) * TINY, denominator)
69      return (2 * n * cos_theta) / denominator
70
71  def t_1(n,theta):
72      t_1_p=t_p(n,theta)
73      t_1_s=t_s(n,theta)
74      return np.sqrt((t_1_p**2+t_1_s**2)/2)
75
76  def t_1_prim(n,theta):
77      t_1_prim_p=t_p_prim(n,theta)
78      t_1_prim_s=t_s_prim(n,theta)
79      return np.sqrt((t_1_prim_p**2+t_1_prim_s**2)/2)
80
81  def R_1(nu,B1,B2,B3,C1,C2,C3,h,r_2,theta,E_lost):
82      lam=1e4/(nu+E_lost)
83      n_val_here=n_val(lam,B1,B2,B3,C1,C2,C3)
84      r_1_here=r_1(n_val_here,theta)
85      t_1_here=t_1(n_val_here,theta)
86      t_1_prim_here=t_1(n_val_here,theta)
87      delta_here=delta_fun(h,n_val_here,theta)
```

```
    result = r_1_here**2 + (t_1_here * t_1_prim_here * r_2)**2 - 2 * t_1_here * t_1_prim_here * r_1_here * r_2 * np.cos(2 * delta_here)
    result = np.clip(result, 0, 1)
    return result*100

def R_2(nu,B1,B2,B3,C1,C2,C3,h,r_2,theta,E_lost):
    lam=1e4/(nu+E_lost)
    n_val_here=n_val(lam,B1,B2,B3,C1,C2,C3)
    r_1_here=r_1(n_val_here,theta)
    t_1_here=t_1(n_val_here,theta)
    t_1_prim_here=t_1(n_val_here,theta)
    delta_here=delta_fun(h,n_val_here,theta)
    denominator=r_1_here**2 *(1 - 2 * r_1_here * r_2*np.cos(2*delta_here)+r_1_here**2 * r_2**2)
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    result = ((t_1_here* t_1_prim_here - r_1_here ** 2) ** 2 +2 * r_1_here**3 * r_2 * ( t_1_here * t_1_prim_here - r_1_here ** 2 ) *np.cos(2*delta_here) + r_1_here**6 * r_2 **2)/denominator
    result = np.clip(result, 0, 1)
    return result*100

current_dir = os.path.dirname(os.path.abspath(__file__)) if '__file__' in globals() else os.getcwd()
file1 = os.path.join(current_dir, "附件1.xlsx")
file2 = os.path.join(current_dir, "附件2.xlsx")

data1 = pd.read_excel(file1, header=None, skiprows=1)
data2 = pd.read_excel(file2, header=None, skiprows=1)

data1.columns = [0, 1]  # "波数", "反射率"
data2.columns = [0, 1]  # "波数", "反射率"

```

```
params1 = Parameters()
params1.add('B1', value=6.52675310, min=0.1, max=10)
params1.add('B2', value=9.99984703, min=0.1, max=10)
params1.add('B3', value=1.86641317, min=0.01, max=2)
params1.add('C1', value=0.01124314 ,min=0.01, max=2)
params1.add('C2', value=151.569144, min=100, max=2000)
params1.add('C3', value=9.63176448, min=0.01, max=10)
params1.add('r_2', value=0.01589025,min=0.001, max=1)
params1.add('h', value=9.6,min=5, max=12)
params1.add('E_lost', value=200,min=170, max=230)

theta1=np.radians(10)
theta2=np.radians(15)

model1 = Model(R_1, independent_vars=['nu', 'theta'])
model2 = Model(R_2, independent_vars=['nu', 'theta'])

#result_附录n_方法1/2

 #一次折射SiC

#附录一10
result1_1 = model1.fit(data1[1], params1, nu=data1[0], theta=
   theta1,nan_policy='omit')
print(result1_1.fit_report())

plt.figure(figsize=(10, 6))
plt.plot(data1[0], data1[1], 'bo', label='Data')
plt.plot(data1[0], result1_1.best_fit, 'r-', label='Fit')
plt.xlabel('Wavenumber (cm□¹)')
plt.ylabel('Reflectance')
plt.title('Reflectance vs Wavenumber single_reflection(SiC,10°
   )')
plt.legend()
plt.show()
```

```
#附录二15
initial_params = params1.valuesdict()
test_result = R_1(data2[0].iloc[0], **initial_params, theta=
    theta2)

result2_1 = model1.fit(data2[1], params1, nu=data2[0], theta=
    theta2,nan_policy='omit')
print(result2_1.fit_report())

plt.figure(figsize=(10, 6))
plt.plot(data2[0], data2[1], 'bo', label='Data')
plt.plot(data2[0], result2_1.best_fit, 'r-', label='Fit')
plt.xlabel('Wavenumber (cm□¹)')
plt.ylabel('Reflectance')
plt.title('Reflectance vs Wavenumber single_reflection(SiC,15°
    )')
plt.legend()
plt.show()
#n次折射SiC

#附录一10
result1_2 = model2.fit(data1[1], params1, nu=data1[0], theta=
    theta1,nan_policy='omit')
print(result1_2.fit_report())

plt.figure(figsize=(10, 6))
plt.plot(data1[0], data1[1], 'bo', label='Data')
plt.plot(data1[0], result1_2.best_fit, 'r-', label='Fit')
plt.xlabel('Wavenumber (cm□¹)')
plt.ylabel('Reflectance')
plt.title('Reflectance vs Wavenumber multiple_reflection(SiC
    ,10°)')
plt.legend()
plt.show()

```

```
#附录二15
result2_2 = model2.fit(data2[1], params1, nu=data2[0], theta=
   theta1,nan_policy='omit')
print(result2_2.fit_report())

plt.figure(figsize=(10, 6))
plt.plot(data2[0], data2[1], 'bo', label='Data')
plt.plot(data2[0], result1_2.best_fit, 'r-', label='Fit')
plt.xlabel('Wavenumber (cm□¹)')
plt.ylabel('Reflectance')
plt.title('Reflectance vs Wavenumber multiple_reflection(SiC
   ,15°)')
plt.legend()
plt.show()
```

B3_fit.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lmfit import Model, Parameters
import os
TINY= 1e-10
def n_square(lam,B1,B2,B3,C1,C2,C3):
    return 1 + (B1 * lam**2) / (lam**2 - C1**2+TINY) + (B2 *
   lam**2) / (lam**2 - C2**2+TINY) + (B3 * lam**2) / (lam**2 -
   C3**2+TINY)

def n_val(lam,B1,B2,B3,C1,C2,C3):
    n_sq=n_square(lam,B1,B2,B3,C1,C2,C3)
    return np.maximum(n_sq,TINY)

def cos_phi(theta,n):
    inside_sqrt=1-(np.sin(theta)/n)**2
    inside_sqrt=np.maximum(inside_sqrt,TINY)
```

```
    return np.sqrt(inside_sqrt)

def delta_fun(h,n,theta):
    cos_theta = np.cos(theta)
    cos_theta = np.where(np.abs(cos_theta) < TINY, np.sign(cos_theta) * TINY, cos_theta)
    return 2 * np.pi * n * h / cos_theta

def r_p(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = n * cos_theta + phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    return (n * cos_theta - phi_term) / denominator

def r_s(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = cos_theta + n * phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
    return (cos_theta-phi_term)/denominator

def r_1(n,theta):
    r_1_p=r_p(n,theta)
    r_1_s=r_s(n,theta)
    return np.sqrt((r_1_p**2+r_1_s**2)/2)

def t_p(n,theta):
    cos_theta = np.cos(theta)
    phi_term = cos_phi(theta, n)
    denominator = n * cos_theta + phi_term
    denominator = np.where(np.abs(denominator) < TINY, np.sign(denominator) * TINY, denominator)
```

```
48      return (2 * cos_theta) / denominator
49
50  def t_s(n,theta):
51      cos_theta = np.cos(theta)
52      phi_term = cos_phi(theta, n)
53      denominator = cos_theta + n * phi_term
54      denominator = np.where(np.abs(denominator) < TINY, np.sign
    (denominator) * TINY, denominator)
55      return (2 * cos_theta) / denominator
56
57  def t_p_prim(n,theta):
58      cos_theta = np.cos(theta)
59      phi_term = cos_phi(theta, n)
60      denominator = n * cos_theta + phi_term
61      denominator = np.where(np.abs(denominator) < TINY, np.sign
    (denominator) * TINY, denominator)
62      return (2 * n * cos_theta) / denominator
63
64  def t_s_prim(n,theta):
65      cos_theta = np.cos(theta)
66      phi_term = cos_phi(theta, n)
67      denominator = cos_theta + n * phi_term
68      denominator = np.where(np.abs(denominator) < TINY, np.sign
    (denominator) * TINY, denominator)
69      return (2 * n * cos_theta) / denominator
70
71  def t_1(n,theta):
72      t_1_p=t_p(n,theta)
73      t_1_s=t_s(n,theta)
74      return np.sqrt((t_1_p**2+t_1_s**2)/2)
75
76  def t_1_prim(n,theta):
77      t_1_prim_p=t_p_prim(n,theta)
78      t_1_prim_s=t_s_prim(n,theta)
79      return np.sqrt((t_1_prim_p**2+t_1_prim_s**2)/2)
```

```
80
81  def R_1(nu,B1,B2,B3,C1,C2,C3,h,r_2,theta,E_lost):
82      lam=1e4/(nu+E_lost)
83      n_val_here=n_val(lam,B1,B2,B3,C1,C2,C3)
84      r_1_here=r_1(n_val_here,theta)
85      t_1_here=t_1(n_val_here,theta)
86      t_1_prim_here=t_1(n_val_here,theta)
87      delta_here=delta_fun(h,n_val_here,theta)
88      result = r_1_here**2 + (t_1_here * t_1_prim_here * r_2)**2
        - 2 * t_1_here * t_1_prim_here * r_1_here * r_2 * np.cos(2
       * delta_here)
89      result = np.clip(result, 0, 1)
90      return result*100
91
92  def R_2(nu,B1,B2,B3,C1,C2,C3,h,r_2,theta,E_lost):
93      lam=1e4/(nu+E_lost)
94      n_val_here=n_val(lam,B1,B2,B3,C1,C2,C3)
95      r_1_here=r_1(n_val_here,theta)
96      t_1_here=t_1(n_val_here,theta)
97      t_1_prim_here=t_1(n_val_here,theta)
98      delta_here=delta_fun(h,n_val_here,theta)
99      denominator=r_1_here**2 *(1 - 2 * r_1_here * r_2*np.cos(2*
        delta_here)+r_1_here**2 * r_2**2)
100     denominator = np.where(np.abs(denominator) < TINY, np.sign
       (denominator) * TINY, denominator)
101     result = ((t_1_here* t_1_prim_here - r_1_here ** 2) ** 2
       +2 * r_1_here**3 * r_2 * ( t_1_here * t_1_prim_here -
       r_1_here ** 2 ) *np.cos(2*delta_here) + r_1_here**6 * r_2
       **2)/denominator
102     result = np.clip(result, 0, 1)
103     return result*100
104
105 current_dir = os.path.dirname(os.path.abspath(__file__)) if '
       __file__' in globals() else os.getcwd()
106 file3 = os.path.join(current_dir, "附件3.xlsx")
```

```
file4 = os.path.join(current_dir, "附件4.xlsx")

data3 = pd.read_excel(file3, header=None, skiprows=1)
data4 = pd.read_excel(file4, header=None, skiprows=1)

data3.columns = [0, 1]  # "波数", "反射率"
data4.columns = [0, 1]  # "波数", "反射率"

params2 = Parameters()
params2.add('B1', value=12.0888185, min=5, max=15)
params2.add('B2', value=0.20727188, min=0.0001, max=2)
params2.add('B3', value=0, min=0, max=1,vary=False)
params2.add('C1', value=0.91733387 , min=0.01, max=2)
params2.add('C2', value=8.72601991, min=0.1, max=10)
params2.add('C3', value=1920.31207, min=1000, max=2000,vary=
   False)
params2.add('r_2', value=0.1, min=0.001, max=1)
params2.add('h', value=7.3, min=5, max=12)
params2.add('E_lost', value=200, min=170, max=230)

theta1=np.radians(10)
theta2=np.radians(15)

model1 = Model(R_1, independent_vars=['nu', 'theta'])
model2 = Model(R_2, independent_vars=['nu', 'theta'])

result3_1 = model1.fit(data3[1], params2, nu=data3[0], theta=
   theta1,nan_policy='omit')
print(result3_1.fit_report())

 #附录四15
result4_1 = model1.fit(data4[1], params2, nu=data4[0], theta=
   theta1,nan_policy='omit')
print(result4_1.fit_report())

```

```
#n次折射Si
#附录三10
result3_2 = model2.fit(data3[1], params2, nu=data3[0], theta=
   theta1,nan_policy='omit')
print(result3_2.fit_report())
#附录四15
result4_2 = model2.fit(data4[1], params2, nu=data4[0], theta=
   theta1,nan_policy='omit')
print(result4_2.fit_report())
```