浙江大学

课题题目 密立根油滴实验的数据处理

姓名学号 _____

_____

_____

_____

指导教师 _____

物理实验教学中心

2026 年 1 月

目录

1 背景知识

1.1 理论背景

1.1.1 密立根油滴实验及其原理

密立根油滴实验是由美国物理学家罗伯特·密立根于 1909 年进行的一项经典实验，用于测定电子的电荷量。实验的基本思路是通过观察油滴在电场中运动的情况，利用电场强度和油滴的下落速度来计算油滴所带的电荷量。通过实验，密立根成功地测量出电子的基本电荷，证实了电荷量的量子化。

实验中，油滴被喷入电场中，并通过观察其在电场中的下落速度与电场强度之间的关系，确定每个油滴所带的电荷量。实验的关键在于通过静电平衡测量油滴的电荷，利用不同电场条件下油滴的下落速度变化进行分析。密立根油滴实验是量子电荷测量的重要里程碑之一，并为后来的电子学和量子力学的发展奠定了基础。

1.1.2 电荷量量子化与电子的基本电荷

密立根油滴实验不仅确认了电子的基本电荷的存在，而且揭示了电荷量是量子化的，即电荷总是以某个基本单位为倍数存在，这一发现为量子力学的发展提供了重要支持。密立根通过多次实验得到了相同的电荷值，表明电子携带的电荷量是离散的，且其数值为1.602×10^{-19} 库仑，这就是今天所知的电子的基本电荷。

1.1.3 实验方法与技术

密立根油滴实验利用了一些当时尚处于发展中的技术，特别是在静电平衡的精密测量和油滴运动的观察方面。实验中，通过在油滴下落过程中施加已知强度的电场，借助高倍率显微镜和光学设备观察油滴的运动，并通过调节电场强度使油滴处于悬浮状态，从而准确测定其所带的电荷量。

随着技术的发展，现代版本的密立根油滴实验已经可以利用数字图像处理、自动化设备以及高速摄像技术来精确记录油滴的运动轨迹，并计算其电荷量,并且可以通过多种方法计算机自动计算得出结果。这些技术的引入大大提高了实验的精度，并使得实验结果更加可靠和可重复。

1.2 现实背景

密立根油滴实验作为大学物理教学中的经典实验，长期以来帮助学生通过观察油滴在电场中的运动，测定电子的基本电荷量。传统的实验方法虽然简单、有效，但在实际教学中存在一些挑战。例如，实验过程繁琐、操作精度要求高，学生往往需要通过反复“试错”来获得合适的油滴电荷量，这不仅消耗了大量的时间，还影响了实验结果的准确性。在此背景下，现代科技，特别是计算机视觉和自动化技术的引入，为解决这些问题提供了新的思路。

近年来，机器视觉技术在实验教学中的应用越来越广泛，尤其是在物理实验中，能够有效提升实验的自动化程度和精度。在密立根油滴实验中，结合机器视觉可以实时监测油滴的运动轨迹，自动识别油滴的位置和速度，减少人为操作的误差，并实时处理实验数据。这一技术不仅优化了实验操作流程，使得学生能够更加专注于理论分析和数据理解，还能通过图像处理和数据分析加速实验结果的生成和验证，从而提高了实验效率和教学质量。

通过结合现代机器视觉与传统的物理实验，学生不仅能够更加精准地进行实验操作，还能培养使用现代技术工具解决实际问题的能力，为未来的科研工作或技术创新打下坚实的基础。这一创新应用也为其他实验教学提供了新的思路和方法，推动了实验教学模式的改革和发展。

1.3 课题概述

本课题基于经典的密立根油滴实验，致力于通过创新的实验设计与数据处理技术，提升元电荷测量的准确性和实验效率。具体而言，课题包含两大主要内容：一是对已有的密立根油滴实验数据进行深入分析，通过改进的数据处理模型计算出元电荷量；二是引入视觉处理技术，以辅助学生在实验中更直观、更高效地进行油滴实验，优化实验操作与教学效果。

2 原理阐述

2.1 基本原理概述

密立根油滴实验是测定油滴带电量的实验，是在 1909 年由美国物理学家密立根（R.A.Millikan）设计并完成的。实验的结果表明所有带电物体的电荷都是不连续性变化的，其所带的电荷量都是某一最小电荷（基本电荷e）的整数倍，而且较为精确地测定了这一基本电荷e的数值。

密立根油滴实验的重要意义在于揭示了物质电结构的量子性，对人们认识组成物质的基本结构有决定性的作用，在近代物理学史上起到十分重要的作用。

2.1.1 基本实验原理

密立根油滴实验有三种实验测量方法：即静态平衡法、油滴反转运动法、动态测量法。本实验采用了静态平衡法，其基本原理如下：利用密立根油滴仪的喷雾器将油滴喷入两块相距为d的水平放置的平行带电平板之间，油滴在喷雾时由于摩擦，一般都是带电的。设油滴的质量为m，带电量为q，两块平行带电平板之间的电压为U，此时油滴在平板之间将同时受到两个力的作用，一个是重力mg，一个是静电力qE。调节板间的电压U，可使作用在油滴上的两个力达到动态平衡，则有：

$$mg = qE = q\frac{U}{d} \tag{1}$$

由上式可见，为了测出油滴所带的电量q，除了测出U和d之外，还需测定油滴的质量m。由于 m 很小（约10^{-15}kg），需要用特殊的方法来测定。

2.1.2 油滴质量m的测定

油滴在表面张力的作用下，一般总是呈小球状。设油的密度为ρ，某油滴的半径为 r ，则该油滴的质量m可用下式表示：

$$\mathrm{m}=\frac{4}{3}\pi \mathrm{r}^{3}\rho \tag{2}$$

平行板不加电压时，油滴受重力而加速下降，但由于空气对油滴的粘滞阻力 F 与油滴的速度 v 成正比，油滴下降一段距离达某一速度后阻力F与重力mg平衡（空气浮力忽略不计），油滴将匀速下降。由斯托克斯定律知：

$$\mathrm{F}=6\pi \mathrm{r}\eta \mathrm{v}=\mathrm{mg} \tag{3}$$

由(2)式和(3)式得油滴半径的大小为：

$$\mathrm{r}=\sqrt{\frac{9\eta \mathrm{v}}{2\rho \mathrm{g}}} \tag{4}$$

对于半径小到10^{-6}m 的小球，空气的粘滞系数应作修正，此时的斯托克斯公式应修正为：

$$\mathrm{F}=\frac{6\pi \mathrm{r}\eta \mathrm{v}}{1+\frac{\mathrm{b}}{\mathrm{pr}}} \tag{5}$$

式中b为修正常数，$b=6.17\times 10^{-6}$ m · cmHg， p为大气压强，单位为 cmHg 。根据修正后的粘滞阻力公式，得油滴半径为：

$$\mathrm{r}=\sqrt{\frac{9\eta \mathrm{v}}{2\rho \mathrm{g}}\frac{1}{1+\frac{\mathrm{b}}{\mathrm{pr}}}} \tag{5}$$

上式根号中还包含油滴的半径r，由于它处在修正项中，故不需十分精确，因此r用(4)式代入计算即可。

在(5)式中还有油滴匀速下降的速度v是未知数，它可用下面方法测出，即在平行板未加电压时，测出油滴下降 l 长度时所用的时间t，即：

$$\mathrm{v}=\frac{\mathrm{l}}{\mathrm{t}} \tag{6}$$

将(6)式代入(5)式，再代入(4)式，就算得油滴质量了。

2.1.3 油滴所带电荷量及基本电荷量

将(6)式代入(5)式，再代入(3)式，最后代入(1)式，整理后得：

$$q = \frac{18\pi}{\sqrt{2\rho g}}\left[\frac{\eta l}{t\left(1+\frac{b}{pr}\right)}\right]^{3/2}\frac{d}{U} \tag{7}$$

实验发现，对于某一颗油滴，如果我们改变它所带的电荷量q，则能够使油滴达到平衡的电压必须是某些特定值U_n，研究这些电压变化的规律可发现，它们都满足以下方程：

$$q = mg\frac{d}{U_n} = ne \tag{8}$$

式中$n = \pm 1,2,\ldots$，而e则是一个不变的值。

对于不同的油滴，可以发现有同样的规律，而且e值却是共同的常量。由此可见，所有带电油滴所带电荷量q都是最小电荷量e的整数倍，这就证明了电荷的不连续性，且最小电荷量e就是电子的电荷值：

$$e = q/n \tag{9}$$

2.2 机器视觉识别原理

2.2.1 电压识别模块（辅助“平衡法”）

目的：自动读取油滴平衡时施加的电压值 V，用于后续电荷计算。

实现原理：

1. 从摄像头画面中截取电压显示区域（ROI）；
2. 对该区域进行图像增强与二值化，提高数字清晰度；
3. 利用模板匹配（或更复杂的 OCR / 深度学习）识别三位数电压值；
4. 提供“电压会话”机制：学生按需触发识别，系统锁定有效电压并记录。

2.2.2 油滴轨迹与速度计算模块（辅助“动态法”）

目的：自动追踪油滴从上标记线到下标记线的运动过程，计算其终端速度 v_g，进而推算电荷量 q 和基本电荷倍数 n。

实现原理：

1. 使用模板匹配（或目标检测）在每帧画面中定位油滴中心；
2. 通过 X 坐标聚类（或轨迹关联）区分不同油滴，避免混淆；
3. 设定上下触发区域（对应已知物理距离 $L = 0.75mm$）；

4. 自动记录油滴进入上区和到达下区的帧号，结合帧率计算下落时间 t，得速度 $v_g = L/t$ ；

5. 结合已识别的电压 V 和密立根公式（含空气粘滞修正），自动计算 q 和 n；

6. 实时在画面上叠加显示速度、电荷数等信息，辅助学生判断油滴是否“可用”。

2.3 数据处理原理

2.3.1 已知元电荷实验原理（验证性实验）

在已知元电荷理论值（约为$e = 1.602 \times 10^{-19}\,C$）的前提下，密立根油滴实验的核心目标是对该基本物理常量进行实验验证。根据电荷量子化理论，任何带电油滴所携带的电荷量（ q ）均为元电荷（e）的整数倍，即：

$$q = ne \quad (n \in \mathbb{Z}^+)$$

实验中通过测量多个油滴的电荷值（ qi ），可利用如下方法反推元电荷：

1. 线性拟合法：将每个测量电荷 q_i 除以理论元电荷 e，四舍五入得到整数电荷数$n_i = round(q_i/e)$，再以n_i

为横坐标、q_i 为纵坐标进行线性拟合。拟合直线的斜率即为实验测得的元电荷值。该方法假设误差较小，适用于高精度数据。

2. 对数修正法：考虑到测量误差对线性拟合的影响，对电荷表达式取对数并引入修正因子，通过拟合 $\ln q_i$ 与$\ln n_i$ 的关系，减小大误差数据的权重，从而获得更稳健的元电荷估计值。

3. 此类方法本质上是在已知物理规律前提下的参数校准与误差分析，属于验证性科学实验，旨在检验实验装置的准确性和操作的规范性。

2.3.2 未知元电荷实验原理（探索性实验）

当元电荷值未知时，实验的目标转变为从大量电荷测量数据中发现电荷的量子化规律并推断其基本单位。这更贴近科学发现的原始过程，属于探索性或发现性实验。

其核心物理假设是：所有测量电荷均是某一最小不可分电荷单位（即元电荷）的整数倍，即存在（e）使得：

$$q_i = n_i e + \varepsilon_i, \quad n_i \in \mathbb{Z}^+, \quad \varepsilon_i \text{ 为实验误差}$$

为从数据中提取（e），可采用多种无先验假设的数据分析方法：

电荷分布分析：绘制电荷直方图，观察是否存在周期性峰值，初步估计（e）的大致范围（如 $1.4–1.8 \times 10^{-19}$ C）。

余数法 / 多重差值法：利用电荷之间的差值或余数，寻找最小公共单位。

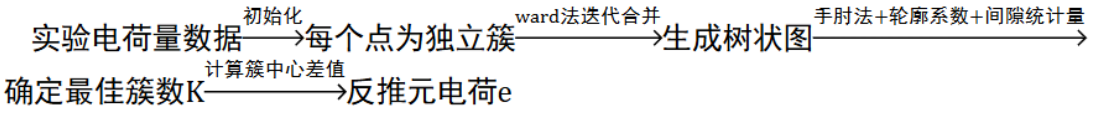
枚举方差法 / 残差法 / LAD 法：在合理区间内枚举候选（e）值，评估其对数据的拟合优度（如方差最小、残差最小、绝对偏差和最小等）。

最大公约数法：通过电荷比值与有理数比的匹配，统计最可能的公约数。

层次聚类法：将每个电荷视为数据点，通过聚类识别出若干“电荷层级”，相邻聚类中心的间距即为元电荷的估计值。

这些方法不依赖已知的（e）值，而是完全基于数据驱动，通过统计规律、数值优化或模式识别来揭示电荷的离散性和基本单位，体现了“从现象归纳物理规律”的科学探究逻辑。

2.3.3 层次聚类实验原理



1. 初始化 —— 每个数据点都是独立簇

密立根实验中，我们会测得一组油滴电荷量数据 $Q=\{q_1,q_2,...,q_m\}$（由公式 $q=Umgd$ 计算得到）。算法初始化时，将每个电荷量数据点视为一个独立的簇，即初始簇数为 m（与数据点数量相同）。此时每个簇的中心就是数据点本身，簇内平方和（WCSS）为 0，聚类的紧致性达到极致，但完全没有分类意义。

2. 迭代合并 —— 以 “最小代价” 合并相似簇

这是层次聚类的核心操作，关键是定义“簇间距离”，决定哪两个簇应该合并。在油滴实验中，代码采用的是 Ward 法（最适合量子化离散数据），其原理如下：

1. 簇间距离定义：Ward 法将“簇间距离”定义为合并两个簇后，总组内平方和（WCSS）的增量。公式表达为：$\Delta WCSS(C_i,C_j)=WCSS(C_i\cup C_j)-WCSS(C_i)-WCSS(C_j)$其中 C_i,C_j 是待合并的两个簇，$C_i\cup C_j$ 是合并后的新簇。
2. 迭代合并规则：每次遍历所有簇，找到使 $\Delta WCSS$ 最小的两个簇进行合并 —— 这意味着合并后簇内数据的离散程度增加得最少，保证了簇的紧致性。
3. 生成树状图：每一次合并都会记录簇间距离和簇的组成，最终将所有合并过程绘制成树状图（Dendrogram）。树的高度代表簇间距离，分支点对应簇的合并顺序：距离越近的簇，在树状图中越早合并。

3. 量化评估 —— 用三种方法确定最佳聚类数 K

层次聚类本身不会给出“最佳簇数”，而是生成一个包含所有可能簇数的树状图。此时需要结合手肘法、轮廓系数、间隙统计量三种方法，从树状图中找到最符合物理意义的簇数 K（对应元电荷的整数倍数量），三种方法的协同作用如下：

1、第一步：用手肘法找“拐点”

原理：计算不同 K 值对应的总 WCSS，绘制 K−WCSS 曲线。当 K 等于真实电荷倍数数量时，WCSS 下降速率会突然变缓，出现“手肘点”。

作用：快速锁定 K 的大致范围 —— 比如曲线在 K=5 时出现拐点，说明最佳 K 大概率在 4~6 之间。

局限性：当数据噪声较大时，手肘点可能不明显，无法单独作为判断依据。

2、第二步：用轮廓系数做“精准筛选”

原理：计算不同 K 值对应的整体轮廓系数，选择系数最大的 K—— 该值代表簇内最紧致、簇间最分离的聚类结果。

作用：在肘法锁定的范围内，精准确定最佳 K。比如肘法锁定 K=4~6，轮廓系数在 K=5 时达到峰值，则 K=5 是更可靠的选择。

物理意义：当 K=5 时，聚类结果恰好对应 e,2e,3e,4e,5e 五个电荷量子化簇，此时同一簇内的电荷量数据因测量误差小幅波动，不同簇之间有明显的距离间隔。

3、第三步：用间隙统计量做“真实性验证”

原理：对比真实电荷量数据与同范围随机数据的 WCSS，计算间隙值 —— 间隙值最大的 K，代表聚类结构最“真实”（由电荷量子化驱动，而非随机误差）。

作用：排除“过度聚类”的风险。比如当 K=8 时，手肘法和轮廓系数可能也会出现局部最优，但间隙统计量会发现此时真实数据的 WCSS 与随机数据几乎无差异，证明这是过度划分的结果。

物理意义：电荷量子化是客观存在的物理规律，因此真实数据的聚类效果必然远优于随机数据；若间隙值低，则说明聚类结果不可信（可能是测量误差导致的虚假簇）。

4、第四步：融合三种方法的结果

若三种方法推荐的 K 一致，则直接采用；

若结果冲突（比如肘法推荐 K=5，轮廓系数推荐 K=4），则取众数；若无众数，则取中间值；

最后还要做“兜底验证”：确保每个簇至少有 2 个数据点，避免因簇内数据过少导致的统计偏差。

4. 物理意义提取 —— 从簇中心反推元电荷

确定最佳 K 后，算法从聚类结果中提取物理意义，核心步骤如下：

1、计算簇中心：将 K 个簇按电荷量从小到大排序，每个簇的中心为簇内所有数据的平均值，对应 $n_1e,n_2e,\ldots,n_Ke$（ni 为正整数）。

2、计算簇中心差值：计算相邻簇中心的差值 $\Delta q_i=q_{i+1}-q_i$，理论上这些差值都是元电荷 e 的整数倍。

3、离群值过滤与元电荷估计：用中位数绝对偏差（MAD）剔除误差导致的异常差值，再取剩余差值的中位数，即为元电荷 e 的最终估计值。

3 软件构架

本次实验主要选取了以下编程语言和库来实现对油滴视觉识别计算和数据处理程序的构建：

1、python 以及安装额外库：openpyxl,pandas,opencv-python,numpy,matplotlib.pyplot,scipy 等。

2、VB6.0

4 实验操作及结果

4.1 视觉辅助实验

4.1.1 具体操作

1. 连接密立根油滴仪器与电脑。
2. 启动"模版生成.py"(见附录)生成电压识别时所需要的模版（每次生成完会存档，在相似光照条件下不需要重新生成）.
3. 启动"主程序.py"(见附录)先寻找合适的油滴,此时按 v 键记录下此时显示电压.
4. 关闭横线显示,开始识别油滴速度,同时还会计算出其电荷数.
5. 实验同时会产生 json 格式的日志，可以用“日志读取.py”读取。

4.1.2 实验结果

我们比对了我们视觉辅助下的测量方法与传统的物理实验测量方法，分别进行了不同油滴的一系列测试与针对同一油滴的多次测试。

1、不同油滴的测试

对于不同油滴进行测试，得到电荷量 N 的计算结果并比较如下图 1，可见两种实验方法之间差距不是特别的大。

	U/V	N视觉	N原始	相对误差
1	207	8.67	8.65	0.23%
2	120	43.01	41.11	4.61%
3	135	3.52	2.77	27.21%
4	350	1.09	1.25	-12.59%
5	86	2.75	4.22	-34.91%

图 1 不同油滴测试下两种方法比较图

2、同一油滴多次测试

对于同一油滴进行多次两种方法的测试，得到每种方法平均所得的电荷量 N，再分别计算两种方法的μ_A，结果得出手动测量的不确定度反而更小一点。

	U/V	V视觉(mm/s)	N视觉	V手动测量(mm/s)	N手动测量
1	290	0.15	6.12	0.149	5.70
2	290	0.17	6.91	0.150	5.64
3	290	0.17	7.23	0.151	5.74
4	290	0.15	6.14	0.150	5.71
5	290	0.14	5.76	0.151	5.78
6	290	0.17	7.09	0.146	5.48
平均		0.16	6.54	0.150	5.67
不确定度			0.249		0.0427

图 2 不同油滴测试下两种方法比较图

4.2 已知元电荷的数据处理实验

在已知电荷分布的量子化和元电荷值$e = 1.602 \times 10^{-19}C$的情况下，对实验所得的电荷值直接分析计算元电荷的值是目前在实验教学中常用的手段，常见有逐差法、线性回归法等。此处选取线性拟合并绘图的方法处理得到元电荷的值，并使用对数法进一步修正元电荷的值。

4.2.1 线性拟合法

原理：理论上所有电荷值均为元电荷值的整数倍，考虑到实验测量中存在的误差可以将实验所得的电荷值表示为$q = ne + u$，q为所测得的电荷值，n为元电荷个数（$n = 1$，2，3 …），e为元电荷的值，u为实验总误差。

本方法直接将测量得到的电荷值除以已知的元电荷的值，并四舍五入得到油滴所带的元电荷个数$n = Abs\left(\frac{q}{e}\right)$，再反过来求出测得的电荷值在该情况下的元电荷的值$e'' = \frac{q}{n}$。实验测得的多组数据

可得到多组数对(n, e'')，以n为横坐标，e''为纵坐标绘图并进行线性拟合，由此计算所得的斜率即为整体实验所得到的元电荷的值。

步骤：

1、输入实验所得的电荷值

2、计算每一个电荷值的电荷个数n和计算所得元电荷值e''

3、绘图并线性拟合

4.2.2 对数法及其修正

原理：由于不同电荷测量时始终存在实验误差u，直接通过线性拟合的方法处理得到的e''可能会使得误差较大的数据得以保留，最终线性拟合时对整个图形的线性产生较大影响，因此需要对实验总误差u进行修正。

现对电荷表达式两边同时取对数：$\ln q = \ln e + \ln\left(n + \frac{u}{e}\right) = \ln e + \ln n + \ln\left(1 + \frac{u}{en}\right)$

由于$\frac{u}{en}$可近似为极小量，有$\ln\left(1 + \frac{u}{en}\right) \approx \frac{u}{en}$

为方便直线拟合，将$\frac{u}{en}$拆分为对$\ln e$和$\ln n$的修正：$\ln q = \ln e' + k\ln n$

$\ln e'$ 为对$\ln e$的修正，k为对$\ln n$的修正。

比较修正后的式子与原式整理有：$e' = \frac{q}{n^k}$; $e = kn^{k-1}e'$。 kn^{k-1}为修正因子。

步骤：

1、根据输入的电荷值计算出其对应的电荷数n

2、将$\ln q$与$\ln n$线性拟合

3、计算出每个电荷值对应的e'、修正因子和修正后的e

4.2.3 实验结果

考虑到上述两个方法均是在已知元电荷的情况下处理，其本质还是对元电荷值的一种验证性的处理，因此将两者通过 vb 整合为一个程序，下图 3 是在示例数据下整合程序的运算结果界面。可见对数法修正后的结果能够部分消除直接计算出的元电荷值的偏差，使得计算结果相对误差更低。

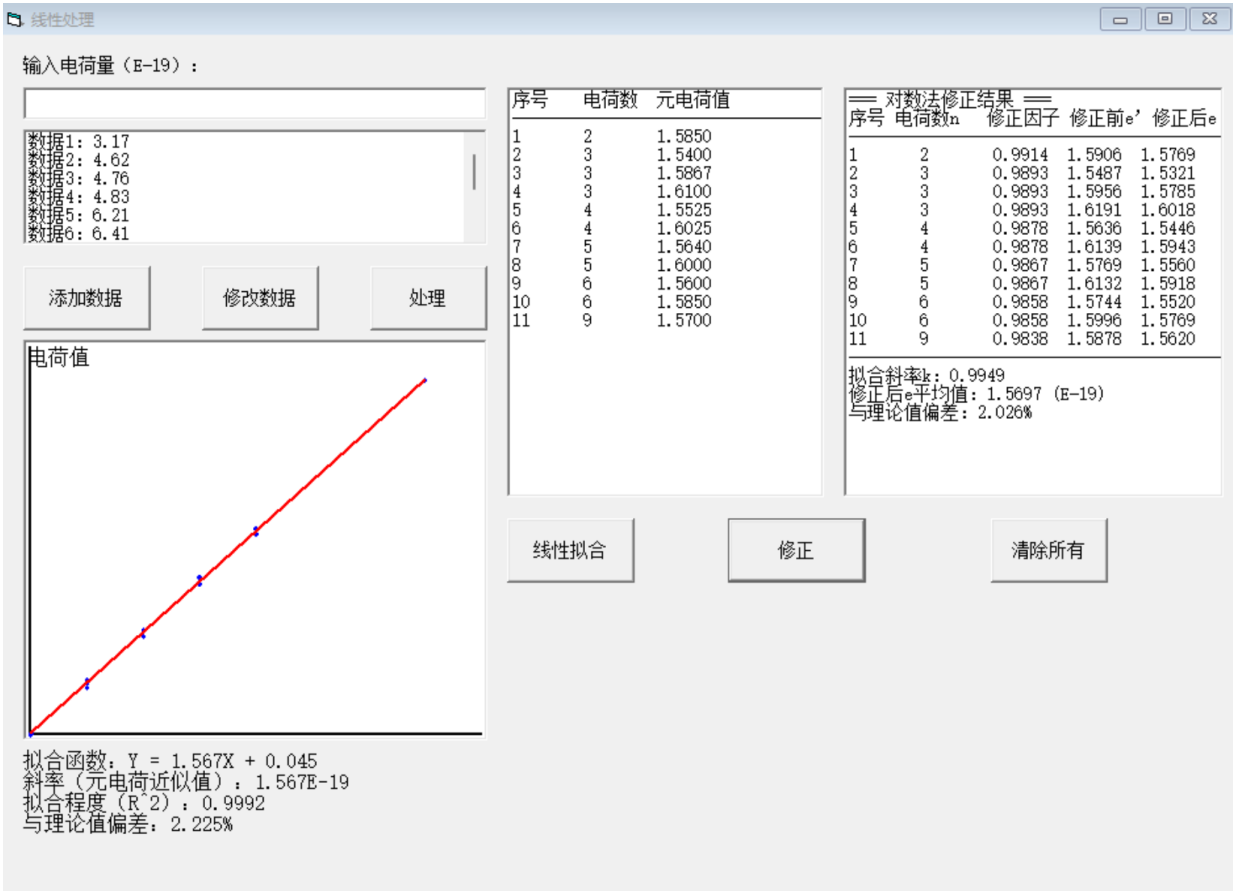


图 3 使用 VB6.0 制作的线性拟合和对数修正的整合程序

4.3 未知元电荷的数据处理实验

在已知元电荷的情况下，来探索实验值计算出的元电荷值，本质还是对元电荷值的“验证性”操作。而基于从物理现象到物理规律的科学探究逻辑，在未知元电荷值的情况下对大量实验电荷值数据统计、分析和探索处理方法，进而求得元电荷的值则是对元电荷值“发现性”的探索。这种从无到有的探索过程不仅能更深刻地理解元电荷的物理意义，也符合科学研究中“从现象归纳规律”的本质路径。因此本实验还从大量实验电荷数据出发，设计多种方法来探究电荷分布规律和元电荷值的计算。

4.3.1 电荷值的量子化分布

基于已知大量实验电荷值，筛选合适的电荷值后取电荷值在 $1\sim10\times10^{-19}C$的数据，再通过 Rstudio 绘图 4 如下。

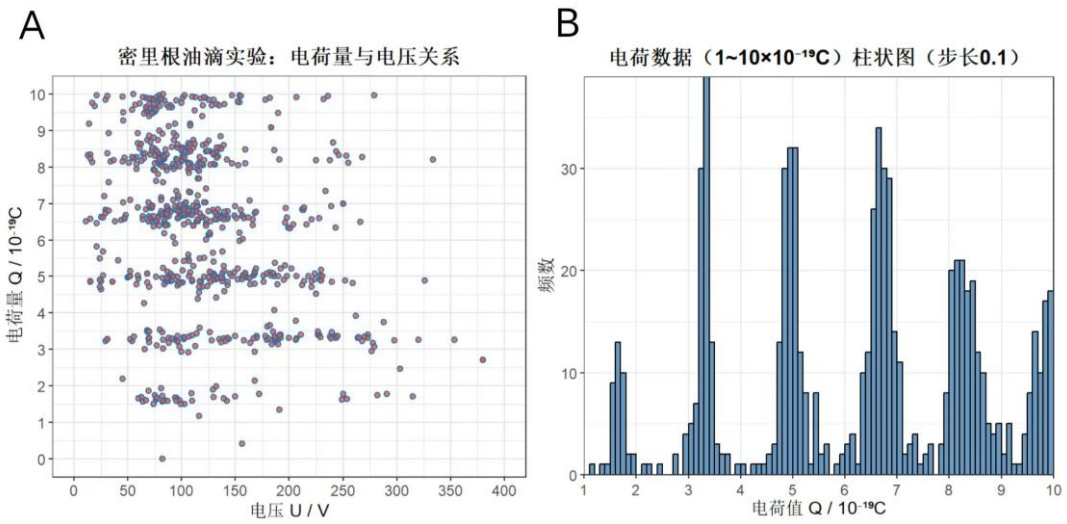


图 4 筛选后大量电荷值综合图

（A）电荷值与实验电压关系图　　（B）电荷值分类柱状图

图 A 以电压值为横坐标，实验电荷值为纵坐标绘图，可以观察到电荷值并非连续分布，而是呈现条带型的分层分布。因此再将这些数据从 1 开始，以 0.1 为步长绘制电荷数据的柱状图 B。明显观察到电荷的分布存在多个峰值，并且相邻峰值距离相近。结合上述大量数据作图分析，可以明确电荷值的分布并非连续分布，而是量子化分布，是最小组成值的整数倍。并且最小电荷值的分布都 1 到 2 之间，推测电荷值的最小组成单位（元电荷）在 1 到 2 之间，任意电荷值均可表示为：$q = ne + u$。下述方法将基于此发现处理电荷值来估计元电荷的值。

4.3.2 余数法

原理：在观察到电荷值的量子化分布后，两个电荷值之间可以用大的电荷值除以小的电荷值，得到的大于 1 的余数同样为某电荷值。因此可以将实验电荷值数据按大小排列，用大的电荷数据除以小于其的电荷值得到余数，考虑到误差经取舍后可以计算出最小电荷值，即元电荷的值。为避免多次余数造成的误差累积，设置以下运算规则：

1、第一次余数法处理时，所有数据均参与计算；后续处理时，1~2 之间的数据直接保留，不参与运算。

2、若存在 1~2 区间余数：仅保留该大数与所有小数运算后“1~2 区间的余数”。

3、若无 1~2 区间余数但有大于 2 的余数：仅保留该大数与所有小数运算后“大于 2 的余数”。

4、若余数全小于 1：直接舍去该大数，不保留任何余数。

步骤：

1、将输入电荷值自动排序

2、基于运算规则循环处理，最多处理 5 步（过多次处理导致误差累积无意义）

3、输入第几次处理，选取该次处理下 1 到 2 之间的值计算该次元电荷结果

如下是运用 vb 构建的余数法运算程序示例图 5：

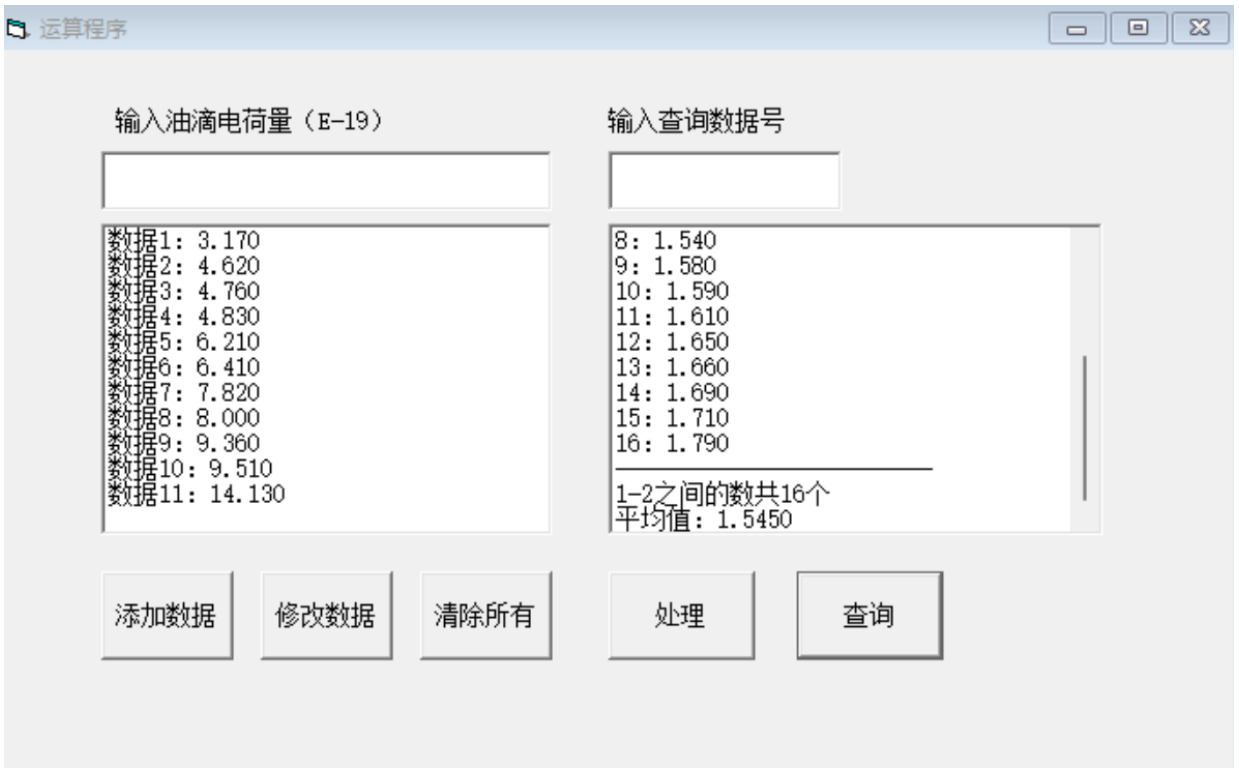


图 5 使用 VB6.0 构建的余数法运算程序图

根据结果可知，处理得到的元电荷值为: 1.5450×10^{-19}C，相对误差为：3.56%.

4.3.3 多重差值法

原理：如果存在一个固定的元电荷值，使得$q = ne$对于自然界一切电荷恒成立，那通过对各个电荷作差所得的差值应当出现在一个范围内富集的现象，通过对这些差值取平均便可以较好地估计元电荷值。多重差值法既保留了油滴电量逐项相减法简单直观而且易于得到电荷不连续性的优点，又克服了其一般情况下得到的基本电荷电量测量值偏小的缺点。

步骤：

1、读取文件，对数据按电荷量升序排列

2、遍历 Series a 中的所有数据，计算相邻两个数据的差值

3、使用循环结构重复以上操作，直至只剩一个差值

4、选取所有介于 1.4~1.8 的差值取平均，可以估计元电荷值

python 原代码见附录.

如下是运用 vb 构建的多重差值运算程序示例图 6：

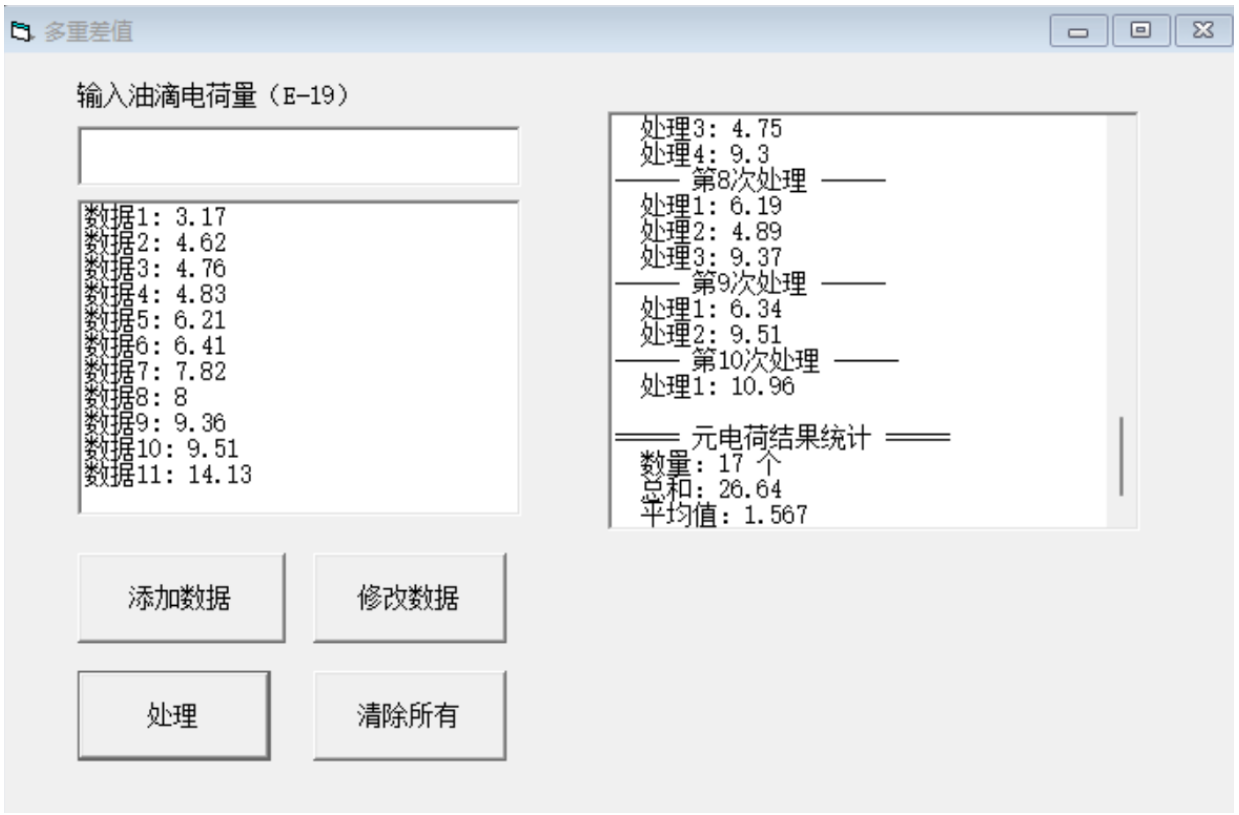


图 6 使用 VB6.0 构建的多重差值法运算程序图

处理步骤	0	1	2	3	4	5	6	7	8	9	10
	3.17	1.45	1.59	1.66	3.04	3.24	4.65	4.83	6.19	6.34	10.96
	4.62	0.14	0.21	1.59	1.79	3.2	3.38	4.74	4.89	9.51	
	4.76	0.07	1.45	1.65	3.06	3.24	4.6	4.75	9.37		
	4.83	1.38	1.58	2.99	3.17	4.53	4.68	9.3			
	6.21	0.2	1.61	1.79	3.15	3.3	7.92				
	6.41	1.41	1.59	2.95	3.1	7.72					
	7.82	0.18	1.54	1.69	6.31						
	8	1.36	1.51	6.13							
	9.36	0.15	4.77								
	9.51	4.62									
	14.1										
	处理后1~2之间的数据共17个，平均值为1.567										

图 7 多重差值法多次处理数据结果图

根据所有处理结果图 7，选择处理中处于 1 到 2 之间的数据（高亮部分）求和后再求均值，处理得到的元电荷值为: 1.567×10^{-19}C，相对误差为：2.18%.

4.3.4 残差法

原理：理想情况下，电荷值为$q = ne$，电荷值刚好是元电荷值的整数倍。考虑到元电荷的值在 1~2 之间，且实验所得的电荷值均存在误差，可以假设元电荷值为e_j，由此可以计算出该元电荷下电

荷的电荷数$n=\left[\frac{q}{e_j}+0.50\right]$（四舍五入取整），再以该电荷数和元电荷计算理想情况与实际情况的差值，记为残差$t_j=|q-ne_j|$，残差与此元电荷值的比值为$u_j=\frac{t_j}{e_j}$，若e_j越接近真实元电荷值，残差在元电荷中的占比应越小。因此选取 1~2 之间的系列数字作为e_j得出个u_j，绘制e_j-u_j图并选取标注最小的 3 个u_j进行观察，多组数据处理后最小值点最多富集的地方可估计出元电荷近似值的范围。考虑到e_j值小数点后位数过多可能存在整除的情况，由此得不到实际的元电荷值的范围，因此仅选取步长为 0.1 和 0.01 来得出系列e_j值，由此基本确定元电荷的值到小数点后两位。

步骤：

1、输入多个电荷值

2、选取 0.1 步长下，对每个电荷值处理计算u_j

3、在同一张图上绘制e_j-u_j图，观察大多极小值存在的点对应的横坐标

4、再选取步长 0.01，估计得到可能的元电荷值的范围

下图 8 和图 9 是在 vb 中残差法程序运算电荷示例结果，选取两个步长对同一组数据进行分析，多组数据比对可以明显观察到在 1.6 有极小值的聚集，因此初步得到元电荷的值在小数点后一位的精度下为$1.6\times10^{-19}C$：

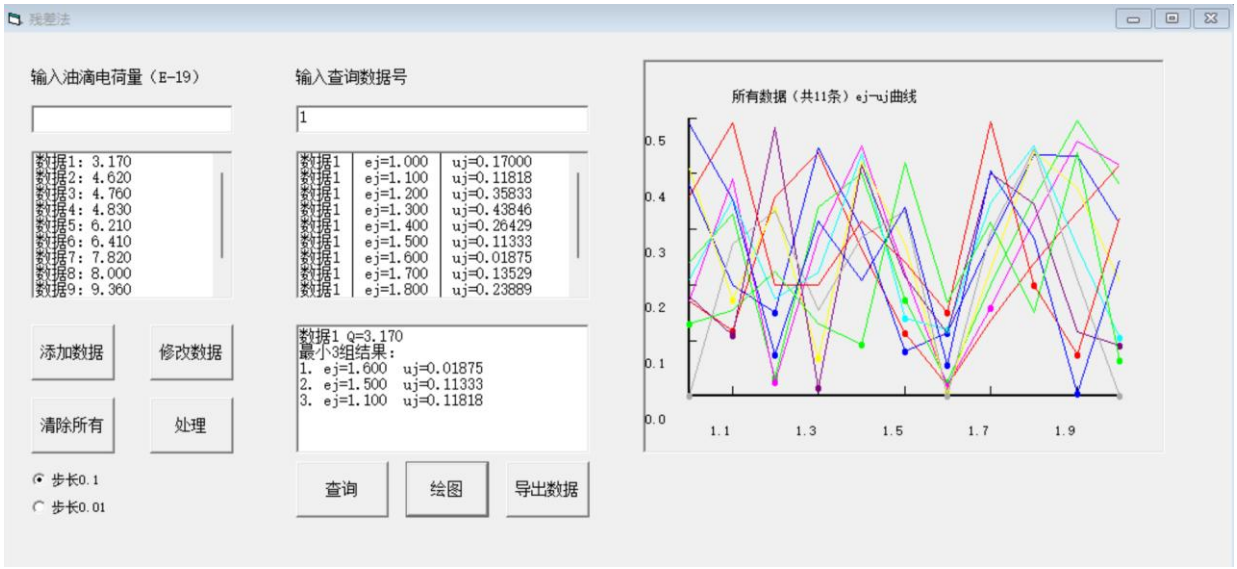


图 8 使用 VB6.0 构建的残差法运算程序图（步长 0.1）

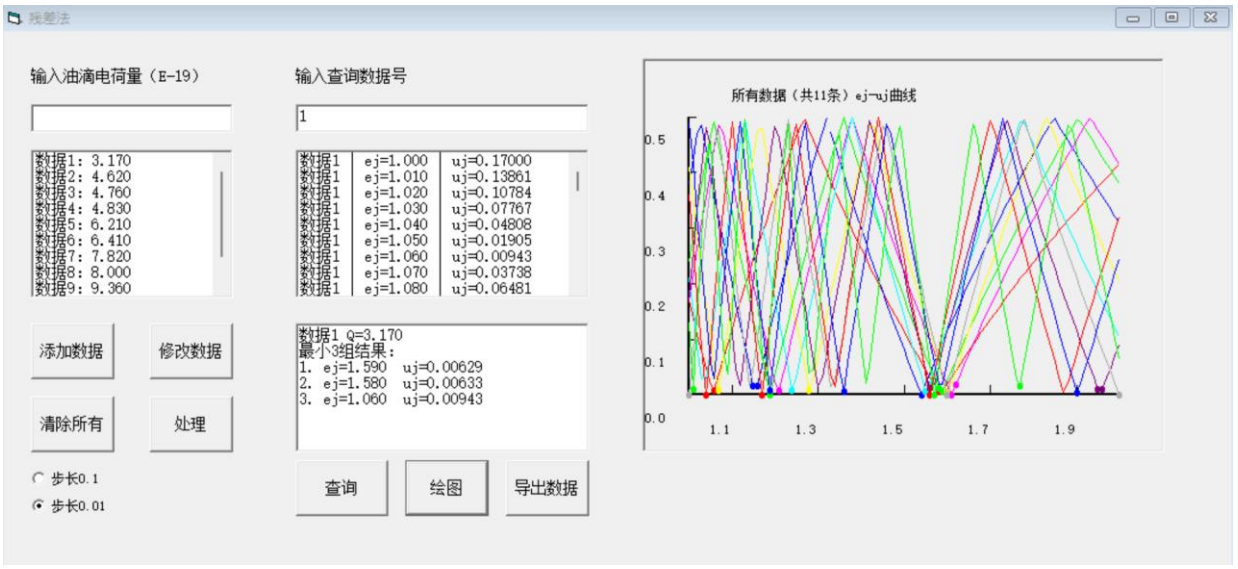
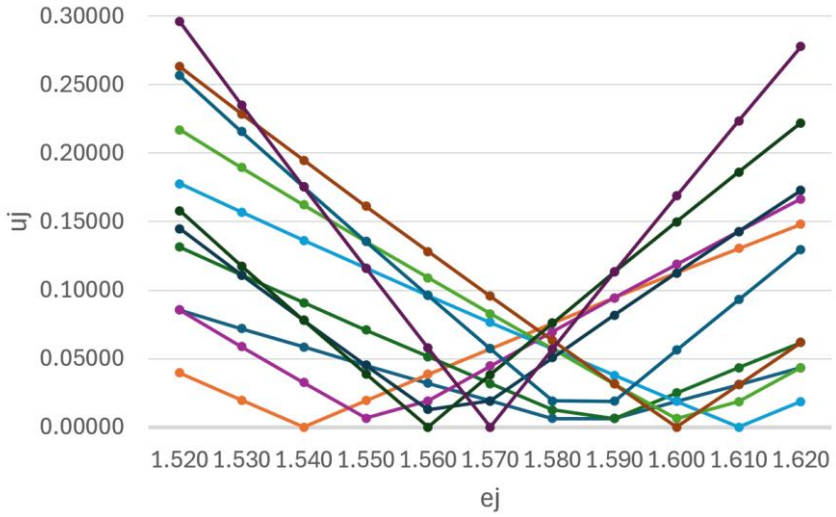


图 9 使用 VB6.0 构建的残差法运算程序图（步长 0.01）



原始电荷值	估算元电荷
3.17	1.60
4.62	1.54
4.76	1.59
4.83	1.61
6.21	1.55
6.41	1.60
7.82	1.56
8	1.60
9.36	1.56
9.51	1.59
14.13	1.57

平均估算元电荷：1.579

图 10 步长 0.01 处理后 1.52 至 1.62 间具体折线和数据图

在确定范围后可将步长为 0.01 下的计算结果导出，并选取每个数据在 1.52 到 1.62 之间数据，进一步详细做出折线和结果图 10，得到每个电荷值处理结果的极小值与对应的 ej 值，求平均值可知处理得到的元电荷值为: 1.579×10^{-19}C，相对误差为：1.44%.

4.3.5 最大公约数法

原理/步骤：假设在密立根油滴实验中测出n个油滴的电量，其实验测量数据分别为q_1、q_2、q_3、…、q_n。取任意两个数据的比值为：φ_{ij}

$$\varphi_{ij} = q_i/q_j \tag{1}$$

其中，i，j为下标$i \neq j$，且均为1到n的整数，这样我们将得到C_n^2组，即$(n \times (n-1)/2)$组比值。设正整数N为实验测得的油滴带的最大电荷数，将$1/N \sim N/1$范围内的所有正整数比记为

b_{km}，即

$$\mathrm{b_{km}} = \mathrm{k/m} \tag{2}$$

其中，k，m为下标，且均为$1\sim N$的整数，k和m可以是相同整数，也可以是不同整数。依次取φ_{ij}中的每个数据与整数比b_{km}逐个进行比较，φ_{ij}中每个数据均可在b_{km}中找到一个与之最为相近的值作为该数据的对应值,即认为两者相等。假设

$$\varphi_{\mathrm{ij}} = \mathrm{q_i/q_j} = \mathrm{b_{km}} = \mathrm{k/m} \tag{3}$$

从而有

$$\mathrm{e_{ik}} = \mathrm{q_i/k} \tag{4}$$

$$\mathrm{e_{jm}} = \mathrm{q_j/m} \tag{5}$$

例如$\varphi_{12} = 3.212/1.608$，$b_{21} = 2/1$，即认为$\varphi_{12} = b_{21}$，即可得到$e_{12} = q_1/1$，$e_{21} = q_2/2$。

利用以上方法，我们得到从q_1到q_n的所有数据两两相比的比对情况，即e_{ik}和e_{jm}。 对所有的e_{ik}和e_{jm}从小到大排序，将相差在 0.1%内的数据作为一组，从而得到多组数据。统计每组数据的数据个数，元电荷值应在数据个数最多的那组数据附近。将这组数据分别记为e_1、e_2、…、e_l。 对其求平均值，即可得到元电荷的实验值

$$\overline{\mathrm{e}} = (\mathrm{e_1} + \mathrm{e_2} + \mathrm{e_3} + \cdots + \mathrm{e_l})/\mathrm{l} \tag{6}$$

下图 11 是用 Python 绘制的最大公约数分布散点图，选取了 1.4~1.8 区间内的数据点，从中可以很清晰看到实验所得数据在 1.675 附近富集，这也是我们得到结果的依据。

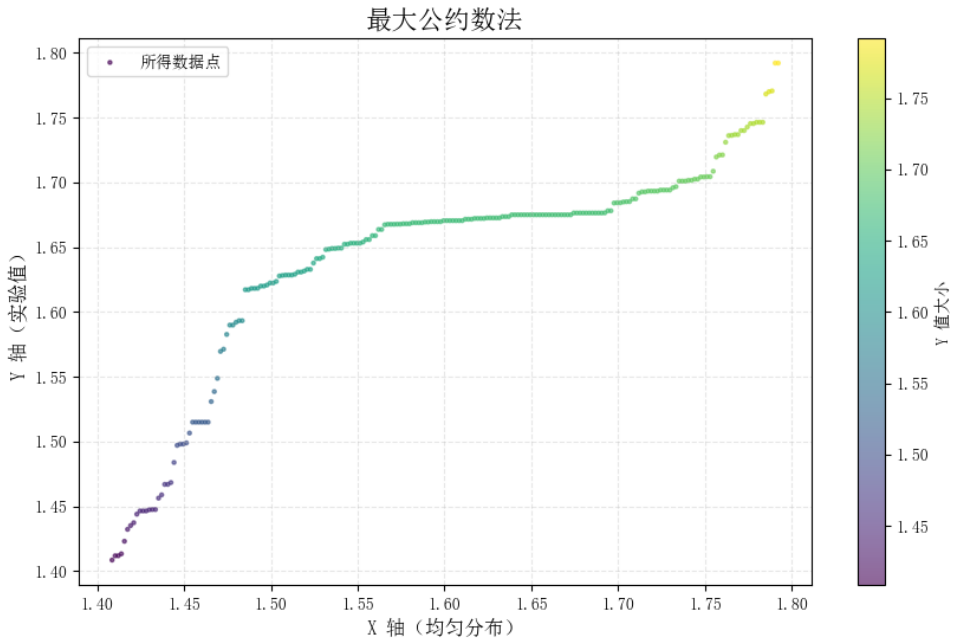


图 11 使用 Python 绘制的最大公约数法结果示意图

如下是用 Python 完成的最大公约数法运算程序结果示例图 12：

```
请输入Excel文件路径（例如：C:\data\file.xlsx）：12.xlsx
1.675
按任意键退出程序.....
```

图 12 使用 Python 构建的最大公约数法运算程序图

根据结果可知，处理得到的元电荷值为：1.675×10^{-19}C，相对误差为：4.56%.

原代码见附录.

4.3.6 LAD 法

原理：最小一乘法（Least Absolute Deviation）是回归分析中通过最小化实测值与预测值间绝对误差之和进行参数估计的方法，属于中位数回归范畴。该方法的核心在于寻找使各数据点到回归直线纵向距离绝对值之和最小的参数解，不要求随机误差服从正态分布，对异常值具有较强稳健性。本方法预设存在一条直线$q = ne$对大多数样本点满足，通过小梯度递增的方法遍历小范围的可行电荷量计算绝对误差之和，最终采用绝对误差之和最小的电荷量作为元电荷值，即$\min = \frac{\overline{|v_{ij}|}}{e_j}$。

步骤：

1、将n个电荷升序排列，找出最小电荷值Q_{min}，粗略估计ê的最小可能值不会小于$\frac{Q_{min}}{K}$，一般预设$K = 6$

2、用小倍率递增法选取扫描变量e_j，$Q_{min}/\sqrt{K(K+1)} \le e_j \le 1.25Q_{min}$，递增倍率$\eta = e_{j+1}/e_j = \left(\frac{1.25}{\sqrt{K(K+1)}}\right)^{\frac{1}{300}}$，j = 1,2, … ,301

3、求残差的绝对值的平均$\overline{|v_{ij}|}$，其中$\overline{v_{ij}} = \frac{1}{n}\sum_{i=1}^{n}\left|Q_i - e_j \cdot int\left(\frac{Q_i}{e_j} + 0.49\right)\right|$，其中$n_i = int(Q_i/e_j + 0.49)$是$Q_i$的电子数，LAD 法的原理就是使$\overline{|v_{ij}|}$为极小值

下图 13 是用 Python 绘制的 LAD 法$\frac{\overline{|v_{ij}|}}{e_j}$随扫描变量$e_j$变化的示意图，从中我们可以看到在 1.660 左右残差平均值与扫描变量之比最小。

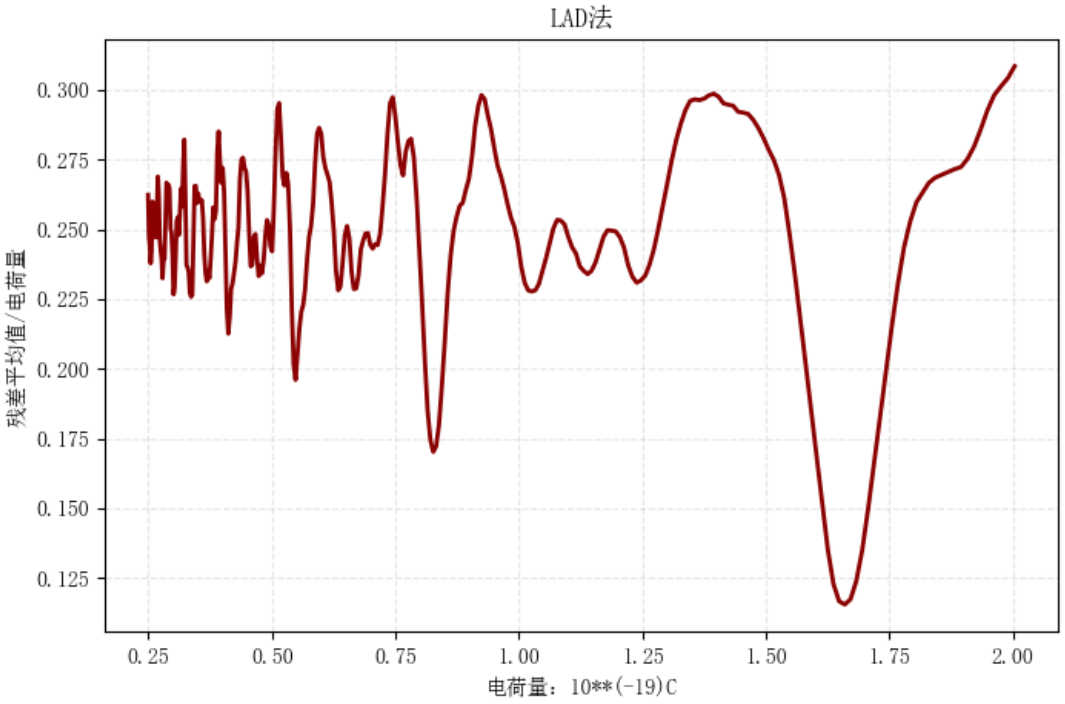


图 13 使用 Python 绘制的 LAD 法结果示意图

如下是用 Python 完成的 LAD 法运算程序结果示例图 14：

```
请输入Excel文件路径（例如：C:\data\file.xlsx）：1.4.xlsx
1.6603924298312687
按任意键退出程序.....
```

图 14 使用 Python 构建的 LAD 法运算程序图

根据结果可知，处理得到的元电荷值为：1.660×10^{-19}C，相对误差为：3.62%.

原代码见附录。

4.3.7 枚举方差法

原理：方差的大小反映了一种排列组合计算得到的元电荷值的精确性，方差越小，实验得到的电荷量除公约数得到的数据越趋向于某个。平均元电荷量（即最大公约数）由方差确定，

极小方差对应数据中最精确的元电荷值

步骤：

1、限定数据点，电荷量不能过大（一般限定小于 20 个元电荷，代码中未限定）

2、选取 1.400~1.800 区间内的值，逐步遍历

3、对于所枚举的元电荷值，遍历每个数据点除以该元电荷值，所得数值四舍五入得到整数，并反过来除得该整数应该对应的元电荷值，存储

4、计算方差，寻找极小方差所对应的电荷量

下图 15 是使用 Python 绘制的枚举方差法所得方差随电荷量变化的示意图，可以很明显看到在 1.620-1.630 处方差较小。

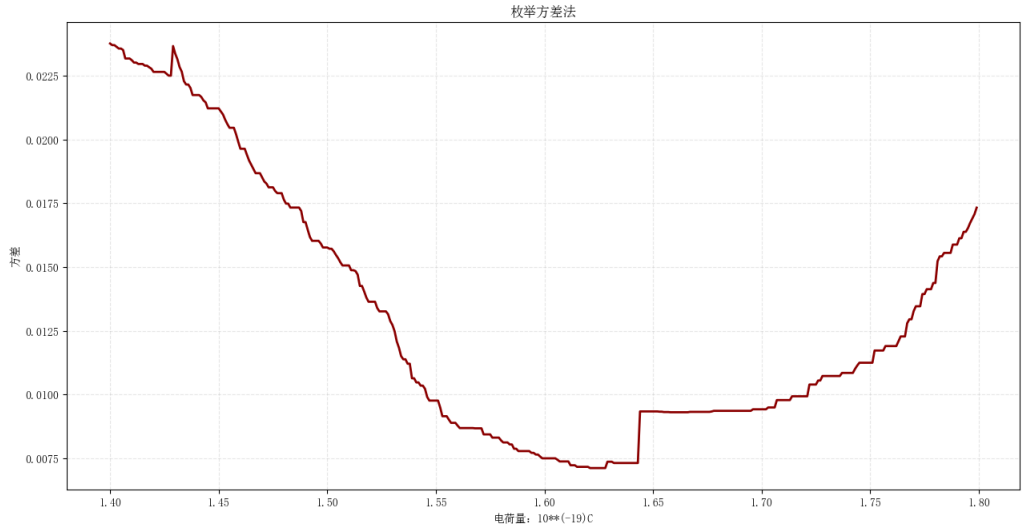


图 15 使用 Python 绘制的枚举方差法结果示意图

如下是用 Python 完成的枚举方差法运算程序结果示例图：

```
请输入Excel文件路径（例如：C:\data\file.xlsx）：12.xlsx
1.621
按任意键退出程序.....
```

图 16 使用 Python 构建的枚举方差法运算程序图

根据结果可知，处理得到的元电荷值为：1.621×10^{-19}C，相对误差为：1.19%.

Python 原代码见附录。

4.3.8 循环试商法

原理：我们假设最小的电荷数据是元电荷的 n 倍，通过循环取出元电荷值，尝试得到误差较小的第一个值被采用为元电荷

步骤：

1、对数据进行升序排序

2、循环取出i，试验所有数据点对其作商取整的方差，类似于枚举方差法

下图为使用 draw.io 绘制的循环试商法流程图。

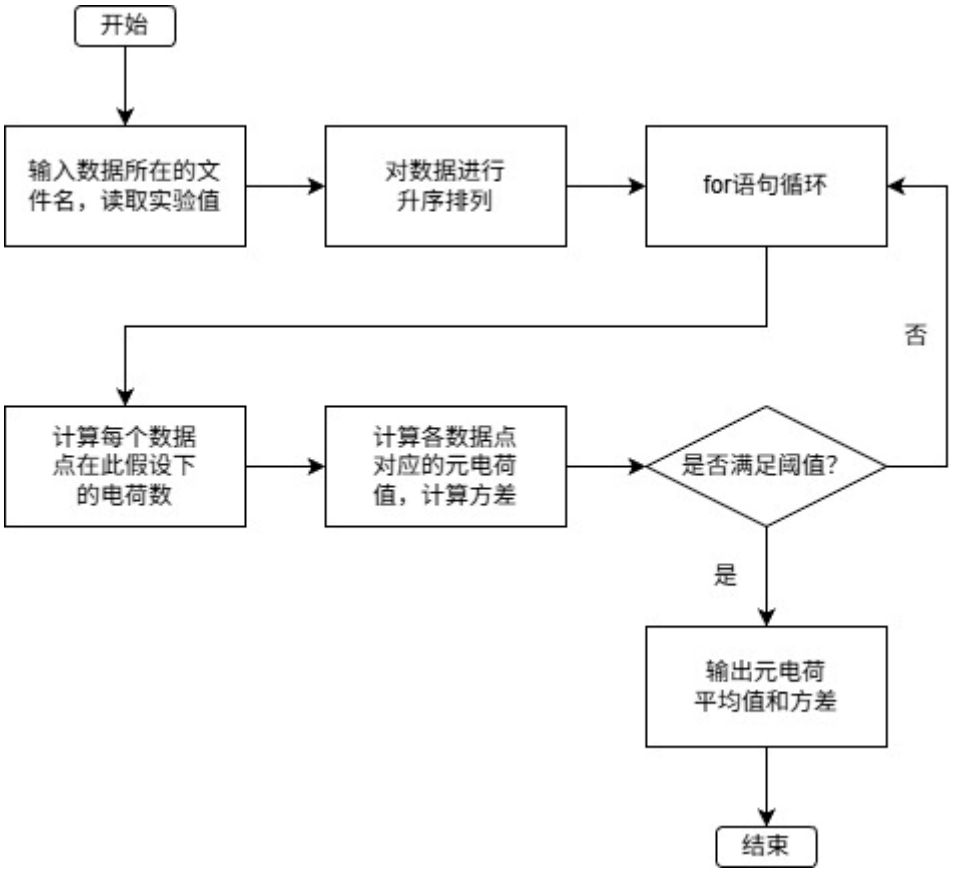


图 17 循环试商法流程图

如下是用 Python 完成的循环试商法运算程序结果示例图：

```
请输入Excel文件路径（例如：C:\data\file.xlsx）：12.xlsx
1.6514496958216367
0.007226597426032821
按任意键退出程序.....
```

图 18 使用 Python 构建的循环试商法运算程序图

根据结果可知，处理得到的元电荷值为：1.651×10^{-19}C，相对误差为：3.06%.

Python 原代码见附录。

4.3.9 层次聚类法

1、实验原理

（1）在获得若干个电荷的平衡电压与速度后，计算出每个电荷的电荷量，并将每一个数据看做一个单独的簇

```Python
class MillikanOilDrop:
    def __init__(self, voltage, time, distance=0.0015):
        self.voltage = voltage
        self.time = time
        self.distance = distance
        self.velocity = self.calculate_velocity()
        self.radius = self.calculate_radius()
        self.mass = self.calculate_mass()
        self.charge = self.calculate_charge()

    def calculate_velocity(self):
        return self.distance / self.time

    def calculate_radius(self):
        velocity = self.velocity
        term1 = 9 * eta * velocity / (2 * g * (rho - rho_air))
        return math.sqrt(term1)

    def calculate_mass(self):
        volume = (4/3) * math.pi * self.radius**3
        return volume * rho

    def calculate_charge(self):
        return (self.mass * g * d) / self.voltage
```

（2）在获得所有油滴的电荷量后，利用 Ward 法不断将最近的两个簇合并，最终形成树状图

```Python
def infer_elementary_charge_hierarchical(charges, n_clusters=None):
    charges_array = np.array(charges).reshape(-1, 1)
    data_for_clustering = charges_array
    n_samples = len(data_for_clustering)

    # 1. 获取 Linkage 矩阵
    print("\n 正在执行层次聚类以获取 linkage 矩阵...")
    linked = linkage(data_for_clustering, method='ward')
    # 2. 绘制树状图（已改为中文）
    plt.figure(figsize=(12, 6))
    plt.title('层次聚类树状图')
    dendrogram(linked, orientation='top', distance_sort='descending',
 show_leaf_counts=True)
    plt.xlabel('油滴序号')
    plt.ylabel('距离')
    plt.grid(True, alpha=0.3)
    plt.show()
```

（3）获得树状图后，可根据轮廓系数等指标判断最佳聚类数

```Python
        # --- 指标 1: 优化版手肘法（斜率变化率） ---
        wcss_diff = np.diff(wcss_list)  # 一阶导数（WCSS 下降幅度）
```

```
        wcss_diff_ratio = np.abs(wcss_diff[:-1] / wcss_diff[1:])  # 斜率变化率（前一个下降幅度 / 后一个下降幅度）
        elbow_k_idx = np.argmax(wcss_diff_ratio) + min_k  # +min_k：还原到原始K值
        # --- 指标2：轮廓系数最优K（分数最高） ---
        sil_best_k_idx = np.argmax(silhouette_scores)
        sil_best_k = k_candidates[sil_best_k_idx]
        # --- 指标3：间隙统计量最优K（分数最高） ---
        gap_best_k_idx = np.argmax(gap_scores[min_k - 1:max_k_to_test])  # 截取有效范围
        gap_best_k = min_k + gap_best_k_idx
        # --- 融合三个指标，确定最终K值 ---
        recommended_ks = [elbow_k_idx, sil_best_k, gap_best_k]
        # 处理众数不存在的情况（三个K值都不同），取中间值
        try:
            best_k = mode(recommended_ks)[0][0]
        except:
            best_k = sorted(recommended_ks)[1]  # 取中间值
```

（4）在确定最佳聚类数后，即可通过每个聚类中心用作差法求出元电荷大小

```
Python
 print(f"\n使用 K = {n_clusters} 进行最终层次聚类...")
    labels = fcluster(linked, n_clusters, criterion='maxclust')
    cluster_labels = labels
    # 5. 计算聚类中心
    cluster_centers = []
    for i in range(1, n_clusters + 1):
        center = data_for_clustering[cluster_labels == i].mean(axis=0)
        cluster_centers.append(center[0])
    cluster_centers = sorted(cluster_centers)
    print(f"\n计算得到的簇中心（已排序）: {[f'{c:.6e}' for c in cluster_centers]}")
    # 6. 推断元电荷（核心优化保留）
    elementary_charge = 0.0
    if n_clusters > 1:
        differences = np.diff(cluster_centers)
        print(f"\n相邻簇中心的原始差值: {[f'{d:.6e}' for d in differences]}")
        if len(differences) > 1:
            median_diff = np.median(differences)
            mad = np.median(np.abs(differences - median_diff))
            if mad > 0:
                modified_z_scores = 0.6745 * (differences - median_diff) / mad
                filtered_diffs = differences[np.abs(modified_z_scores) <= 3.5]

                if len(filtered_diffs) > 0:
                    print(f"检测到并去除了 {len(differences) - len(filtered_diffs)} 个离群差值。")
                    print(f"过滤后的差值: {[f'{d:.6e}' for d in filtered_diffs]}")
```

```
                    elementary_charge = np.median(filtered_diffs)
                else:
                    print("警告：过滤后没有剩余的差值，将使用原始中位数。
")
                    elementary_charge = median_diff
            else:
                print("警告：中位数绝对偏差(MAD)为零，无法过滤离群值。")
                elementary_charge = median_diff

        elif len(differences) == 1:
            print("只有一个差值，直接使用该值作为元电荷估计。")
            elementary_charge = differences[0]

        print(f"使用稳健估计方法得到的元电荷
为：{elementary_charge:.6e} C")
    else:
        print("警告：仅检测到一个聚类。这可能导致元电荷计算不准确。")
        print("尝试使用所有油滴电荷量的最小值作为元电荷估计...")
        if charges:
            elementary_charge = min(charges)
            print(f"使用最小值估计的元电荷
为：{elementary_charge:.6e} C")
        else:
            elementary_charge = 0
```

2、实验结果

根据上述实验操作，运用 Python 构建程序并可得到如下结果图：

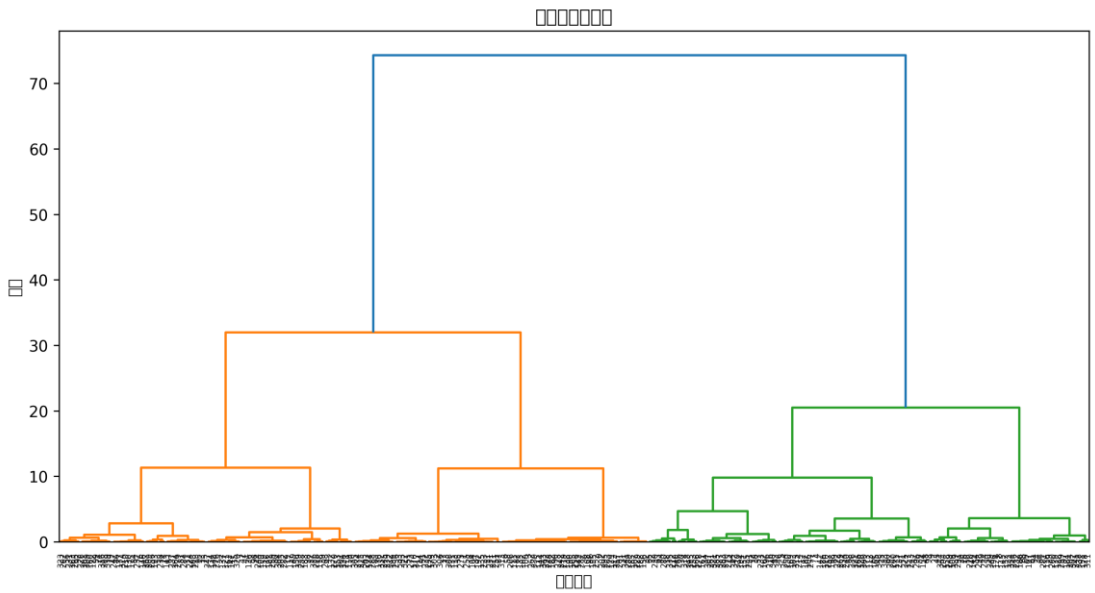


图 19 层次聚类树状图

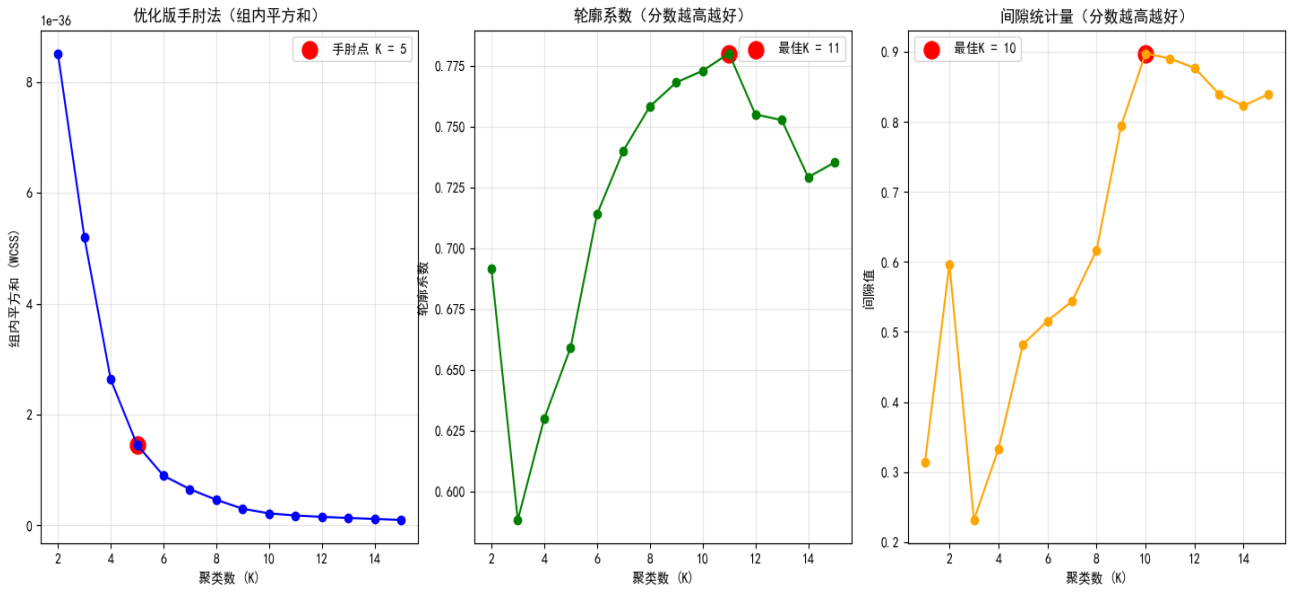


图 20 最佳聚类数的判断图

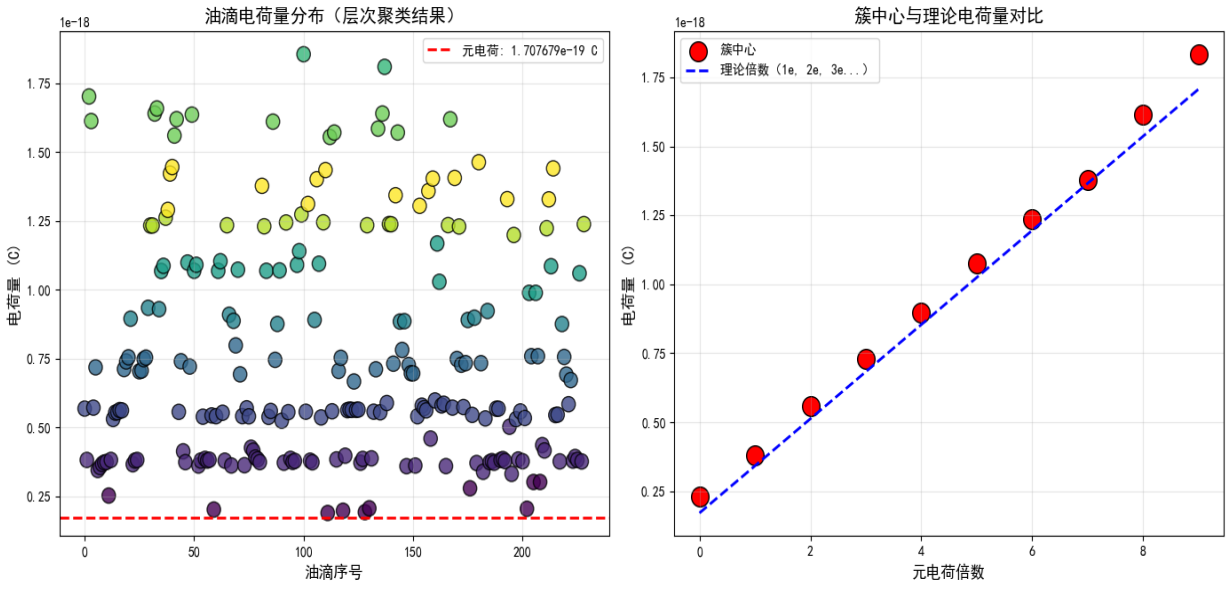


图 21 确定最佳聚类后的簇图

最后取 K=10，推断的元电荷值为：1.707679×10^{-19}C，公认理论值：$1.602176634 \times 10^{-19}$C，相对误差：6.58%.

5 误差分析

5.1 视觉辅助实验

实验取得了一定成果，整体效果良好。在最终测试中，实验数据与传统方法的结果相差不大，但仍存在一定误差。针对误差的可能来源，我们进行了如下分析：

5.1.1 摄像头分辨率限制

在视频处理过程中，由于设备自身的局限性，摄像头分辨率不足，且录像过程中存在大量噪点。这导致油滴在运动过程中可能出现暂时消失的情况，从而造成油滴信息的丢失。

5.1.2 光照条件的影响

油滴识别基于初始的模板匹配方法。虽然在不同光照条件下油滴的整体形态在人眼观察下变化不大，但计算机识别容易受光照变化影响，导致运动过程中油滴信息丢失或时间顺序错乱。这可能解释了视觉法的不确定度在某些情况下反而高于传统方法的现象。

5.1.3 ROI 上下边界判定引入的提前/滞后误差

为防止因丢帧导致油滴经过目标线时未被识别，识别区域由单一目标线扩展为目标线两侧的带状区域（±12 像素）。当油滴中心进入该区域时，即视为到达目标线，并记录当前帧号。然而，该方法存在以下问题：

1、油滴可能在带状区域的任意位置首次被检测到；

2、模板匹配可能并非每帧都成功；

因此，首次进入带状区域的时间并不等于油滴中心真实通过$Y_{textStart}$的时间。这引入了 ±Δt 的随机误差，进而影响测得速度和最终电荷 q 的计算。

5.1.4 跨帧油滴身份匹配的不完善

由于处理限制和油滴数量较多，采用基于空间分桶（binning）的简化关联策略：在连续帧中，如果同一油滴的水平坐标变化不超过 ±12.5 像素（半个 bin 宽度），则认为其属于同一x_{bin}，从而继承先前轨迹信息。这种方法提高了运行效率并保证实时处理的准确性，但仍存在以下问题：

1、水平漂移：若油滴因气流或布朗运动水平移动超过 12.5 像素，会被分配到相邻 bin，导致系统将其识别为新油滴，旧轨迹丢失，无法完成完整计时；

2、相邻油滴干扰：当两个油滴的水平距离小于 25 像素时，它们会被分配到同一 bin，后检测到的油滴轨迹会覆盖前者，导致轨迹混淆，速度和电荷计算错误；

3、短暂丢失：若某帧未检测到油滴，则 trajectories 中对应x_{bin}被清空；下一帧重新检测到的油滴被视作新油滴，轨迹被拆分，影响测量连续性。

5.1.5 轨迹状态机设计的局限性

每条轨迹的状态机设计如下：

1、油滴进入起点带 → 记录$t_{textStart}$；

2、已记录 $t_{textStart}$ 且油滴进入终点带 → 计算v_g 和 n，并存入轨迹；

3、若 $v_{g_{stored}}$ 已存在 → 不再更新。

因此，每个 x_{bin}的轨迹仅可完成一次测量。即使油滴继续存在或在可逆电场实验中再次运动，也无法复用同一油滴进行多次测量。此外，油滴在终点区域附近的抖动可能多次触发终点条件，但状态机仅记录第一次，从而避免重复计时。

5.2 数据处理实验

数据处理实验中得到的结果整体与元电荷值接近，由于各类方法所采用的判据与计算方法各不相同，误差的来源也会呈现多样性。如图 13 是对本实验不同数据处理方法结果与相对误差比较，其中线性拟合与对数修正、余数法、多重差值法和残差法选取 11 组文献中的数据进行处理。其余方式是选择大量数据的分处理结果。

单位：$\times 10^{-19}C$

方法/标准值	e_i		$E_r = \frac{\lvert e_i - e_{standard} \rvert}{e_{standard}} \times 100\%$
元电荷标准值	1.602		
线性拟合法	1.567		2.22%
对数修正法	1.57		2.03%
余数法	1.545		3.56%
多重差值法	1.567		2.18%
残差法	1.579		1.44%
最大公约数法	1.675		4.56%
LAD 法	1.66		3.62%
枚举方差法	1.621		1.19%
循环试商法	1.651		3.06%
层次聚类法	1.708		6.62%

图 22　不同方法下数据处理结果与相对误差比较

可见上述方法较为合理，误差均在合理范围内，对此我们对误差做了以下分析：

5.2.1 层次聚类算法

在层次聚类算法中，使用不同方法判断最佳聚类数得到的值是不同的，可能会与实际上按照电荷量进行分组的聚类数有差距。

5.2.2 已知元电荷值情况下的数据处理

在已知元电荷的情况下，线性拟合法元电荷的计算主要依赖测量数据的准确性，由于四舍五入得到电荷的带电量，其反过来计算元电荷的值时无法消除数据的偏差，当一组测量数据整体偏大或

偏小时，就会导致测量的元电荷值与实际值偏大或偏小。而对数法修正后能够一定程度上消除数据的整体偏差，但是可能存在的误差即为在公式中的估算，即将极小值的近似、$\ln e$和$\ln n$的修正带来的误差。

5.2.3 未知元电荷值情况下的数据处理

在未知元电荷的情况下，首先是对大量数据处理的电荷分布量子化的发现和最小电荷值范围的确定，在一定程度上也需要电荷值测量的相对聚集。整体而言，误差的来源绝大程度上仍取决于电荷数据的误差，一部分却决于具体方法处理的误差。

余数法和多重差值法其本质都是多次做差，当循环步骤次数过多时会导致不同值之间的差值误差增大并累积，最终会导致较大的误差，因此控制输入的数据量和处理次数对结果的精度尤为重要。

而残差法的误差主要还依赖对步长的选择，步长选择越小其精度反而下降，这是由于步长越小，系列连续数据更多，可能出现计算所得的电荷量的四舍五入进位、整除的情况，这些会导致极小值的选择出现“假阳性”，对最终的极小值聚集判断带来更大困难。

最大公约数法受限于时间复杂度较高，使用较多数据时容易卡顿，使用较少数据时又会出现随机误差；同时，方法中所确定的N来源于数据最大值与最小值之比取整，容易受到这两个数据的影响，使用时若最小值和最大值出现较大的不合理性容易出现错误。

LAD 法的误差主要来源于e_0的选取，由于扫描初值与数据中最小值有关，若最小值出现异常（如小于 0.8 个元电荷或大于 7 个元电荷），将会把公认元电荷值排除在外。

而枚举方差法和循环试商法本质上都是通过方差来判断结果，若枚举方差法中未限定初值为 1.4 左右，容易出现得到$\frac{1}{2}e$等等数字，而循环试商法是假定数据最小值对e为整数，因此对初值都有要求；同时，枚举方差法得到的结果是方差最小的数值，而循环试商法是第一个方差小于某个误差范围的值，判据的不同导致了其结果存在差异。

6 实验总结

6.1 实验总述

本项目围绕密立根油滴实验的核心目标——元电荷的精确测定，开展了数据后处理与实验过程智能化的系统性创新。

我们采用余数法、聚类分析等多种不同的数据处理模型，验证了电荷的量子化特性，同时对比了不同方法在抗噪性、稳定性和精度上的表现，为科研与教学提供了多种可行的数据分析路径。

同时，我们开发的基于 OpenCV 和 Python 的实时视觉辅助系统，成功实现了油滴运动追踪、电压自动识别、电荷量实时计算和结果可视化的一体化操作，降低了实验操作门槛，帮助学生更直观地理解实验原理和数据生成过程。

最后，我们对不同实验数据进行了分析，探讨了数据处理模型的适用性，并对实验过程中可能产生的误差进行了详细的讨论，进一步验证了该系统的可靠性和可行性。

6.2 未来展望

本项目在许多方面仍有不成熟之处，但是有了此次实践结果，我们知道该项目在整体的思路与想法上是可行的。若是要实现进一步的呀优化，可以根据上一小节提到的众多误差与不足指出，在以下方面进行改进。

1、程序算法：当前的视觉识别与数据处理程序在功能上已能完成基本测量，但仍有不少可优化空间。例如，油滴识别算法可以进一步改进，使其在光线变化或图像模糊的情况下依然稳定工作；轨迹匹配逻辑也可以调整，减少因油滴轻微晃动或短暂消失带来的误判。同时，电荷计算和结果展示部分可加入更多数据筛选和自动校正机制，提升最终元电荷结果的准确性。

而在数据处理过程中，我们可以逐步扩大数据范畴，进一步微调数据处理模型，实现元电荷量的更精准测量。也可以通过优化代码结构和引入更高效的处理流程，也有望在普通电脑上实现更快速流畅的数据处理效果。

2、仪器配件：可以选择一个分辨率更高、帧率更高，信号更稳定的摄像头来进行实验，可以很好的提升结果的准确性。可以优化电压调节功能，把电压调节也与计算机连接起来，实现整个过程的相对自动化。也可以考虑借用高算力设备，实现更流畅的数据处理效果，也可以处理更多的数据。

3、实验 ui 的整体设计：实验目前基本上是可运行但缺乏图形界面的，为了提升使用体验，未来还可以在现有程序基础上开发一个简单的图形操作界面（GUI），将视频显示、参数设置、结果输出等功能整合在一起，让整个系统更直观、易用，更适合教学实验场景。

7 课题总结

7.1 程序构建图

下图 23 和 24 是本次课题最终完成的视觉辅助程序和数据处理程序构建图。

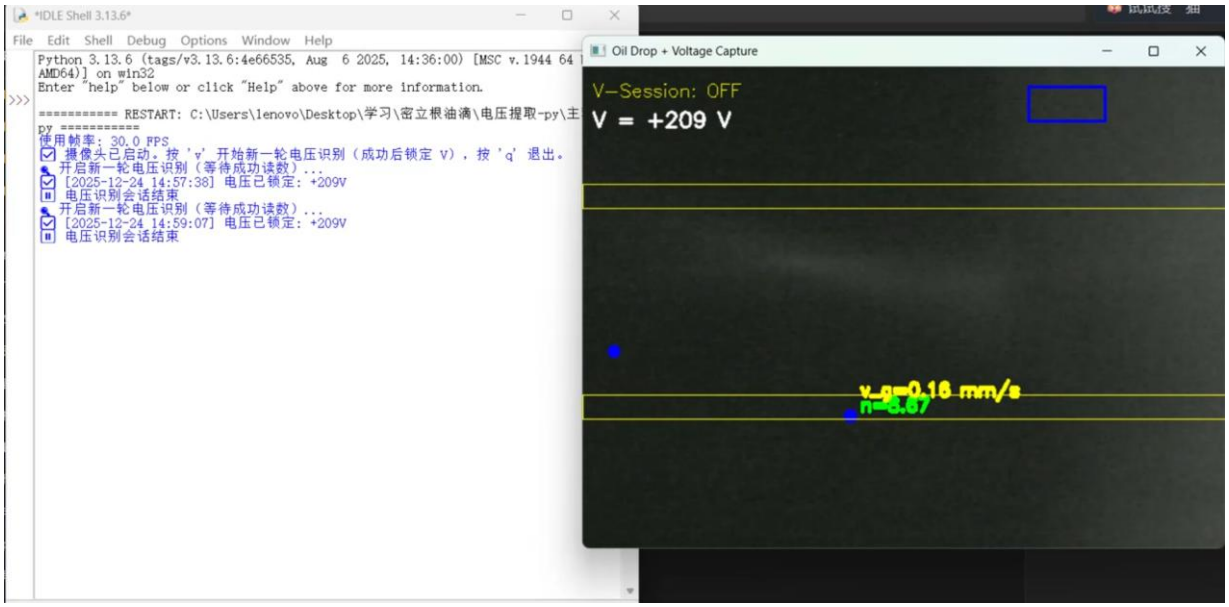


图 23 Python 构建的联动密立根油滴仪的视觉辅助程序图

图 24 VB6.0 构建的整合的数据处理实验程序图

7.2 代码图

以下是视觉辅助实验和层次聚类法的 Python 总代码

7.2.1 视觉辅助实验

1、模版生成.py

```
Python
import cv2
import os
import numpy as np
# ------------------------------
# 配置1
# ------------------------------
ROI_REL = (0.6953, 0.0420, 0.8125, 0.1125)  # 与你原程序一致
TEMPLATE_DIR = "templates"
os.makedirs(TEMPLATE_DIR, exist_ok=True)
# 模板目标尺寸：宽 x 高
TEMPLATE_SIZE = (50, 80)  # (width, height)
class TemplateGenerator:
    def __init__(self):
        self.cap = cv2.VideoCapture(0)
        if not self.cap.isOpened():
            raise Exception("无法打开摄像头")
        print("✅ 摄像头已打开。")
        print("操作说明：")
        print("   - 按 's' 键暂停并标注当前画面中的三位数字")
        print("   - 输入如 '234'，程序将自动切分并保存
为 2.png, 3.png, 4.png（50x80 像素）")
        print("   - 按 'q' 退出")
    def enhance_roi(self, roi_gray):
        """预处理 ROI 图像"""
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
        enhanced = clahe.apply(roi_gray)
        _, binary = cv2.threshold(enhanced, 0, 255, cv2.THRESH_BINARY
 + cv2.THRESH_OTSU)
        return binary
    def save_digit_templates(self, roi_binary, digits_str):
        """将 ROI 切分为三个数字区域，保存为 50x80 的模板"""
        h, w = roi_binary.shape
        digit_width = w // 3
        for i, digit_char in enumerate(digits_str):
            x1 = i * digit_width
            x2 = (i + 1) * digit_width
            digit_roi = roi_binary[:, x1:x2]
            # 调整为统一尺寸 (50x80)
            digit_resized = cv2.resize(digit_roi, TEMPLATE_SIZE, inte
rpolation=cv2.INTER_AREA)
            # 保存为 PNG
            filename = os.path.join(TEMPLATE_DIR, f"{digit_char}.png"
)
            cv2.imwrite(filename, digit_resized)
            print(f"已保存: {filename} (尺
寸: {TEMPLATE_SIZE[0]}x{TEMPLATE_SIZE[1]})")
    def run(self):
        while True:
            ret, frame = self.cap.read()
            if not ret:
                print("❌ 摄像头读取失败")
                break
            h, w = frame.shape[:2]
            x1 = int(w * ROI_REL[0])
```

```
            y1 = int(h * ROI_REL[1])
            x2 = int(w * ROI_REL[2])
            y2 = int(h * ROI_REL[3])
            roi = frame[y1:y2, x1:x2]
            if roi.size == 0:
                continue
            gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
            processed = self.enhance_roi(gray)
            # 绘制 ROI 框
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, "Press 's' to save templates", (10, 30
),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2
)
            cv2.imshow("Live View", frame)
            cv2.imshow("Processed ROI", processed)
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                break
            elif key == ord('s'):
                # 暂停画面（可选：显示提示）
                print("\n 请在终端输入看到的三位数字（如 234）:")
                try:
                    user_input = input().strip()
                except KeyboardInterrupt:
                    break
                if len(user_input) != 3 or not user_input.isdigit():
                    print("❌ 输入无效，请输入三位数字！")
                    continue
                # 使用当前帧的 ROI 生成模板
                gray_current = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
                processed_current = self.enhance_roi(gray_current)
                self.save_digit_templates(processed_current, user_inp
ut)
                # 可选：在画面上显示已保存
                cv2.putText(frame, f"Saved: {user_input}", (10, 60),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 2
55), 2)
                cv2.imshow("Live View", frame)
                cv2.waitKey(500)  # 短暂显示反馈
        self.cap.release()
        cv2.destroyAllWindows()
        print("模板生成完成！")
if __name__ == "__main__":
    generator = TemplateGenerator()
    generator.run()
```

2、主程序.py

```
Python
import cv2
import numpy as np
import os
```

```
from datetime import datetime
import json
# ==================== 油滴实验配置 ====================
TEMPLATE_PATH = "oil_drop_template.png"
VIDEO_PATH = 0
PIXELS_PER_MM = (339-129)/0.75
L_MM = 0.75
Y_START = 129
Y_END = 339
ZONE_HEIGHT = 12
X_TOLERANCE = 25
MIN_MATCH_THRESHOLD = 0.8
FPS = 30.0
rho = 886.0
g = 9.8
eta = 1.8e-5
b = 8.2e-3
p = 101325.0
d = 5e-3
e0 = 1.602e-19
# ==================== 电压识别配置 ====================
ROI_REL = (0.6953, 0.0420, 0.8125, 0.1125)
SAVE_PATH = "voltage_screenshots"
TEMPLATE_DIR = "templates"
os.makedirs(SAVE_PATH, exist_ok=True)
os.makedirs(TEMPLATE_DIR, exist_ok=True)
# ==================== 电荷计算函数 ====================
def calculate_charge(v_g_mm_per_s, V):
    if V is None or V <= 0:
        return None, None
    v_g = v_g_mm_per_s / 1000.0
    try:
        a = np.sqrt(9 * eta * v_g / (2 * rho * g))
        eta_p = eta / (1 + b / (p * a))
        m = (4/3) * np.pi * rho * (9 * eta_p * v_g / (2 * rho * g))**
(3/2)
        q = m * g * d / V
        n = q / e0
        return q, n
    except Exception:
        return None, None
# ==================== 电压识别器 ====================
class DigitTemplateMatcher:
    def __init__(self):
        self.templates = {}
        self.load_templates()
    def enhance_roi(self, roi_gray):
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        enhanced = clahe.apply(roi_gray)
        _, binary = cv2.threshold(enhanced, 0, 255, cv2.THRESH_BINARY
 + cv2.THRESH_OTSU)
        return binary
    def load_templates(self):
        for digit in range(10):
            path = os.path.join(TEMPLATE_DIR, f"{digit}.png")
            if os.path.exists(path):
```

```
                img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
                if img is not None:
                    self.templates[digit] = self.enhance_roi(img)
    def match_digit(self, roi_digit):
        if not self.templates:
            return None
        best_match = -1
        best_score = -1
        for digit, tmpl in self.templates.items():
            resized = cv2.resize(roi_digit, (tmpl.shape[1], tmpl.shap
e[0]))
            res = cv2.matchTemplate(resized, tmpl, cv2.TM_CCOEFF_NORM
ED)
            _, max_val, _, _ = cv2.minMaxLoc(res)
            if max_val > best_score:
                best_score = max_val
                best_match = digit
        return best_match if best_score > 0.5 else None
    def extract_voltage(self, roi_gray):
        h, w = roi_gray.shape
        if w < 30 or h < 10:  # 防止过小
            return None
        digit_width = w // 3
        digits = []
        for i in range(3):
            x1 = i * digit_width
            x2 = (i + 1) * digit_width
            digit_roi = roi_gray[:, x1:x2]
            digit = self.match_digit(digit_roi)
            if digit is None:
                return None
            digits.append(digit)
        if digits[0] not in {0,1,2,3}:
            return None
        return digits[0]*100 + digits[1]*10 + digits[2]
# ==================== 主程序 ====================
def main():
    template = cv2.imread(TEMPLATE_PATH, cv2.IMREAD_GRAYSCALE)
    if template is None:
        raise FileNotFoundError(f"请准备油滴模板: {TEMPLATE_PATH}")
    cap = cv2.VideoCapture(VIDEO_PATH)
    if not cap.isOpened():
        raise IOError("无法打开摄像头")
    actual_fps = cap.get(cv2.CAP_PROP_FPS)
    if actual_fps > 5:
        FPS = actual_fps
    print(f"使用帧率: {FPS:.1f} FPS")
    # ===== 电压状态 =====
    voltage_session_active = False  # 当前是否在进行一次电压采集会话
    locked_V = None                 # 当前使用的电压值（用于油滴计算）
    last_logged_voltage_str = None  # 日志去重
    # ===== 油滴状态 =====
    trajectories = {}
    frame_id = 0
```

```
    print("✅ 摄像头已启动。按 'v' 开始新一轮电压识别（成功后锁定 V），
按 'q' 退出。")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_id += 1
        h_full, w_full = frame.shape[:2]
        x1_roi = int(w_full * ROI_REL[0])
        y1_roi = int(h_full * ROI_REL[1])
        x2_roi = int(w_full * ROI_REL[2])
        y2_roi = int(h_full * ROI_REL[3])
        roi = frame[y1_roi:y2_roi, x1_roi:x2_roi]
        # ===== 电压识别逻辑 =====
        if voltage_session_active:
            if roi.size > 0:
                roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
                voltage_int = DigitTemplateMatcher().extract_voltage(
roi_gray)
                if voltage_int is not None and locked_V != voltage_in
t:
                    # 成功识别新电压
                    locked_V = float(voltage_int)
                    voltage_str = f"+{locked_V:.0f}V"
                    # 日志
                    if voltage_str != last_logged_voltage_str:
                        timestamp = datetime.now().strftime("%Y-%m-%d
 %H:%M:%S")
                        print(f"✅ [{timestamp}] 电压已锁
定: {voltage_str}")
                        with open(os.path.join(SAVE_PATH, "voltage_lo
g.jsonl"), "a", encoding='utf-8') as f:
                            f.write(json.dumps({"time": timestamp, "v
oltage": voltage_str}, ensure_ascii=False) + "\n")
                        last_logged_voltage_str = voltage_str
                    # 自动结束本次会话（可选：也可让用户手动关闭）
                    # 这里我们保持会话开启，但只记录第一次
（因 locked_V 已设，后续 != 会跳过）
        else:
            # 会话未激活，不识别
            pass
        # ===== 油滴追踪 =====
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        res = cv2.matchTemplate(gray, template, cv2.TM_CCOEFF_NORMED)
        loc = np.where(res >= MIN_MATCH_THRESHOLD)
        detections = [(pt[0] + template.shape[1]//2, pt[1] + template
.shape[0]//2) for pt in zip(*loc[::-1])]
        new_trajs = {}
        for (x, y) in detections:
            x_bin = round(x / X_TOLERANCE)
            traj = trajectories.get(x_bin, {'t_start': None, 'v_g_sto
red': None, 'n_stored': None})
            if traj['t_start'] is None and (Y_START - ZONE_HEIGHT <=
y <= Y_START + ZONE_HEIGHT):
                traj['t_start'] = frame_id
```

```
            if (traj['t_start'] is not None and
                traj['v_g_stored'] is None and
                Y_END - ZONE_HEIGHT <= y <= Y_END + ZONE_HEIGHT):
                dt = frame_id - traj['t_start']
                if dt > 0:
                    t_sec = dt / FPS
                    v_g = L_MM / t_sec
                    traj['v_g_stored'] = v_g
                    q, n = calculate_charge(v_g, locked_V)
                    traj['n_stored'] = n if q is not None else None
            if traj['v_g_stored'] is not None:
                cv2.putText(frame, f"v_g={traj['v_g_stored']:.2f} mm/
s", (int(x)+10, int(y)-20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,255
), 2)
                if traj['n_stored'] is not None:
                    cv2.putText(frame, f"n={traj['n_stored']:.2f}", (
int(x)+10, int(y)-5),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255
,0), 2)
            new_trajs[x_bin] = traj
            cv2.circle(frame, (int(x), int(y)), 6, (255,0,0), -1)
        trajectories = new_trajs
        # ===== 绘制 =====
        cv2.rectangle(frame, (x1_roi, y1_roi), (x2_roi, y2_roi), (255
,0,0), 2)
        cv2.rectangle(frame, (0, Y_START - ZONE_HEIGHT), (w_full, Y_S
TART + ZONE_HEIGHT), (0,255,255), 1)
        cv2.rectangle(frame, (0, Y_END - ZONE_HEIGHT), (w_full, Y_END
 + ZONE_HEIGHT), (0,255,255), 1)
        # 状态显示
        status = "V-Session: ON" if voltage_session_active else "V-
Session: OFF"
        v_txt = f"V = +{locked_V:.0f} V" if locked_V is not None else
 "V = ???"
        cv2.putText(frame, status, (10, 30), cv2.FONT_HERSHEY_SIMPLEX
, 0.6, (0,255,255), 1)
        cv2.putText(frame, v_txt, (10, 60), cv2.FONT_HERSHEY_SIMPLEX,
 0.7, (255,255,255), 2)
        cv2.imshow("Oil Drop + Voltage Capture", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
        elif key == ord('v'):
            # 切换会话，并重置 V
            voltage_session_active = not voltage_session_active
            if voltage_session_active:
                locked_V = None  # 关键：清空旧 V，允许重新识别
                last_logged_voltage_str = None
                print("开启新一轮电压识别（等待成功读数）...")
            else:
                print("电压识别会话结束")
    cap.release()
    cv2.destroyAllWindows()
    print("退出")
if __name__ == "__main__":
```

```
    main()
```

3、日志读取.py

```
Python
import json
import os
import csv
from datetime import datetime
LOG_FILE = "voltage_screenshots/voltage_log.jsonl"
CSV_OUTPUT = "voltage_screenshots/voltage_log.csv"
def read_jsonl(file_path):
    """安全读取 JSONL 文件，跳过无效行"""
    records = []
    if not os.path.exists(file_path):
        print(f" 文件不存在: {file_path}")
        return records
    with open(file_path, "r", encoding="utf-8") as f:
        for line_num, line in enumerate(f, 1):
            line = line.strip()
            if not line:
                continue  # 跳过空行
            try:
                record = json.loads(line)
                records.append(record)
            except json.JSONDecodeError as e:
                print(f" 第 {line_num} 行 JSON 解析失败，已跳过: {e}")
    return records
def save_to_csv(records, output_path):
    """将记录保存为 CSV 文件"""
    if not records:
        print(" 无可导出数据。")
        return
    with open(output_path, "w", encoding="utf-8", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=["time", "voltage"])
        writer.writeheader()
        writer.writerows(records)
    print(f"✅ 已保存 CSV: {os.path.abspath(output_path)}")
def main():
    print(" 正在读取电压日志...\n")
    records = read_jsonl(LOG_FILE)
    if not records:
        print(" 未找到有效记录。")
        return
    # 打印到控制台
    print(f" 共 {len(records)} 条记录：")
    print("-" * 40)
    for i, rec in enumerate(records, 1):
        print(f"{i:2d}. {rec['time']} → {rec['voltage']}")
    # 询问是否导出 CSV
    print("\n" + "="*50)
    choice = input("是否导出为 CSV 文件？(y/n): ").strip().lower()
    if choice in ("y", "yes"):
```

```
        save_to_csv(records, CSV_OUTPUT)
if __name__ == "__main__":
    main()
```

7.2.2 层次聚类算法

聚类算法 Python 总代码如下

```
Python
import numpy as np
import matplotlib.pyplot as plt
# -------------------------- 新增中文显示配置（关键 3 行）----------------
-----------
plt.rcParams['font.sans-
serif'] = ['SimHei', 'Microsoft YaHei', 'Arial Unicode MS']  # 优先使
用黑体/微软雅黑
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示为方块的问题
# ------------------------------------------------------------------
-------------
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from scipy.stats import mode
import math
import os
# --- 物理常数 ---
g = 9.8
rho = 981
rho_air = 1.29
eta = 1.83e-5
d = 0.005
# --- 油滴类 ---
class MillikanOilDrop:
    def __init__(self, voltage, time, distance=0.0015):
        self.voltage = voltage
        self.time = time
        self.distance = distance
        self.velocity = self.calculate_velocity()
        self.radius = self.calculate_radius()
        self.mass = self.calculate_mass()
        self.charge = self.calculate_charge()

    def calculate_velocity(self):
        return self.distance / self.time

    def calculate_radius(self):
        velocity = self.velocity
        term1 = 9 * eta * velocity / (2 * g * (rho - rho_air))
        return math.sqrt(term1)

    def calculate_mass(self):
        volume = (4/3) * math.pi * self.radius**3
```

```
        return volume * rho

    def calculate_charge(self):
        return (self.mass * g * d) / self.voltage
# --- 数据加载和筛选函数 ---
def load_and_filter_data(filename):
    oil_drops = []
    line_num = 0
    min_voltage = 100.0
    max_voltage = 400.0
    min_time = 10.0
    max_time = 40.0

    try:
        with open(filename, 'r', encoding='utf-8') as f:
            for line in f:
                line_num += 1
                line = line.strip()
                if not line or line.startswith('#'): continue
                parts = line.split()
                if len(parts) != 2:
                    print(f"警告: 第{line_num}行格式错误，跳过此
行: {line}")
                    continue
                try:
                    voltage = float(parts[0])
                    time = float(parts[1])

                    if not (min_voltage <= voltage <= max_voltage):
                        print(f"警告: 第{line_num}行电
压 ({voltage:.1f}V) 超出范
围 [{min_voltage:.1f}V, {max_voltage:.1f}V]，跳过。")
                        continue
                    if not (min_time <= time <= max_time):
                        print(f"警告: 第{line_num}行时
间 ({time:.1f}s) 超出范围 [{min_time:.1f}s, {max_time:.1f}s]，跳过。")
                        continue
                    if voltage <= 0 or time <= 0:
                        print(f"警告: 第{line_num}行数据无效（电压和时间
必须为正数），跳过。")
                        continue

                    oil_drop = MillikanOilDrop(voltage, time)
                    oil_drops.append(oil_drop)
                    print(f"加载油滴 {len(oil_drops)}: 电压
={voltage:.1f}V, 时间={time:.1f}s, 电荷量={oil_drop.charge:.6e}C")

                except ValueError:
                    print(f"警告: 第{line_num}行数据格式错误，跳过此
行: {line}")

        if not oil_drops:
            print("错误: 未加载到有效数据")
            return None
```

```
        print(f"\n 数据加载完成。共加载 {len(oil_drops)} 个有效油滴数据。
")
        return oil_drops

    except FileNotFoundError:
        print(f"错误: 找不到文件 '{filename}'")
        return None
    except Exception as e:
        print(f"错误: 读取文件时发生错误 - {str(e)}")
        return None
# --- 新增：计算轮廓系数（衡量聚类质量） ---
def calculate_silhouette_score(data, labels):
    """
    轮廓系数：衡量聚类的紧致性和分离度，范围[-1,1]
    越接近 1：聚类效果越好（同一簇内数据集中，不同簇分离好）
    越接近-1：聚类效果越差（数据分错簇）
    """
    n_samples = len(data)
    if len(np.unique(labels)) <= 1 or len(np.unique(labels)) == n_sam
ples:
        return -1.0  # 只有1 个簇或每个簇1 个数据，无意义

    silhouette_vals = []
    for i in range(n_samples):
        # 第i 个数据的簇标签
        label_i = labels[i]
        # 同一簇内其他数据
        same_cluster = data[labels == label_i]
        # 不同簇的数据
        other_clusters = data[labels != label_i]

        # a_i：同一簇内，i 到其他数据的平均距离（簇内紧致性）
        if len(same_cluster) > 1:
            a_i = np.mean(np.linalg.norm(same_cluster - data[i], axis
=1))
        else:
            a_i = 0.0

        # b_i：不同簇中，i 到最近簇的平均距离（簇间分离度）
        b_i = float('inf')
        for cluster_label in np.unique(labels):
            if cluster_label == label_i:
                continue
            cluster_data = data[labels == cluster_label]
            dist = np.mean(np.linalg.norm(cluster_data - data[i], axi
s=1))
            if dist < b_i:
                b_i = dist

        # 轮廓系数：(b_i - a_i) / max(a_i, b_i)
        if max(a_i, b_i) == 0:
            silhouette_vals.append(0.0)
        else:
            silhouette_vals.append((b_i - a_i) / max(a_i, b_i))
```

```
    return np.mean(silhouette_vals)
# --- 新增：计算间隙统计量（对比随机数据，避免假聚类） ---
def calculate_gap_statistic(data, linked, max_k):
    """
    间隙统计量：比较真实数据的 WCSS 和“随机数据的 WCSS 期望”
    间隙值越大，说明该 K 值的聚类越“真实”（不是随机分布造成的）
    """
    n_samples, n_features = data.shape
    # 生成参考分布（随机数据，范围和真实数据一致）
    reference_data = np.random.uniform(
        low=data.min(axis=0),
        high=data.max(axis=0),
        size=(10, n_samples, n_features)  # 生成 10 组随机数据，减少随机
性影响
    )

    # 计算真实数据的WCSS
    true_wcss = []
    for k in range(1, max_k + 1):
        labels = fcluster(linked, k, criterion='maxclust')
        wcss = 0.0
        for i in range(1, k + 1):
            cluster_points = data[labels == i]
            if len(cluster_points) > 0:
                center = cluster_points.mean(axis=0)
                wcss += np.sum((cluster_points - center) ** 2)
        true_wcss.append(wcss)

    # 计算参考数据的WCSS 期望
    ref_wcss_mean = []
    for k in range(1, max_k + 1):
        ref_wcss = []
        for ref_data in reference_data:
            ref_linked = linkage(ref_data, method='ward')
            ref_labels = fcluster(ref_linked, k, criterion='maxclust'
)
            wcss = 0.0
            for i in range(1, k + 1):
                cluster_points = ref_data[ref_labels == i]
                if len(cluster_points) > 0:
                    center = cluster_points.mean(axis=0)
                    wcss += np.sum((cluster_points - center) ** 2)
            ref_wcss.append(wcss)
        ref_wcss_mean.append(np.mean(ref_wcss))

    # 计算间隙值：Log(参考WCSS 期望) - Log(真实WCSS)
    gap = np.log(ref_wcss_mean) - np.log(true_wcss)
    return gap
# --- 核心优化：改进K 值选择逻辑 ---
def infer_elementary_charge_hierarchical(charges, n_clusters=None):
    charges_array = np.array(charges).reshape(-1, 1)
    data_for_clustering = charges_array
    n_samples = len(data_for_clustering)

    # 1. 获取 Linkage 矩阵
```

```
    print("\n 正在执行层次聚类以获取 linkage 矩阵...")
    linked = linkage(data_for_clustering, method='ward')
    # 2. 绘制树状图（已改为中文）
    plt.figure(figsize=(12, 6))
    plt.title('层次聚类树状图')
    dendrogram(linked, orientation='top', distance_sort='descending',
 show_leaf_counts=True)
    plt.xlabel('油滴序号')
    plt.ylabel('距离')
    plt.grid(True, alpha=0.3)
    plt.show()
    # 3. 优化版 K 值选择：融合手肘法+轮廓系数+间隙统计量
    if n_clusters is None:
        print("\n 正在使用多指标融合法寻找最佳聚类数 K...")
        # 合理设置 K 的搜索范围：至少 2 个簇，最多不超过数据量的 1/2（避免过
度聚类）
        min_k = 2
        max_k_to_test = min(15, n_samples // 2)  # 扩大搜索范围到 15
（适配更多簇场景）
        if max_k_to_test < min_k:
            max_k_to_test = min_k
        k_candidates = range(min_k, max_k_to_test + 1)
        # 计算三个核心指标
        wcss_list = []       # 手肘法：组内平方和
        silhouette_scores = []  # 轮廓系数：聚类质量
        gap_scores = []      # 间隙统计量：对比随机数据
        for k in k_candidates:
            labels = fcluster(linked, k, criterion='maxclust')

            # 计算 WCSS
            wcss = 0.0
            for i in range(1, k + 1):
                cluster_points = data_for_clustering[labels == i]
                if len(cluster_points) > 0:
                    center = cluster_points.mean(axis=0)
                    wcss += np.sum((cluster_points - center) ** 2)
            wcss_list.append(wcss)

            # 计算轮廓系数
            sil_score = calculate_silhouette_score(data_for_clusterin
g, labels)
            silhouette_scores.append(sil_score)

            # 计算间隙统计量（只算一次，避免重复计算）
            if k == k_candidates[0]:
                gap_scores = calculate_gap_statistic(data_for_cluster
ing, linked, max_k_to_test)
            else:
                gap_scores[k - min_k] = gap_scores[k - min_k]  # 复用
之前的计算结果
        # --- 指标 1：优化版手肘法（斜率变化率） ---
        wcss_diff = np.diff(wcss_list)  # 一阶导数（WCSS 下降幅度）
        wcss_diff_ratio = np.abs(wcss_diff[:-
1] / wcss_diff[1:])  # 斜率变化率（前一个下降幅度 / 后一个下降幅度）
```

```
        elbow_k_idx = np.argmax(wcss_diff_ratio) + min_k  # +min_k: 还原到原始K值
        # --- 指标2：轮廓系数最优K（分数最高） ---
        sil_best_k_idx = np.argmax(silhouette_scores)
        sil_best_k = k_candidates[sil_best_k_idx]
        # --- 指标3：间隙统计量最优K（分数最高） ---
        gap_best_k_idx = np.argmax(gap_scores[min_k - 1:max_k_to_test]) # 截取有效范围
        gap_best_k = min_k + gap_best_k_idx
        # --- 融合三个指标，确定最终K值 ---
        recommended_ks = [elbow_k_idx, sil_best_k, gap_best_k]
        # 处理众数不存在的情况（三个K值都不同），取中间值
        try:
            best_k = mode(recommended_ks)[0][0]
        except:
            best_k = sorted(recommended_ks)[1]  # 取中间值
        # 兜底：确保最佳K值在合理范围，且每个簇至少有2个数据（避免簇内数据过少）
        while True:
            labels_temp = fcluster(linked, best_k, criterion='maxclust')
            cluster_sizes = [sum(labels_temp == i) for i in range(1, best_k + 1)]
            if all(size >= 2 for size in cluster_sizes):
                break
            best_k -= 1
            if best_k < min_k:
                best_k = min_k
                break
        print(f"\n各指标推荐K值：")
        print(f"- 优化版手肘法推荐 K = {elbow_k_idx}")
        print(f"- 轮廓系数推荐 K = {sil_best_k}（分数：{silhouette_scores[sil_best_k_idx]:.3f}）")
        print(f"- 间隙统计量推荐 K = {gap_best_k}")
        print(f"融合后最终选择的最佳聚类数 K = {best_k}")
        # 绘制三个指标的可视化图（已改为中文）
        fig, axes = plt.subplots(1, 3, figsize=(18, 6))
        # 手肘法图（标注最佳K）
        axes[0].plot(k_candidates, wcss_list, marker='o', linestyle='-', color='b')
        axes[0].scatter(elbow_k_idx, wcss_list[elbow_k_idx - min_k], color='r', s=150, label=f'手肘点 K = {elbow_k_idx}')
        axes[0].set_title('优化版手肘法（组内平方和）')
        axes[0].set_xlabel('聚类数 (K)')
        axes[0].set_ylabel('组内平方和 (WCSS)')
        axes[0].grid(True, alpha=0.3)
        axes[0].legend()
        # 轮廓系数图
        axes[1].plot(k_candidates, silhouette_scores, marker='o', linestyle='-', color='g')
        axes[1].scatter(sil_best_k, silhouette_scores[sil_best_k_idx], color='r', s=150, label=f'最佳 K = {sil_best_k}')
        axes[1].set_title('轮廓系数（分数越高越好）')
        axes[1].set_xlabel('聚类数 (K)')
```

```
        axes[1].set_ylabel('轮廓系数')
        axes[1].grid(True, alpha=0.3)
        axes[1].legend()
        # 间隙统计量图
        axes[2].plot(range(1, max_k_to_test + 1), gap_scores, marker=
'o', linestyle='-', color='orange')
        axes[2].scatter(gap_best_k, gap_scores[gap_best_k - 1], color
='r', s=150, label=f'最佳 K = {gap_best_k}')
        axes[2].set_title('间隙统计量（分数越高越好）')
        axes[2].set_xlabel('聚类数 (K)')
        axes[2].set_ylabel('间隙值')
        axes[2].grid(True, alpha=0.3)
        axes[2].legend()
        plt.tight_layout()
        plt.show()

        n_clusters = best_k
    # 4. 根据 K 值进行最终聚类
    print(f"\n 使用 K = {n_clusters} 进行最终层次聚类...")
    labels = fcluster(linked, n_clusters, criterion='maxclust')
    cluster_labels = labels
    # 5. 计算聚类中心
    cluster_centers = []
    for i in range(1, n_clusters + 1):
        center = data_for_clustering[cluster_labels == i].mean(axis=0
)
        cluster_centers.append(center[0])
    cluster_centers = sorted(cluster_centers)
    print(f"\n 计算得到的簇中心 (已排
序): {[f'{c:.6e}' for c in cluster_centers]}")
    # 6. 推断元电荷 (核心优化保留)
    elementary_charge = 0.0
    if n_clusters > 1:
        differences = np.diff(cluster_centers)
        print(f"\n 相邻簇中心的原始差
值: {[f'{d:.6e}' for d in differences]}")
        if len(differences) > 1:
            median_diff = np.median(differences)
            mad = np.median(np.abs(differences - median_diff))
            if mad > 0:
                modified_z_scores = 0.6745 * (differences - median_di
ff) / mad
                filtered_diffs = differences[np.abs(modified_z_scores
) <= 3.5]

                if len(filtered_diffs) > 0:
                    print(f"检测到并去除
了 {len(differences) - len(filtered_diffs)} 个离群差值。")
                    print(f"过滤后的差
值: {[f'{d:.6e}' for d in filtered_diffs]}")
                    elementary_charge = np.median(filtered_diffs)
                else:
                    print("警告: 过滤后没有剩余的差值，将使用原始中位数。
")
```

```
                    elementary_charge = median_diff
            else:
                print("警告：中位数绝对偏差(MAD)为零，无法过滤离群值。")
                elementary_charge = median_diff

        elif len(differences) == 1:
            print("只有一个差值，直接使用该值作为元电荷估计。")
            elementary_charge = differences[0]

        print(f"使用稳健估计方法得到的元电荷
为：{elementary_charge:.6e} C")
    else:
        print("警告：仅检测到一个聚类。这可能导致元电荷计算不准确。")
        print("尝试使用所有油滴电荷量的最小值作为元电荷估计...")
        if charges:
            elementary_charge = min(charges)
            print(f"使用最小值估计的元电荷
为：{elementary_charge:.6e} C")
        else:
            elementary_charge = 0

    # 检查元电荷是否有效
    epsilon = 1e-25
    if abs(elementary_charge) < epsilon:
        print("\n 错误：计算出的元电荷值接近于零。")
        return None, cluster_centers, cluster_labels
    return elementary_charge, cluster_centers, cluster_labels
# --- 主程序 ---
def main():
    print("=" * 60)
    print("  密立根油滴实验 - 元电荷求解程序 (K 值优化版)")
    print("=" * 60)

    filename = input("\n 请输入数据文件的路径或名称 (例
如：data.txt):\n> ").strip()
    if not os.path.exists(filename):
        print(f"\n 错误：文件 '{filename}' 不存在或无法访问。")
        return
    print(f"\n 正在从文件 '{filename}' 加载数据... (电压范围：100V-
400V，时间范围：10s-40s)")
    oil_drops = load_and_filter_data(filename)
    if not oil_drops: return

    charges = [drop.charge for drop in oil_drops]

    print("\n 正在进行层次聚类分析...")
    result = infer_elementary_charge_hierarchical(charges)
    if result is None:
        print("\n 程序因无法计算有效元电荷而终止。")
        return

    elementary_charge, cluster_centers, cluster_labels = result
    # 结果分析和可视化（已改为中文）
    print("\n" + "=" * 60)
```

```
    print("                          实验结果分析")
    print("=" * 60)
    print(f"推断的元电荷值: {elementary_charge:.6e} C")
    print(f"公认理论值:    1.602176634e-19 C")
    relative_error = abs(elementary_charge - 1.602176634e-
19) / 1.602176634e-19 * 100
    print(f"相对误差:      {relative_error:.2f}%")

    # 绘图
    plt.figure(figsize=(14, 6))

    plt.subplot(1, 2, 1)
    plt.scatter(range(len(charges)), charges, c=cluster_labels, cmap=
'viridis', s=120, alpha=0.8, edgecolors='black')
    plt.axhline(y=elementary_charge, color='r', linestyle='--
', linewidth=2, label=f'元电荷: {elementary_charge:.6e} C')
    plt.title('油滴电荷量分布（层次聚类结果）', fontsize=14)
    plt.xlabel('油滴序号', fontsize=12)
    plt.ylabel('电荷量 (C)', fontsize=12)
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.subplot(1, 2, 2)
    plt.scatter(range(len(cluster_centers)), cluster_centers, s=200,
c='red', label='簇中心', edgecolors='black')
    theoretical_charges = [elementary_charge * (i+1) for i in range(l
en(cluster_centers))]
    plt.plot(range(len(theoretical_charges)), theoretical_charges, 'b
--', linewidth=2, label='理论倍数（1e, 2e, 3e...）')
    plt.title('簇中心与理论电荷量对比', fontsize=14)
    plt.xlabel('元电荷倍数', fontsize=12)
    plt.ylabel('电荷量 (C)', fontsize=12)
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 聚类详情
    print("\n" + "-" * 60)
    print("                          聚类分析详情")
    print("-" * 60)
    for i, center in enumerate(cluster_centers):
        count = sum(cluster_labels == (i+1))
        multiple = round(center / elementary_charge)
        print(f"聚类 {i+1}: 中心电荷 = {center:.6e} C, 油滴数
量 = {count:2d}, 约为 {multiple}倍元电荷")
    print("-" * 60)
if __name__ == "__main__":
    main()
input("按回车键退出...")
```

7.2.3 多重差值法

```Python
import pandas as pd
file_path = input("请输入 Excel 文件路径（例如：C:\\data\\file.xlsx）：")
df = pd.read_excel(file_path)
df=df.sort_values("电荷(10^(-19)C)")    #默认 ascending=True，对所有数据按电荷量升序排列
a=df["电荷(10^(-19)C)"]                 #取出电荷量，放进 a 中（Series，处理时与 list 同样操作）
n=len(df)
b=[]
sum=0
cnt=0
for i in range(n-1):
    delta=abs(a[i+1]-a[i])              #delta 即差值
    b.append(delta)                     #另存入其他 list，减少编程上覆盖数据后造成的麻烦
    if 1.4<=delta<=1.8:
        sum+=delta
        cnt+=1
while len(b)>1:
    c=[]                                #设置一个空列表 c，存储差值
    for i in range(len(b)-1):
        delta=abs(b[i+1]-b[i])
        c.append(delta)
        if 1.4<=delta<=1.8:
            sum+=delta
            cnt+=1
    b=c                                 #用 b 复制 c，防止占用内存过大
ave=sum/cnt                             #取平均
print(ave)
input("按任意键退出程序.....")
```

7.2.4 最大公约数法

```Python
import math, random
import pandas as pd
file_path = input("请输入 Excel 文件路径（例如：C:\\data\\file.xlsx）：")
df = pd.read_excel(file_path)
df=df.sort_values("电荷(10^(-19)C)",ascending=False)      #冒泡排序，结果为前大后小
a=df["电荷(10^(-19)C)"].tolist()
n=len(a)
#取得各组 q_i/q_j
B=[]
for i in range(n):
    b=[]
    for j in range(i):
        k=a[j]/a[i]
        b.append(k)
    B.append(b)
len_B=n*(n-1)/2
#得到 1/N~N/1
N=round(a[0]/a[n-1])
C=[]
for i in range(N,0,-1):
```

```
    c=[]
    for j in range(1,N+1,1):
        m=i/j
        c.append(m)
    C.append(c)
len_C=N**2
#比较整数比与电荷比
e=[]
cnt=[[0 for i in range(N)]for j in range (N)]
for i in range(n):
    for j in range(len(B[i])):
        for I in range(len(C)):
            for J in range(len(C[I])):
                if abs(C[I][J]-B[i][j])/B[i][j]<=0.001:
                    cnt[I][J]+=1
                    x=1
                    while((I+1)*x<=N & (J+1)*x<=N):
                        e1=round(a[j]/x/(I+1),4)
                        e2=round(a[i]/x/(J+1),4)
                        e.append(e1)
                        e.append(e2)
                        x+=1
                    break
#利用字典进行对应数据频次的记录
ecnt={}
for i in e:
    ecnt[i]=ecnt.get(i,0)+1
#输出 values 中最大值对应的 key
e_max=max(ecnt,key=ecnt.get)
print(e_max)
#print(ecnt)'''
input("按任意键退出程序.....")
```

7.2.5 LAD 法

```
Python
import pandas as pd
import math
file_path = input("请输入 Excel 文件路径（例如: C:\\data\\file.xlsx）: ")
df = pd.read_excel(file_path)
df=df.sort_values("电荷(10^(-19)C)")
a=df["电荷(10^(-19)C)"].tolist()              #tolist()函数将 Series 转
化为 list
n=len(a)
minq=e_j=e0=a[0]/math.sqrt(6*7)              #此处取 K=6
maxq=a[0]*1.25
eta=(maxq/minq)**(1/300)
E=[]                                          #列表 E 用于存储各次扫描得
到的残差的绝对值的平均数
while e_j <= maxq:                            #以 \eta 的倍率扫描
    sum=0
    for i in range(n):
        d=abs(a[i]-e_j*int(a[i]/e_j+0.49))
        sum+=d
    E_j=sum/n/e_j
    E.append(E_j)
```

```
    e_j*=eta
mine=100000                                    #储存最小值，初始设为大数以保
证能够被替换为实际的值
minindex=0
for i in range(len(E)):
    if E[i]<mine:
        mine=E[i]
        minindex=i
e_fin=e0*eta**minindex                         #E[i]存储了e0*eta**i 的残
差，由E[0]存储e0 可以验证
print(e_fin)
input("按任意键退出程序......")
```

7.2.6 枚举方差法

```
Python
import pandas as pd
file_path = input("请输入 Excel 文件路径（例如：C:\\data\\file.xlsx）：")
df = pd.read_excel(file_path)
sr=df["电荷(10^(-19)C)"]
def ave1(a):                                   #计算 a 的平均数
    s=cnt=0
    for i in a:
        s+=i
        cnt+=1
    ave=s/cnt
    return ave
def ave2(a):                                   #计算 a**2 的平均数，与 ave1 函数
一起计算方差
    s=cnt=0
    for i in a:
        s+=i**2
        cnt+=1
    ave=s/cnt
    return ave
def find_min(a):                               #遍历，求得最小方差对应的索引
    cal=[]
    min=a[0]
    min_index=0
    for i in range(1,len(a)):
        if a[i]<min:
            min=a[i]
            min_index=i
    return min_index
sig=[]
for e0 in range(1400, 1800):                   #限定 e 在 1.400~1.800 内，
以 0.001 的速率累加递增
    q=[]
    e=e0/1000
    for i in sr:                               #取整，计算单一数据点此处所对应的元
电荷
        n=round(i/e)
        q.append(i/n)
    s=ave2(q)-ave1(q)**2
    sig.append(s)
min_sig=find_min(sig)
```

```
m=(min_sig+1400)/1000                    #假设 min_sig = 2，表明结果应该为
1.402，故作此处理
print(m)
input("按任意键退出程序.....")
```

7.2.7 循环试商法

```
Python
import pandas as pd
file_path = input("请输入 Excel 文件路径（例如：C:\\data\\file.xlsx）： ")
df = pd.read_excel(file_path)
df=df.sort_values("电荷(10^(-19)C)")
def sig(a):
    x_ave1=x_ave2=0
    sum1=sum2=0
    for x in a:
        sum1+=x
        sum2+=x**2
    x_ave1=sum1/len(a)
    x_ave2=sum2/len(a)

    sig=x_ave2-x_ave1**2

    return sig
a=df["电荷(10^(-19)C)"]
n=len(df)
q=min(a)
for i in range(1,20):
    e=[]
    q_i=q/i
    for j in range(len(a)):
        r=a[j]/q_i
        n=round(r)
        e_j=a[j]/n
        e.append(e_j)
    d=sig(e)
    e_r=0
    if d<0.01:
        sum_e=0
        for k in e:
            sum_e+=k
            e_r=sum_e/len(e)
        break
print(e_r)
print(d)
input("按任意键退出程序.....")
```

7.3 课题分工

8 参考文献

[1]徐中炜,吕佩伟,施洋,等.基于机器视觉密立根油滴实验的教学探索与实践[J].大学物理实验,2024,37(04):101-107.DOI:10.14139/j.cnki.cn22-1228.2024.04.018.

[2]张家伟,陈学文,肖雨,等. 视频追踪技术在密立根油滴实验中的应用[J]. 大学物理实验,2025,38(4):44-49. DOI:10.14139/j.cnki.cn22-1228.2025.04.008.

[3]柳辛迪,罗昌玲,王荣超,等. 密立根油滴实验随机性研究[J]. 成都信息工程大学学报,2024,39(1):113-118. DOI:10.16836/j.cnki.jcuit.2024.01.017.

[4]范晓珍,李迟昊,吴闻彬,等. 基于枚举-方差法寻找密立根油滴 实验中的元电荷值[J]. 浙江师范大学学报（自然科学版）,2020,43(1):46-49. DOI:10.16218/j.issn.1001-5051.2020.01.008.

[5]宋元军. 密立根油滴实验数据处理的多重差值法[J]. 物理实验,2019,39(3):12-14. DOI:10.19655/j.cnki.1005-4642.2019.03.003.

[6]袁哲诚,王宽亮,陆李威,等. 密立根油滴实验的数据处理方法研究[J]. 物理实验,2019,39(5):13-16. DOI:10.19655/j.cnki.1005-4642.2019.05.003.

[7]朱鹤年,郭旭波,常缨,等. 油滴实验计算元电荷的简明方法[J]. 物理实验,2018,38(2):22-26,30. DOI:10.19655/j.cnki.1005-4642.2018.02.006.

[8]亓东林,鲍祎楠,张师平,等. 基于大数据分析思路的油滴实验数据处理方法[J]. 物理与工程,2018,28(6):91-94,99. DOI:10.3969/j.issn.1009-7104.2018.06.019.

[9]杨会静. 密立根油滴实验数据处理方法分析比较[J]. 唐山师范学院学报,2016,38(5):126-129. DOI:10.3969/j.issn.1009-9115.2016.05.038.

[10]陈森,刘昳,付硕,等. 一种密立根油滴实验数据处理的新方法[J]. 大学物理,2014,33(9):32-34,58.

[11]周海涛,唐美玲. 密立根油滴实验数据分析软件的设计[J]. 实验室科学,2009(3):73-75. DOI:10.3969/j.issn.1672-4305.2009.03.029.

[12]Ward, J. H., Jr. (1963). Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association, 58(301), 236-244.

[13]Sibson, R. (1973). SLINK: An optimally efficient algorithm for the single-link cluster method. The Computer Journal, 16(1), 30-45.

[14]Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 20, 53-65.