密里根实验报告

2026 年 1 月 3 日

目录

第一章　实验目的

1. 通过密立根油滴实验精确测量基本电荷 e 的数值；
2. 验证电荷的量子化特性（即任何带电体的电荷量都是基本电荷的整数倍）；
3. 使用 `uncertainties` 库的不确定度计算与 `pint` 库单位管理；
4. 展示 `phyexp` 库在实验数据处理中的应用例如带不确定度的加权线性拟合，简化计算流程。

第二章　实验原理

2.1　公式推导

2.1.1　油滴半径的计算

当油滴在重力和空气粘滞力作用下匀速下落时，受力平衡满足：

重力 $F_g = mg = \frac{4}{3}\pi a^3 \rho g$

粘滞力 $F_\eta = 6\pi\eta a v_g$（斯托克斯定律）

联立平衡条件 $F_g = F_\eta$，结合下落速度 $v_g = \frac{l}{t_g}$（l 为下落距离，t_g 为下落时间），解得油滴半径：

$$a = \sqrt{\frac{9\eta l}{2\rho g t_g}}$$

2.1.2　油滴电荷量的计算

当油滴在电场中静止时，电场力与重力平衡：

电场力 $F_e = qE = q\frac{U}{d}$（U 为平衡电压，d 为极板间距）

联立 $F_e = F_g$，代入油滴半径公式，最终得到电荷量：

$$q = \frac{18\pi}{\sqrt{2\rho g}} \left(\frac{\eta l}{t_g \left(1 + \frac{b}{pa}\right)} \right)^{\frac{3}{2}} \frac{d}{U}$$

2.2　基本电荷 e 的求解方法

1. **平均法**：由 $q = ne$ 得 $n = \text{round}\left(\frac{|q|}{|e_{\text{公认}}|}\right)$，再通过 $\bar{e} = \frac{\sum q}{\sum n}$ 求解；
2. **加权线性拟合**：以电荷数 n （同上）为横坐标、电荷量 q 为纵坐标，采用加权最小二乘拟合 $q = a + bn$，斜率 b 即为 e；
3. **作图法**：绘制 $n - q$ 方格图，若通过原点的一条射线穿过多个图中格点则验证电荷量子化，提取格点数据计算 $e_i = \frac{q_i}{n_i}$，取平均值作为测量结果。

第三章　实验仪器与常数

3.1　实验仪器

OM99 CCD 微机密里根油滴仪和喷雾器

3.2　实验常数

物理量	数值	单位	备注
油的密度 ρ	981	$\mathrm{kg \cdot m^{-3}}$	20℃ 时油的密度
重力加速度 g	9.801	$\mathrm{m \cdot s^{-2}}$	当地重力加速度
空气粘度 η	1.83×10^{-5}	$\mathrm{kg \cdot m^{-1} \cdot s^{-1}}$	20℃ 时空气粘度
下落距离 l	1.5 ± 0.01	mm	转换为 1.5×10^{-3}m，不确定度 0.01mm
修正系数 b	8.224×10^{-3}	$\mathrm{m \cdot Pa}$	空气分子修正系数
大气压 p	1.013×10^{5}	Pa	标准大气压
极板间距 d	5×10^{-3}	m	极板间距离
电压仪器误差限 ΔU	0.5	V	平衡电压测量误差
反应时间误差限 Δt	0.005	s	人工计时反应时间误差
基本电荷公认值 $e_{\text{公认}}$	$-1.602176634 \times 10^{-19}$	C	国际公认值

第四章　原始数据

测量数据记录（$t = 20^{\circ}\mathrm{C}$）			
测量序号	平衡电压 U/V	下落时间 t_g/s（多次测量）	时间均值 $\overline{t_g}$/s
1	118	10.61, 10.57, 10.44, 10.62, 10.27	10.502
2	103	8.34, 8.30, 8.26, 8.30, 8.42	8.324
3	149	9.84, 10.04, 9.77, 9.94, 10.12	9.942
4	101	11.11, 11.39, 11.21, 10.93, 11.04	11.136
5	100	12.90, 12.79, 13.11, 12.81, 13.09	12.940
6	124	19.37, 18.51, 19.41, 19.02, 18.84	19.030
7	107	11.06, 11.13, 11.01, 10.93, 11.24	11.074
8	105	21.19, 21.50, 21.16, 20.83, 21.44	21.224
9	100	9.28, 9.22, 9.18, 9.16, 9.21	9.210
10	127	18.87, 19.15, 19.11, 19.01, 18.78	18.984

第五章　数据处理

In [1]:
```
import numpy as np
import phyexp
from phyexp import SLR, plt
from phyexp.AB_uncert import A_uncert, 仪器误差限转 B 类不确定度, 不确定度合成
from phyexp.meas import 一次测量结果, 多次测量结果, ureg, Q_ # 单位
from phyexp.error import 相对误差
from uncertainties import unumpy as unp # 不确定度
from uncertainties import ufloat # 不确定度
from matplotlib.ticker import AutoMinorLocator
```

In [2]:
```
t_raw = np.array([
    [10.61, 10.57, 10.44, 10.62, 10.27],
    [8.34, 8.30, 8.26, 8.30, 8.42],
    [9.84, 10.04, 9.77, 9.94, 10.12],
    [11.11, 11.39, 11.21, 10.93, 11.04],
    [12.90, 12.79, 13.11, 12.81, 13.09],
    [19.37, 18.51, 19.41, 19.02, 18.84],
    [11.06, 11.13, 11.01, 10.93, 11.24],
    [21.19, 21.50, 21.16, 20.83, 21.44],
    [9.28, 9.22, 9.18, 9.16, 9.21],
    [18.87, 19.15, 19.11, 19.01, 18.78]
])*ureg('s')

u_raw = np.array([118, 103, 149, 101, 100, 124, 107, 105, 100, 127])*ureg('V')
```

In [3]:
```
# 实验常数定义（带单位）
rho = Q_(981, 'kg/m^3')  # 油的密度
g = Q_(9.801, 'm/s^2')  # 重力加速度
eta = Q_(1.83e-5, 'Pa*s')  # 空气粘度
b = Q_(8.224e-3, 'm • Pa')  # 修正系数
```

```
p = Q_(1.013e5, 'Pa')  # 大气压
d = Q_(5e-3, 'm')  # 极板间距
e_known = Q_(-1.602176634e-19, 'C')  # 基本电荷公认值
```

为简单起见，我们把来自人眼判断是否与线重合及停止计时的反应时间的误差合并, 取估计值 0.01mm

```
In [4]: l = 一次测量结果 (1.5, 'mm', 仪器误差限转 B 类不确定度 (0.01)).to('m') # 下落距离，转换为

In [5]: t=np.array([多次测量结果 (data.magnitude,'s',B 类不确定度=仪器误差限转 B 类不确定度 (0.5)
        u=np.array([一次测量结果 (data.magnitude, 'V', 仪器误差限转 B 类不确定度 (0.5)) for data

In [6]: print("时间均值: ")
        for i in range(10):
            print(f"第{i+1}组: {t[i]}")
        print("电压: ")
        for i in range(10):
            print(f"第{i+1}组: {u[i]}")
```

```
时间均值:
第 1 组: 10.50+/-0.30 second
第 2 组: 8.32+/-0.29 second
第 3 组: 9.94+/-0.30 second
第 4 组: 11.14+/-0.30 second
第 5 组: 12.94+/-0.30 second
第 6 组: 19.03+/-0.33 second
第 7 组: 11.07+/-0.29 second
第 8 组: 21.22+/-0.31 second
第 9 组: 9.21+/-0.29 second
第 10 组: 18.98+/-0.30 second
电压:
第 1 组: 118.00+/-0.29 volt
第 2 组: 103.00+/-0.29 volt
第 3 组: 149.00+/-0.29 volt
第 4 组: 101.00+/-0.29 volt
第 5 组: 100.00+/-0.29 volt
第 6 组: 124.00+/-0.29 volt
第 7 组: 107.00+/-0.29 volt
第 8 组: 105.00+/-0.29 volt
```

```
第 9 组: 100.00+/-0.29 volt
第 10 组: 127.00+/-0.29 volt
```

```
In [7]: def 计算油滴半径(t):
            """ 根据时间 t 计算油滴半径 a"""
            a = np.array([(9 * eta * l / (2 * rho * g * i))**0.5 for i in t])
            return a
```

```
In [8]: a=计算油滴半径 (t)
        print("油滴半径: ")
        for i in range(10):
            a[i].ito('micrometre')
            print(f"第{i+1}组: {a[i]}")
```

```
油滴半径:
第 1 组: 1.106+/-0.016 micrometer
第 2 组: 1.242+/-0.022 micrometer
第 3 组: 1.137+/-0.017 micrometer
第 4 组: 1.074+/-0.015 micrometer
第 5 组: 0.996+/-0.012 micrometer
第 6 组: 0.822+/-0.007 micrometer
第 7 组: 1.077+/-0.014 micrometer
第 8 组: 0.778+/-0.006 micrometer
第 9 组: 1.181+/-0.019 micrometer
第 10 组: 0.823+/-0.007 micrometer
```

```
In [9]: def 计算电荷量(t, u, a):
            """ 根据密立根公式计算电荷量 q（带单位）"""
            term1 = 18 * np.pi / (2 * rho * g)**0.5
            term2 = (eta * l) / (t * (1 + b/(p*a)))  # 含修正项
            term3 = d / u
            q = term1 * (term2 ** 1.5) * term3
            return q

        q = np.array([计算电荷量 (t[i], u[i], a[i]) for i in range(10)])
```

```
In [10]: print("电荷量 q: ")
         for i in range(10):
```

```
            q[i].ito('C')
            print(f"第{i+1}组: {q[i]}")
```

```
电荷量 q:
第 1 组: (2.08+/-0.09)e-18 coulomb
第 2 组: (3.41+/-0.18)e-18 coulomb
第 3 组: (1.79+/-0.08)e-18 coulomb
第 4 组: (2.21+/-0.09)e-18 coulomb
第 5 组: (1.77+/-0.06)e-18 coulomb
第 6 组: (7.82+/-0.22)e-19 coulomb
第 7 组: (2.11+/-0.09)e-18 coulomb
第 8 组: (7.78+/-0.19)e-19 coulomb
第 9 组: (3.00+/-0.15)e-18 coulomb
第 10 组: (7.67+/-0.19)e-19 coulomb
```

```
In [11]: n = np.array([round(abs(qq.magnitude.n / e_known.magnitude)) for qq in q])
```

```
        print("电荷数 n（整数）: ")
        for i in range(10):
            print(f"第{i+1}组: n={n[i]}")
```

```
电荷数 n（整数）:
第 1 组: n=13
第 2 组: n=21
第 3 组: n=11
第 4 组: n=14
第 5 组: n=11
第 6 组: n=5
第 7 组: n=13
第 8 组: n=5
第 9 组: n=19
第 10 组: n=5
```

5.1　平均法

```
In [12]: e1=q.sum()/n.sum()
```

```
In [13]: error1=相对误差 (e1,-e_known,True)

In [14]: print(f"平均法求得 |e1| = {e1}")
         print(f"平均法相对误差: {error1}")
```

```
平均法求得 |e1| = (1.598+/-0.028)e-19 coulomb
平均法相对误差: -0.241%
```

5.2　加权线性拟合

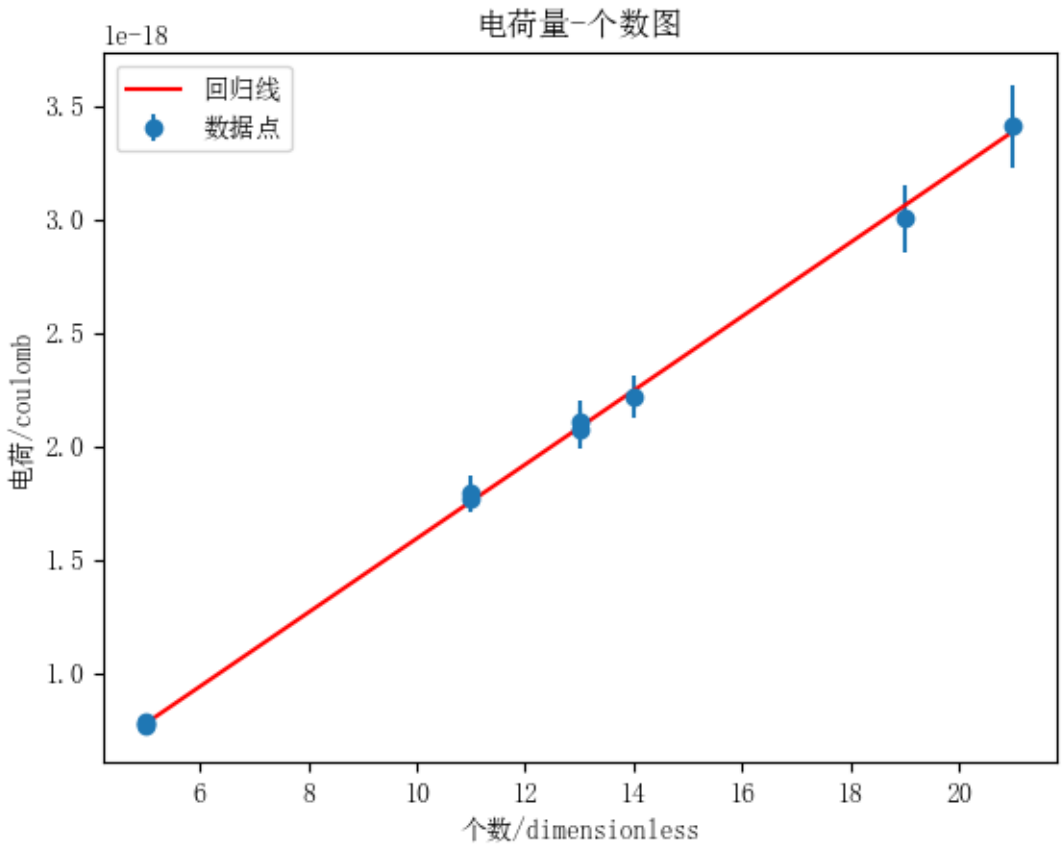
```
In [15]: y=np.array([i.magnitude for i in q])
         x=n

In [16]: a,b=SLR.一元线性回归 (x,y)

In [17]: e2=Q_(b,'C')
         error2=相对误差 (e2,-e_known,True)
         print(f"平均法求得 |e2| = {e2}")
         print(f"平均法相对误差: {error2}")
```

```
加权线性拟合求得 |e2| = (1.63+/-0.05)e-19 coulomb
加权线性拟合相对误差: 1.698%
```

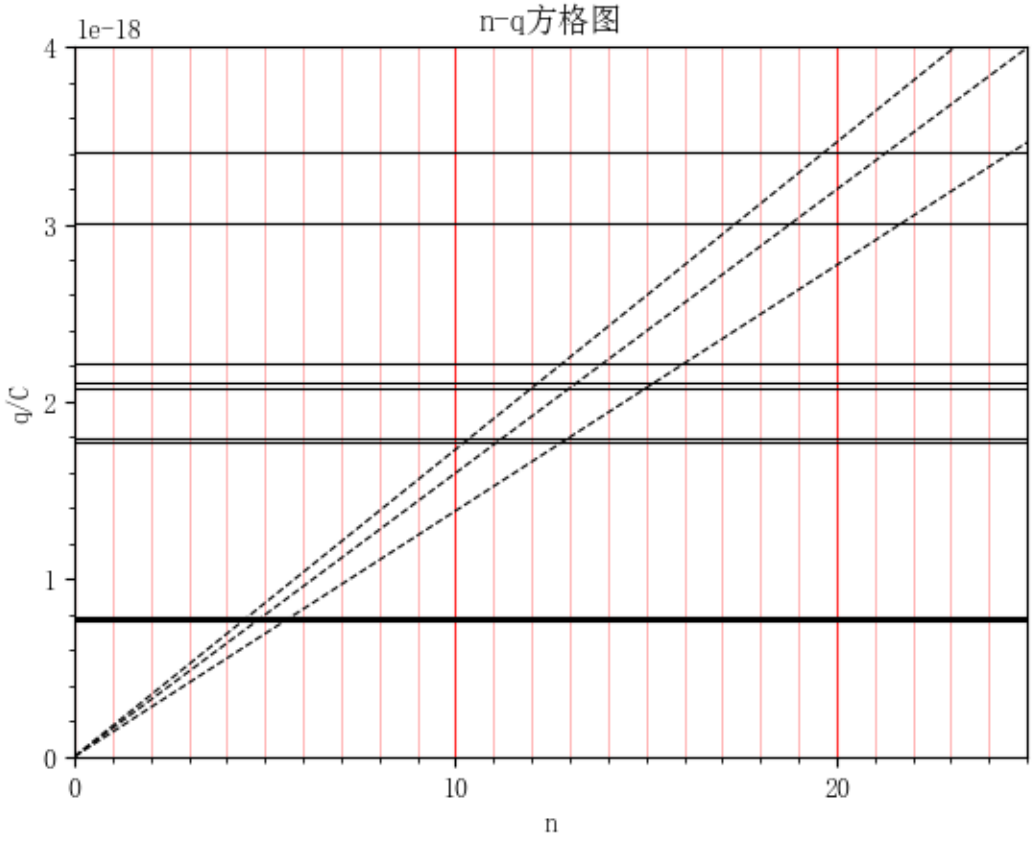
```
In [18]: SLR.绘制回归图 (Q_(x),q,"电荷量-个数图","个数","电荷")
```



5.3　作图法

```
In [19]: for i in y:
             plt.plot([0,25],[i.n,i.n],"k",linewidth=0.8)
         plt.title("n-q 方格图")
         plt.xlabel("n")
         plt.ylabel("q/C")
         plt.xlim([0,25])
         plt.ylim([0,4e-18])
         plt.xticks([0,10,20])
         plt.yticks([0,1e-18,2e-18,3e-18,4e-18])
         plt.minorticks_on()
         plt.gca().xaxis.set_minor_locator(AutoMinorLocator(n=10))
         # plt.gca().yaxis.set_minor_locator(AutoMinorLocator(n=10))
         plt.grid(axis='x',which='major',  linewidth=0.8, color='r', alpha=0.8)  # 主网格
```

```
plt.grid(axis='x',which='minor',  linewidth=0.4, color='r', alpha=0.5)  # 次网格
for i in [12,13,15]:
    plt.plot([0,i,25],[0,y[0].n,25/i*y[0].n],"--",c="k",linewidth=0.8)
plt.show()
```



中间的射线更接近所有格点，取所有格点的平均，结果与平均法相同。本方法优势是不需要已知基本电荷值，可以验证电荷的量子性。

第六章　实验结果与分析

6.1　核心结果汇总

方法	基本电荷 e（含不确定度）	相对误差
平均法	$(1.60 \pm 0.03) \times 10^{-19}\mathrm{C}$	0.24%
加权线性拟合	$(1.63 \pm 0.05) \times 10^{-19}\mathrm{C}$	1.7%
作图法	$(1.60 \pm 0.03) \times 10^{-19}\mathrm{C}$	0.24%

6.2　结果分析

1. 三种方法测得的基本电荷 e 均与公认值 $1.602176634 \times 10^{-19}\mathrm{C}$ 接近，相对误差均小于 2%，验证了电荷的量子化特性；
2. 作图法通过 $n - q$ 方格图直观验证了 “电荷量是基本电荷整数倍”；

第七章　实验结论

1. 本实验通过密立根油滴法成功测量了基本电荷 $ e $，三种方法的测量结果均与公认值一致，其中平均法的相对误差最小（0.24%），最终实验结果为 $ e = (1.60 \pm 0.03) \times 10^{-19} , C $；
2. 实验结果验证了电荷的量子化特性，即带电体的电荷量是基本电荷的整数倍；
3. `uncertainties` 库实现了便捷的不确定度计算，`pint` 库规范了物理量单位管理，`phyexp` 库简化了加权线性拟合等复杂数据处理流程，提高了实验效率和计算准确性，适用于物理实验中的数据处理场景。

附录 A　使用的工具与依赖库

A.1　开发与编辑环境

本实验的代码编写、交互式运行与结果调试均基于 `Jupyter Notebook`（v7.5.1）完成，该工具为 Python 生态的交互式计算环境，支持代码分段执行、实时可视化与文本注释结合，适配物理实验数据处理的迭代式分析需求。

A.2　核心依赖库

实验数据处理代码中使用的主要 Python 库及版本如下：

1. `numpy`（v2.3.5）：用于数组运算、数值计算（如均值/标准差求解等）；
2. `matplotlib`（v3.10.6）：用于实验结果的可视化（如电荷量-电荷数关系图、不确定度误差棒图等）；
3. `uncertainties`（v3.2.3）：用于带不确定度的数值计算与误差传播分析；
4. `pint`（v0.25）：用于物理量的单位定义、转换与维度一致性校验。

A.3　官方文档

官方文档参考：

- Jupyter Notebook：`https://jupyter-notebook.readthedocs.io/en/stable/`；
- numpy：`https://numpy.org/doc/stable/`；
- matplotlib：`https://matplotlib.org/stable/contents.html`；
- uncertainties：`https://uncertainties.readthedocs.io/en/latest/`；
- pint：`https://pint.readthedocs.io/en/stable/`。

附录 B　phyexp 库主要代码说明

phyexp 是面向物理实验场景开发的轻量级 Python 工具库，旨在简化物理实验数据处理流程，封装了不确定度计算、测量值处理、误差分析、线性回归等实验数据处理的核心功能，降低重复编码成本。

本库的远程代码仓库地址为：`https://gitee.com/Log-Dog012/phyexp`。

以下将逐一说明库中核心代码模块的实现逻辑与功能用途：

1. 单位与不确定度基础工具（phyexp/utils.py）

```
"""
物理量单位与不确定度基础工具
"""
from pint import UnitRegistry

ureg = UnitRegistry(auto_reduce_dimensions=True)
Q_ = ureg.Quantity

# pint的Measurements不具备误差传播能力
from uncertainties import ufloat
from uncertainties.unumpy import uarray
```

2. 绘图配置（phyexp/plt.py）

```
"""
绘图配置工具
"""
from matplotlib.pyplot import *

# 设置中文字体和负号正常显示
rcParams["font.family"] = ["SimSun"]
rcParams["axes.unicode_minus"] = False
```

3. 带不确定度的一元线性回归（phyexp/SLR.py）

```
"""
带不确定度的一元线性回归
"""

from .utils import ureg, Q_
from .AB_uncert import 不确定度合成
from functools import wraps
import numpy as np
from . import plt
from .error import 提取标称值

def 提取不确定度(带不确定度的数值):
    """提取带不确定度的数值的不确定度。"""
    if hasattr(带不确定度的数值, "magnitude") and hasattr(带不确定度的数值, "units"):
        if hasattr(带不确定度的数值.magnitude, "nominal_value") and hasattr(带不确定度的数值.magnitude, "std_dev"):
            return Q_(带不确定度的数值.magnitude.std_dev, 带不确定度的数值.units)
    elif hasattr(带不确定度的数值, "nominal_value") and hasattr(带不确定度的数值, "std_dev"):
        return 带不确定度的数值.std_dev
    raise ValueError("输入的数值不包含不确定度信息。")

def 一元线性回归(x, y):
    """
    带不确定度的一元线性回归
    参数:
        x: 自变量数组，元素可以是带不确定度的数值。
        y: 因变量数组，元素必须是带不确定度的数值。
    返回:
        截距 a 和斜率 b，与输入同种类。
    备注:
        使用加权最小二乘法进行回归，权重为因变量的不确定度的倒数。
    """
    if len(x) != len(y):
        raise ValueError("x 和 y 的长度必须相等。")

    w=np.array([1/提取不确定度(yi) for yi in y],dtype=object)

```

```
    a=((w*y).sum()*(w*x**2).sum()-(w*x).sum()*(w*x*y).sum())/(w.sum()*(w*x**2)
        .sum()-((w*x).sum())**2)
    b=(w.sum()*(w*x*y).sum()-(w*y).sum()*(w*x).sum())/(w.sum()*(w*x**2).sum()
        -((w*x).sum())**2)

    return a,b

def 绘制回归图(x:Q_, y:Q_, title=None, xlabel=None, ylabel=None):
    """
    绘制带不确定度的一元线性回归图
    参数:
        x: 自变量数组，元素可以是带不确定度的数值。
        y: 因变量数组，元素必须是带不确定度的数值。
    """
    a, b = 一元线性回归(x, y)

    if title is None:
        title = "带不确定度的一元线性回归图"
    if xlabel is None:
        xlabel = "自变量"
    if ylabel is None:
        ylabel = "因变量"

    x_vals = np.linspace(min(x).magnitude, max(x).magnitude, 100)
    y_vals = a.n + b.n * x_vals

    plt.errorbar(
        [xi.magnitude for xi in x],
        [提取标称值(yi).magnitude for yi in y],
        yerr=[提取不确定度(yi).magnitude for yi in y],
        fmt='o', label='数据点'
    )
    plt.plot(x_vals, y_vals, 'r-', label='回归线')
    plt.xlabel(f'{xlabel}/{x.units}')
    y=Q_([yi.magnitude for yi in y],y[0].units)
    plt.ylabel(f'{ylabel}/{y.units}')
    plt.title(title)
    plt.legend()
    plt.show()
```

4. 实验误差计算（phyexp/error.py）

```python
"""
实验误差计算工具
"""

from .utils import ureg, Q_
from functools import wraps


def 提取标称值(带不确定度的数值):
    """提取带不确定度的数值的标称值（即测量值）。"""
    if hasattr(带不确定度的数值, "magnitude") and hasattr(带不确定度的数值, "units"):
        if hasattr(带不确定度的数值.magnitude, "nominal_value") and hasattr(
            带不确定度的数值.magnitude, "std_dev"
        ):
            return Q_(带不确定度的数值.magnitude.nominal_value, 带不确定度的数值.units)
    elif hasattr(带不确定度的数值, "nominal_value") and hasattr(
        带不确定度的数值, "std_dev"
    ):
        return 带不确定度的数值.nominal_value
    return 带不确定度的数值


def 预处理(func):
    """装饰器：预处理函数参数，提取带不确定度数值的标称值。"""

    @wraps(func)
    def wrapper(*args, **kwargs):
        新args = [提取标称值(arg) for arg in args]
        新kwargs = {k: 提取标称值(v) for k, v in kwargs.items()}
        return func(*新args, **新kwargs)

    return wrapper


@预处理
def 相对误差(测量值, 真值, str: bool = False, n=3):
    """计算测量值与真值之间的误差。"""
    if str:
```

```
        tem = (测量值 - 真值) / 真值
        if hasattr(tem, "magnitude"):
            tem = tem.magnitude
        return f"{tem:.{n}%}"
    else:
        return (测量值 - 真值) / 真值
```

5. 包初始化（phyexp/___init___.py）

```
from .AB_uncert import 求A类不确定度 as A_uncert
```

6. 测量结果处理（phyexp/meas.py）

```
"""
测量量处理工具
"""

from .utils import ureg, Q_, ufloat
from typing import Sequence
from .AB_uncert import 求A类不确定度, 不确定度合成
from uncertainties.core import UFloat, AffineScalarFunc
import numpy as np


def 一次测量结果(数值, 单位: str = "", B类不确定度: float = 0.0, 名称: str = ""):
    # 一次测量结果的不确定度等于其B类不确定度
    带不确定度的数值 = ufloat(数值, B类不确定度, tag=名称 if 名称 else None)
    obj = Q_(带不确定度的数值, 单位)
    return obj


def 多次测量结果(数值列表, 单位: str = "", B类不确定度: float = 0.0, 名称: str = ""):
    # 多次测量结果的不确定度等于A类不确定度与B类不确定度合成
    数值列表 = 数值列表.magnitude if isinstance(数值列表, Q_) else np.array(数值列表)
    A类不确定度 = 求A类不确定度(数值列表)
    不确定度 = 不确定度合成(A类不确定度, B类不确定度)
    平均值 = 数值列表.mean()
    带不确定度的数值 = ufloat(平均值, 不确定度, tag=名称 if 名称 else None)
```

```
    obj = Q_(带不确定度的数值, 单位)
    return obj
```

7. 有效数字修约（phyexp/sigfigs.py）

```
"""
有效数字修约工具
"""
from uncertainties import UFloat
from uncertainties import ufloat
from math import log10, floor


def 修约(带不确定度数值: UFloat, 字符串形式=True):
    """修约一个不确定度数值到其有效数字位数。

    参数:
        带不确定度数值 (UFloat): 需要修约的不确定度数值。

    返回:
        UFloat: 修约后的不确定度数值。
    """
    if 带不确定度数值.std_dev == 0:
        if 字符串形式:
            return f"{带不确定度数值:g}"
        else:
            return 带不确定度数值
    else:
        位 = -floor(log10(abs(带不确定度数值.std_dev)))
        修约值 = ufloat(
            round(带不确定度数值.nominal_value, 位), round(带不确定度数值.
                std_dev, 位)
        )
        if 字符串形式:
            return f"{修约值:g}"
        else:
            return 修约值


# 无法处理单位

```

```

from uncertainties import ufloat_fromstr as u
from uncertainties.umath import *

"""
用u来创建需考虑不确定度的数值，并直接使用数学函数进行计算
可以用dir()查看可用的函数
"""
```

8. 不确定度计算核心（phyexp/AB_uncert.py）

```
"""
物理实验中涉及的不确定度计算
"""

import numpy as np
import types
import warnings


def 可向量化(数字列表):
    """
    将不支持向量化的列表转换为numpy数组。

    参数:
    数字列表: 一组数字。

    返回:
    numpy数组(dtype可以为object)或原始输入。
    """
    attrs = [
        "__len__",
        "mean",
        "std",
    ]
    test = all(hasattr(数字列表, attr) for attr in attrs)
    if test:
        return 数字列表
    else:
        try:
            result = np.asarray(数字列表)
```

```
        except Exception as e:
            try:
                result = np.asarray(数字列表, dtype=object)
            except Exception:
                raise TypeError(f"无法将输入转换为numpy数组: {e}")

        return result


def 输入转换(func):
    """
    装饰器: 将输入调整为不确定度合成所需类型。

    参数:
    func: 需要装饰的函数。

    返回:
    function: 装饰后的函数。
    """

    def wrapper(*分量):
        新分量 = [可向量化(i) for i in 分量]
        return func(*新分量)

    return wrapper


def generator_to_list_warning(func):
    """
    装饰器: 如果输入是生成器, 则转换为列表并发出警告。

    参数:
    func: 需要装饰的函数。

    返回:
    function: 装饰后的函数。
    """

    def wrapper(测量值列表):
        if isinstance(测量值列表, types.GeneratorType):
            测量值列表 = list(测量值列表)
```

```
            warnings.warn(
                "生成器已被迭代并转换为列表，后续无法再次使用此生成器（生成器是一次性迭代对象）。",
                UserWarning,  # 警告类型（用户级警告，最常用）
                stacklevel=2,  # 控制警告的栈层级，让用户看到自己的代码行（而非函数内部）
            )
        return func(测量值列表)

    return wrapper


@generator_to_list_warning
@输入转换
def 求A类不确定度(测量值列表):
    """
    计算A类不确定度，基于测量值均值的样本标准差（贝塞尔公式）。

    参数:
    测量值列表: 一组测量值（支持列表、元组、np数组、pint带单位量、pd.Series等可迭代对象）。

    返回:
    float/np.float64/pint.Quantity: A类不确定度（输入带单位则返回带单位结果）。
    """

    if len(测量值列表) < 2:
        raise ValueError("测量值列表至少应包含两个值以计算A类不确定度。")
    标准偏差 = 测量值列表.std(ddof=1) / (len(测量值列表) ** 0.5)

    return 标准偏差


A_uncert = 求A类不确定度

# 支持的分布及其对应的包含因子K
包含因子 = {
    "均匀": 3**0.5,
    "正态": 3,  # 3 原则
}
```

```
109
110
111 # 这样的函数只是为了使用时更直观
112 def 仪器误差限转B类不确定度(仪器误差限, 分布="均匀", K=3**0.5):
113     """
114     将仪器误差限转换为B类不确定度。
115     参数:
116     仪器误差限: 仪器误差限值（正数）。
117     分布: 仪器误差的分布类型，支持"均匀"和"正态"。默认值为"均匀"。
118     K: 如果分布类型未知，可直接提供分布因子K（正数）。默认值为3的平方根（对应
        均匀分布）。
119     返回:
120     float: B类不确定度。
121     """
122
123     if 分布 in 包含因子:
124         K = 包含因子[分布]
125     elif 分布 is not None:
126         现有分布类型 = list(包含因子.keys())
127         raise ValueError(f"未知分布类型'{分布}', 目前只支持{现有分布类型}。")
128     elif K <= 0:
129         raise ValueError("分布因子K应为正数。")
130     elif K > 0:
131         pass
132     else:
133         raise ValueError("未知错误。")
134
135     return 仪器误差限 / K
136
137
138 InstErr_to_B_uncert = 仪器误差限转B类不确定度
139
140
141 # 这个函数只是为了使用时更直观
142 def 不确定度合成(A分量, B分量):
143     """
144     计算不确定度合成，基于各分量的不确定度平方和的平方根。
145     只能处理标量输入和数组输入。
146
147     参数:
148     分量: AB分量。
```

```
149
150     返回:
151     float: 合成不确定度。
152     """
153     return (A分量**2 + B分量**2) ** 0.5
154
155
156 uncert_comb = 不确定度合成
```