散件时代

1 第一版

```python
import pandas as pd
import matplotlib.pyplot as plt

# 解决 Matplotlib 中文显示问题
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 例如使用黑体（SimHei）
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取 Excel 文件
file_path = "./datas_learn/B1.xlsx"  # 请确保该文件在当前目录
xls = pd.ExcelFile(file_path)

# 选择第一个 sheet
df = xls.parse(sheet_name=0)

# 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

# 重新命名列名
df_filtered.columns = ["SMU-1 电压（V）", "SMU-1 电流（A）"]

# 转换数据为数值类型（防止字符串干扰）
df_filtered = df_filtered.astype(float)

# 提取自变量（电压）和因变量（电流）
x = df_filtered["SMU-1 电压（V）"]
y = df_filtered["SMU-1 电流（A）"]

print("df_filtered: ", df_filtered.head())  # 打印前几行查看数据

# 绘制曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', linestyle='-', color='b', label="Voltage-Current Curve")
plt.xlabel("SMU-1 电压（V）")
plt.ylabel("SMU-1 电流（A）")
plt.title("电压-电流曲线")
plt.legend()
plt.grid(True)

# 显示图像
plt.show()

```

2 第二版

```python
import pandas as pd
import matplotlib.pyplot as plt

# 解决 Matplotlib 中文显示问题
import matplotlib
```

```
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 例如使用黑体（SimHei）
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取 Excel 文件
file_path = "./datas_learn/B1.xlsx"  # 请确保该文件在当前目录
xls = pd.ExcelFile(file_path)

# 选择第一个 sheet
df = xls.parse(sheet_name=0)

# 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

# 重新命名列名
df_filtered.columns = ["SMU-1 电压（V）", "SMU-1 电流（A）"]

# 转换数据为数值类型（防止字符串干扰）
df_filtered = df_filtered.astype(float)

# 提取自变量（电压）和因变量（电流）
x = df_filtered["SMU-1 电压（V）"]
y = df_filtered["SMU-1 电流（A）"]

print("df_filtered: ", df_filtered.head())  # 打印前几行查看数据

# 绘制曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', linestyle='-', color='b', label="Voltage-Current Curve")
plt.xlabel("SMU-1 电压（V）")
plt.ylabel("SMU-1 电流（A）")
plt.title("电压-电流曲线")
plt.legend()
plt.grid(True)

# 显示图像
plt.show()
```

3 基于第二版修改列名

```
import pandas as pd
import matplotlib.pyplot as plt

# 解决 Matplotlib 中文显示问题
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 例如使用黑体（SimHei）
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取 Excel 文件
file_path = "./datas_learn/B1.xlsx"  # 请确保该文件在当前目录
xls = pd.ExcelFile(file_path)

# 选择第一个 sheet
df = xls.parse(sheet_name=0)

# 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
```

```
df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列 (0-based 索引)

# 重新命名列名
df_filtered.columns = ["电压 (V)", "电流 (A)"]

# 转换数据为数值类型 (防止字符串干扰)
df_filtered = df_filtered.astype(float)

# 提取自变量 (电压) 和因变量 (电流)
x = df_filtered["电压 (V)"]
y = df_filtered["电流 (A)"]

print("df_filtered: ", df_filtered.head())  # 打印前几行查看数据

# 绘制曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', linestyle='-', color='b', label="V - A Curve")
plt.xlabel("电压 (V)")
plt.ylabel("电流 (A)")
plt.title("电压-电流曲线")
plt.legend()
plt.grid(True)

# 显示图像
plt.show()
```

4 基于第二版+写出拟合函数表达式

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 解决 Matplotlib 中文显示问题
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 例如使用黑体 (SimHei)
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取 Excel 文件
file_path = "./datas_learn/B2-3-21-30.xlsx"  # 请确保该文件在当前目录
xls = pd.ExcelFile(file_path)

# 选择第一个 sheet
df = xls.parse(sheet_name=0)

# 提取 C35 到 D536 的数据 (假设列名在第 35 行开始)
df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列 (0-based 索引)

# 重新命名列名
df_filtered.columns = ["电压 (V)", "电流 (A)"]

# 转换数据为数值类型 (防止字符串干扰)
df_filtered = df_filtered.astype(float)

# 提取自变量 (电压) 和因变量 (电流)
x = df_filtered["电压 (V)"]
```

```python
y = df_filtered["电流 (A)"]

# 多项式拟合函数
def polynomial_fit(x, y, degree=3):
    p = np.polyfit(x, y, degree)  # 拟合多项式
    poly = np.poly1d(p)
    y_fit = poly(x)
    return p, y_fit

# 调用多项式拟合
degree = 10  # 可以调整为你需要的拟合次数
p, y_fit = polynomial_fit(x, y, degree)

# 打印拟合的多项式系数
print(f"拟合多项式的系数（从高次到低次）：{p}")

# 准备拟合函数的字符串表达式
equation_str = " + ".join([f"{coef:.2e}x^{degree-i}" for i, coef in enumerate(p)])

# 绘制数据和拟合曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据")
plt.plot(x, y_fit, linestyle='--', color='r', label=f"{degree}次多项式拟合")

# 在图形中添加拟合函数的表达式
plt.text(0.1, 0.1, f"拟合函数: y = {equation_str}", transform=plt.gca().transAxes,
fontsize=12, verticalalignment='top')

# 图表设置
plt.xlabel("电压 (V)")
plt.ylabel("电流 (A)")
plt.title("电压-电流曲线及拟合")
plt.legend()
plt.grid(True)

# 显示图像
plt.show()

```

5 增加了决定系数R^2

为了衡量拟合效果，常用的指标是 **决定系数 (R^2)**，它能够衡量拟合曲线与实际数据的接近程度。R^2 值越接近 1，说明拟合效果越好。

R^2 的计算公式为：

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

其中：

- y_i 是原始数据点的实际值，
- $\hat{y}_i$ 是拟合曲线上的预测值，
- $\bar{y}$ 是实际数据的均值。

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 解决 Matplotlib 中文显示问题
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 使用微软雅黑（或其他支持上标的字体）
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取 Excel 文件
# file_path = "./datas_learn/B1.xlsx"  # 请确保该文件在当前目录
# file_path = "./datas_learn/B2-3-21-30.xlsx"  # 请确保该文件在当前目录
file_path = "./datas_learn/T2.xlsx"  # 请确保该文件在当前目录
xls = pd.ExcelFile(file_path)

# 选择第一个 sheet
df = xls.parse(sheet_name=0)

# 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

# 重新命名列名
df_filtered.columns = ["电压（V）", "电流（A）"]

# 转换数据为数值类型（防止字符串干扰）
df_filtered = df_filtered.astype(float)

# 提取自变量（电压）和因变量（电流）
x = df_filtered["电压（V）"]
y = df_filtered["电流（A）"]

# 多项式拟合函数
def polynomial_fit(x, y, degree=3):
    p = np.polyfit(x, y, degree)  # 拟合多项式
    poly = np.poly1d(p)
    y_fit = poly(x)
    return p, y_fit

# 调用多项式拟合
degree = 10  # 可以调整为你需要的拟合次数
p, y_fit = polynomial_fit(x, y, degree)

# 打印拟合的多项式系数
print(f"拟合多项式的系数（从高次到低次）：{p}")

# 准备拟合函数的字符串表达式
equation_str = " + ".join([f"{coef:.2e}x^{degree-i}" for i, coef in enumerate(p)])

# 计算 R²（决定系数）
ss_residual = np.sum((y - y_fit)**2)  # 残差平方和
ss_total = np.sum((y - np.mean(y))**2)  # 总平方和
r_squared = 1 - (ss_residual / ss_total)

# 打印 R² 值
print(f"决定系数：R^2 = {r_squared:.4f}")
```

```
# 绘制数据和拟合曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据")
plt.plot(x, y_fit, linestyle='--', color='r', label=f"{degree}次多项式拟合")

# 在图形中添加拟合函数的表达式
plt.text(0, 0.04, f"拟合函数: y = {equation_str}", transform=plt.gca().transAxes, fontsize=12, verticalalignment='top')

# 在图形中添加 R² 值，使用 R^2 替代 R²
plt.text(0.8, 0.5, f"R^2 = {r_squared:.4f}", transform=plt.gca().transAxes, fontsize=12, verticalalignment='top')

# 图表设置
plt.xlabel("电压 (V)")
plt.ylabel("电流 (A)")
plt.title("电压-电流曲线及拟合")
plt.legend()
plt.grid(True)

# 显示图像
plt.show()

```

函数时代

1 单文件画图+多项式拟合二合一函数 - analyze_data

功能:

1. 自定义参数
   (a) 文件路径
   (b) 拟合次数
   (c) 拟合曲线的线型, 粗细和颜色

```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# 设置中文字体，解决字体显示问题
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 例如使用微软雅黑 (SimHei)
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

def analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
                 line_width=1.5):
    """
    读取数据，进行多项式拟合并绘制结果，同时可以控制图表的细节和拟合的显示内容。

    参数:
    - file_path: Excel 文件的路径。
```

```
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '-'（实线）。
    - line_color: 曲线的颜色，默认为 'b'（蓝色）。
    - line_width: 曲线的线宽，默认为 1.5。
    """

    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)

    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

    # 重新命名列名
    df_filtered.columns = ["电压（V）", "电流（A）"]

    # 转换数据为数值类型（防止字符串干扰）
    df_filtered = df_filtered.astype(float)

    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压（V）"]
    y = df_filtered["电流（A）"]

    # 多项式拟合函数
    def polynomial_fit(x, y, degree=3):
        p = np.polyfit(x, y, degree)  # 拟合多项式
        poly = np.poly1d(p)
        y_fit = poly(x)
        return p, y_fit

    # 调用多项式拟合
    p, y_fit = polynomial_fit(x, y, degree)

    # 打印拟合的多项式系数
    print(f"拟合多项式的系数（从高次到低次）：{p}")

    # 准备拟合函数的字符串表达式
    equation_str = " + ".join([f"{coef:.2e}x^{degree - i}" for i, coef in enumerate(p)])

    # 计算 R²（决定系数）
    ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
    ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
    r_squared = 1 - (ss_residual / ss_total)

    # 打印 R² 值
    print(f"决定系数：R^2 = {r_squared:.12f}")

    # 绘制数据和拟合曲线
    plt.figure(figsize=(8, 6))
    plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据")
    plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟
合", linewidth=line_width)

    # 在图形中添加拟合函数的表达式
```

```
    if show_equation:
        plt.text(0, 0.1, f"拟合函数: y = {equation_str}", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top')

    # 在图形中添加 R² 值，使用 R^2 替代 R²
    if show_r_squared:
        plt.text(0.8, 0.5, f"R^2 = {r_squared:.4f}", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()


# 示例：使用该函数进行分析
file_path = "./datas_learn/B1.xlsx"  # 替换为你自己的文件路径
analyze_data(file_path, degree=5, show_equation=True, show_r_squared=True, line_style='--', line_color='r',
             line_width=1)

```

2 DeepSeek修改

> 必须吐槽两句, gpt-4o打死改不出Latex指数显示, ds一次成功...

增加功能:

1. 原始数据点和连接线可以自定义
2. 修改了表达式的输出方式, 更加好看了

```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# 设置中文字体，解决字体显示问题
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 例如使用微软雅黑 (SimHei)
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

def analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
                 line_width=1.5):
    """
    读取数据，进行多项式拟合并绘制结果，同时可以控制图表的细节和拟合的显示内容。

    参数:
    - file_path: Excel 文件的路径。
```

```
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '-'（实线）。
    - line_color: 曲线的颜色，默认为 'b'（蓝色）。
    - line_width: 曲线的线宽，默认为 1.5。
    """

    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)

    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

    # 重新命名列名
    df_filtered.columns = ["电压 (V)", "电流 (A)"]

    # 转换数据为数值类型（防止字符串干扰）
    df_filtered = df_filtered.astype(float)

    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压 (V)"]
    y = df_filtered["电流 (A)"]

    # 多项式拟合函数
    def polynomial_fit(x, y, degree=3):
        p = np.polyfit(x, y, degree)  # 拟合多项式
        poly = np.poly1d(p)
        y_fit = poly(x)
        return p, y_fit

    # 调用多项式拟合
    p, y_fit = polynomial_fit(x, y, degree)

    # 打印拟合的多项式系数
    print(f"拟合多项式的系数（从高次到低次）：{p}")

    # 准备拟合函数的字符串表达式（优化科学计数法显示）
    equation_terms = []
    for i, coef in enumerate(p):
        power = degree - i
        # 格式化系数为科学计数法
        coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
        # 处理负号和小数点
        if coef < 0:
            coef_str = f"({coef_str})"
        # 添加 x 的幂次
        if power == 0:
            term = coef_str
        else:
            term = f"{coef_str}x^{power}"
        equation_terms.append(term)

    # 拼接拟合函数表达式
    equation_str = " + ".join(equation_terms)
```

```python

    # 计算 R² (决定系数)
    ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
    ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
    r_squared = 1 - (ss_residual / ss_total)

    # 打印 R² 值
    print(f"决定系数: R^2 = {r_squared:.12f}")

    # 绘制数据和拟合曲线
    plt.figure(figsize=(8, 6))

    # 设置原始数据点的样式和连接线的样式
    plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据", markersize=3,
             markerfacecolor='white', markeredgewidth=1, linewidth=2)  # 数据点为白色，线条粗细为2

    # 绘制拟合曲线
    plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟合", linewidth=line_width)

    # 在图形中添加拟合函数的表达式（使用 LaTeX 样式）
    if show_equation:
        plt.text(0.0, 0.06, f"拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))

    # 在图形中添加 R² 值，使用 LaTeX 样式
    if show_r_squared:
        plt.text(0.8, 0.5, f"$R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()


# 示例：使用该函数进行分析
file_path = "./datas_learn/B2-3-21-30.xlsx"  # 替换为你自己的文件路径
analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='--', line_color='r',
             line_width=1)

```

3 迭代拟合函数 - analyze_data_with_outlier_removal

```python
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import IsolationForest

# 设置中文字体，解决字体显示问题
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 例如使用微软雅黑（SimHei）
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题


# # 对应使用例 1
# def analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
#                  line_width=1.5):
#     """
#     读取数据，进行多项式拟合并绘制结果，同时可以控制图表的细节和拟合的显示内容。
#
#     参数:
#     - file_path: Excel 文件的路径。
#     - degree: 多项式拟合的次数，默认3。
#     - show_equation: 是否在图中显示拟合函数表达式，默认显示。
#     - show_r_squared: 是否显示决定系数R²，默认显示。
#     - line_style: 曲线的线型，默认为 '-'（实线）。
#     - line_color: 曲线的颜色，默认为 'b'（蓝色）。
#     - line_width: 曲线的线宽，默认为 1.5。
#     """
#
#     # 读取 Excel 文件
#     xls = pd.ExcelFile(file_path)
#     df = xls.parse(sheet_name=0)
#
#     # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
#     df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）
#
#     # 重新命名列名
#     df_filtered.columns = ["电压（V）", "电流（A）"]
#
#     # 转换数据为数值类型（防止字符串干扰）
#     df_filtered = df_filtered.astype(float)
#
#     # 提取自变量（电压）和因变量（电流）
#     x = df_filtered["电压（V）"]
#     y = df_filtered["电流（A）"]
#
#     # 多项式拟合函数
#     def polynomial_fit(x, y, degree=3):
#         p = np.polyfit(x, y, degree)  # 拟合多项式
#         poly = np.poly1d(p)
#         y_fit = poly(x)
#         return p, y_fit
#
#     # 调用多项式拟合
```

```
#     p, y_fit = polynomial_fit(x, y, degree)
#
#     # 打印拟合的多项式系数
#     print(f"拟合多项式的系数（从高次到低次）：{p}")
#
#     # 准备拟合函数的字符串表达式（优化科学计数法显示）
#     equation_terms = []
#     for i, coef in enumerate(p):
#         power = degree - i
#         # 将python中 "e数字" 的科学计数法格式更改为latex形式
#         coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
#         # 处理负号和小数点
#         if coef < 0:
#             coef_str = f"({coef_str})"
#         # 添加 x 的幂次
#         if power == 0:
#             term = coef_str
#         else:
#             term = f"{coef_str}x^{{{power}}}"
#         equation_terms.append(term)# 使用 {{}} 包裹幂次 防止出现x^10的情况出现（LaTex规范:x^{10}）
#
#
#     # 拼接拟合函数表达式
#     equation_str = " + ".join(equation_terms)
#
#     # 计算 R²（决定系数）
#     ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
#     ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
#     r_squared = 1 - (ss_residual / ss_total)
#
#     # 打印 R² 值
#     print(f"决定系数：R^2 = {r_squared:.12f}") # 12位保证在拟合效果较好时能看出差距
#
#     # 绘制数据和拟合曲线
#     plt.figure(figsize=(8, 6))
#
#     # 设置原始数据点的样式和连接线的样式
#     plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据", markersize=3,
#              markerfacecolor='white', markeredgewidth=1, linewidth=2)  # 数据点为白色，线条粗细为2
#
#     # 绘制拟合曲线
#     plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟合", linewidth=line_width)
#         # 不想传参就改这里即可
#
#     # 当 show_equation == True，在图形中添加拟合函数的表达式（使用 LaTeX 样式）
#     if show_equation:
#         plt.text(0.0, 0.06, f"拟合函数：$y = {equation_str}$", transform=plt.gca().transAxes, fontsize=12,
#                  verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))
#
#     # 当 show_r_squared == True，在图形中添加 R² 值，使用 LaTeX 样式
#     if show_r_squared:
```

```
#         plt.text(0.8, 0.5, f"$R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
#                  verticalalignment='top', ha='left')
#
#     # 图表设置
#     plt.xlabel("电压 (V)")
#     plt.ylabel("电流 (A)")
#     plt.title("电压-电流曲线及拟合")
#     plt.legend()
#     plt.grid(True)
#
#     # 显示图像
#     plt.show()


# 对应使用例 2
def plot_multiple_files(file_paths, colors=None, labels=None, line_styles=None, line_widths=None):
    """
    绘制多个文件的数据到同一张图中，方便对比。

    参数:
    - file_paths: 文件路径列表，例如 ["./file1.xlsx", "./file2.xlsx"]。
    - colors: 每个文件的曲线颜色列表，例如 ['b', 'r', 'g']。
    - labels: 每个文件的图例标签列表，例如 ["文件1", "文件2", "文件3"]。
    - line_styles: 每个文件的线型列表，例如 ['-', '--', ':']。
    - line_widths: 每个文件的线宽列表，例如 [1.5, 1.5, 1.5]。
    """
    # 设置默认值
    if colors is None:
        colors = ['b', 'r', 'g', 'c', 'm', 'y', 'k']  # 默认颜色列表
    if labels is None:
        labels = [f"文件 {i+1}" for i in range(len(file_paths))]  # 默认标签
    if line_styles is None:
        line_styles = ['-'] * len(file_paths)  # 默认线型
    if line_widths is None:
        line_widths = [1.5] * len(file_paths)  # 默认线宽

    # 创建图表
    plt.figure(figsize=(10, 6))

    # 遍历文件路径列表
    for i, file_path in enumerate(file_paths):
        # 读取 Excel 文件
        xls = pd.ExcelFile(file_path)
        df = xls.parse(sheet_name=0)

        # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
        df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列 (0-based 索引)

        # 重新命名列名
        df_filtered.columns = ["电压 (V)", "电流 (A)"]

        # 转换数据为数值类型（防止字符串干扰）
        df_filtered = df_filtered.astype(float)

```

```
        # 提取自变量（电压）和因变量（电流）
        x = df_filtered["电压 (V)"]
        y = df_filtered["电流 (A)"]

        # 绘制数据
        plt.plot(x, y, linestyle=line_styles[i], color=colors[i], label=labels[i], linewidth=line_widths[i])

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线对比")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()

# # 对应使用例 3
# def analyze_data_with_outlier_removal(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-',
#                                       line_color='b', line_width=1.5, remove_outliers=True, threshold=3):
#     """
#     读取数据，进行多项式拟合并绘制结果，支持去除异常值。
#
#     参数:
#     - file_path: Excel 文件的路径。
#     - degree: 多项式拟合的次数，默认3。
#     - show_equation: 是否在图中显示拟合函数表达式，默认显示。
#     - show_r_squared: 是否显示决定系数R²，默认显示。
#     - line_style: 曲线的线型，默认为 '-'（实线）。
#     - line_color: 曲线的颜色，默认为 'b'（蓝色）。
#     - line_width: 曲线的线宽，默认为 1.5。
#     - remove_outliers: 是否去除异常值，默认 True。
#     - threshold: 异常值判断的阈值（基于残差的标准差倍数），默认 3。
#     """
#     # 读取 Excel 文件
#     xls = pd.ExcelFile(file_path)
#     df = xls.parse(sheet_name=0)
#
#     # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
#     df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）
#
#     # 重新命名列名
#     df_filtered.columns = ["电压 (V)", "电流 (A)"]
#
#     # 转换数据为数值类型（防止字符串干扰）
#     df_filtered = df_filtered.astype(float)
#
#     # 提取自变量（电压）和因变量（电流）
#     x = df_filtered["电压 (V)"]
#     y = df_filtered["电流 (A)"]
#
#     # 第一次拟合（用于检测异常值）
#     p, y_fit = np.polyfit(x, y, degree), np.poly1d(np.polyfit(x, y, degree))(x)
```

```
#     residuals = y - y_fit  # 计算残差
#     residual_std = np.std(residuals)  # 残差的标准差
#
#     # 去除异常值
#     if remove_outliers:
#         # 判断异常值：残差的绝对值大于 threshold * 残差的标准差
#         mask = np.abs(residuals) <= threshold * residual_std
#         x_cleaned = x[mask]
#         y_cleaned = y[mask]
#     else:
#         x_cleaned = x
#         y_cleaned = y
#
#     # 第二次拟合（使用去除异常值后的数据）
#     p_cleaned, y_fit_cleaned = np.polyfit(x_cleaned, y_cleaned, degree),
np.poly1d(np.polyfit(x_cleaned, y_cleaned, degree))(x_cleaned)
#
#     # 计算清除异常值前后的 R²
#     ss_residual = np.sum((y - y_fit) ** 2)  # 清除前的残差平方和
#     ss_total = np.sum((y - np.mean(y)) ** 2)  # 清除前的总平方和
#     r_squared = 1 - (ss_residual / ss_total)
#
#     ss_residual_cleaned = np.sum((y_cleaned - y_fit_cleaned) ** 2)  # 清除后的残差平方和
#     ss_total_cleaned = np.sum((y_cleaned - np.mean(y_cleaned)) ** 2)  # 清除后的总平方和
#     r_squared_cleaned = 1 - (ss_residual_cleaned / ss_total_cleaned)
#
#     # 打印清除异常值前后的 R²
#     print(f"清除异常值前的决定系数：R^2 = {r_squared:.12f}")
#     print(f"清除异常值后的决定系数：R^2 = {r_squared_cleaned:.12f}")
#
#     # 准备拟合函数的字符串表达式（优化科学计数法显示）
#     def format_equation(p):
#         equation_terms = []
#         for i, coef in enumerate(p):
#             power = degree - i
#             coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
#             if coef < 0:
#                 coef_str = f"({coef_str})"
#             if power == 0:
#                 term = coef_str
#             else:
#                 term = f"{coef_str}x^{{{power}}}"
#             equation_terms.append(term)
#         return " + ".join(equation_terms)
#
#     equation_str = format_equation(p)
#     equation_str_cleaned = format_equation(p_cleaned)
#
#     # 绘制数据和拟合曲线
#     plt.figure(figsize=(10, 6))
#
#     # 绘制原始数据点
#     plt.plot(x, y, marker='o', linestyle='', color='b', label="原始数据", markersize=3,
#              markerfacecolor='white', markeredgewidth=1)
#
#     # 绘制清除异常值后的数据点
```

```
#     if remove_outliers:
#         plt.plot(x_cleaned, y_cleaned, marker='o', linestyle='', color='g', label="清除异常值后的数据", markersize=3,
#                  markerfacecolor='green', markeredgewidth=1)
#
#     # 绘制清除前的拟合曲线
#     plt.plot(x, y_fit, linestyle='--', color='r', label=f"清除前 {degree}次多项式拟合", linewidth=line_width)
#
#     # 绘制清除后的拟合曲线
#     plt.plot(x_cleaned, y_fit_cleaned, linestyle='-', color='m', label=f"清除后 {degree}次多项式拟合", linewidth=line_width)
#
#     # 添加拟合函数表达式
#     if show_equation:
#         plt.text(0.05, 0.95, f"清除前拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=10,
#                  verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))
#         plt.text(0.05, 0.85, f"清除后拟合函数: $y = {equation_str_cleaned}$", transform=plt.gca().transAxes, fontsize=10,
#                  verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))
#
#     # 添加 R² 值
#     if show_r_squared:
#         plt.text(0.8, 0.5, f"清除前 $R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
#                  verticalalignment='top', ha='left')
#         plt.text(0.8, 0.4, f"清除后 $R^2 = {r_squared_cleaned:.4f}$", transform=plt.gca().transAxes, fontsize=12,
#                  verticalalignment='top', ha='left')
#
#     # 图表设置
#     plt.xlabel("电压 (V)")
#     plt.ylabel("电流 (A)")
#     plt.title("电压-电流曲线及拟合（清除异常值）")
#     plt.legend()
#     plt.grid(True)
#
#     # 显示图像
#     plt.show()


# # 对应使用例 4
# def analyze_data_with_outlier_removal(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-',
#                                       line_color='b', line_width=1.5, remove_outliers=True, threshold=3):
#     """
#     读取数据，进行多项式拟合并绘制结果，支持去除异常值。
#
#     参数:
#     - file_path: Excel 文件的路径。
#     - degree: 多项式拟合的次数，默认3。
#     - show_equation: 是否在图中显示拟合函数表达式，默认显示。
```

```
#     - show_r_squared: 是否显示决定系数R²，默认显示。
#     - line_style: 曲线的线型，默认为 '-'（实线）。
#     - line_color: 曲线的颜色，默认为 'b'（蓝色）。
#     - line_width: 曲线的线宽，默认为 1.5。
#     - remove_outliers: 是否去除异常值，默认 True。
#     - threshold: 异常值判断的阈值（基于残差的标准差倍数），默认 3。
#     """
#     # 读取 Excel 文件
#     xls = pd.ExcelFile(file_path)
#     df = xls.parse(sheet_name=0)
#
#     # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
#     df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）
#
#     # 重新命名列名
#     df_filtered.columns = ["电压（V）", "电流（A）"]
#
#     # 转换数据为数值类型（防止字符串干扰）
#     df_filtered = df_filtered.astype(float)
#
#     # 提取自变量（电压）和因变量（电流）
#     x = df_filtered["电压（V）"]
#     y = df_filtered["电流（A）"]
#
#     # 第一次拟合（用于检测异常值）
#     p, y_fit = np.polyfit(x, y, degree), np.poly1d(np.polyfit(x, y, degree))(x)
#     residuals = y - y_fit  # 计算残差
#     residual_std = np.std(residuals)  # 残差的标准差
#
#     # 去除异常值
#     if remove_outliers:
#         # 方法 1：基于残差的绝对值
#         mask_residual = np.abs(residuals) <= threshold * residual_std
#
#         # 方法 2：基于局部离群点检测（Isolation Forest）
#         clf = IsolationForest(contamination=0.05)  # 假设 5% 的数据是异常值
#         mask_isolation = clf.fit_predict(np.column_stack((x, y))) == 1
#
#         # 结合两种方法
#         mask = mask_residual & mask_isolation
#
#         x_cleaned = x[mask]
#         y_cleaned = y[mask]
#     else:
#         x_cleaned = x
#         y_cleaned = y
#
#     # 第二次拟合（使用去除异常值后的数据）
#     p_cleaned, y_fit_cleaned = np.polyfit(x_cleaned, y_cleaned, degree),
np.poly1d(np.polyfit(x_cleaned, y_cleaned, degree))(x_cleaned)
#
#     # 计算清除异常值前后的 R²
#     ss_residual = np.sum((y - y_fit) ** 2)  # 清除前的残差平方和
#     ss_total = np.sum((y - np.mean(y)) ** 2)  # 清除前的总平方和
#     r_squared = 1 - (ss_residual / ss_total)
#
```

```
#     ss_residual_cleaned = np.sum((y_cleaned - y_fit_cleaned) ** 2)  # 清除后的残差平方和
#     ss_total_cleaned = np.sum((y_cleaned - np.mean(y_cleaned)) ** 2)  # 清除后的总平方和
#     r_squared_cleaned = 1 - (ss_residual_cleaned / ss_total_cleaned)
#
#     # 打印清除异常值前后的 R²
#     print(f"清除异常值前的决定系数: R^2 = {r_squared:.12f}")
#     print(f"清除异常值后的决定系数: R^2 = {r_squared_cleaned:.12f}")
#
#     # 准备拟合函数的字符串表达式（优化科学计数法显示）
#     def format_equation(p):
#         equation_terms = []
#         for i, coef in enumerate(p):
#             power = degree - i
#             coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
#             if coef < 0:
#                 coef_str = f"({coef_str})"
#             if power == 0:
#                 term = coef_str
#             else:
#                 term = f"{coef_str}x^{{{power}}}"
#             equation_terms.append(term)
#         return " + ".join(equation_terms)
#
#     equation_str = format_equation(p)
#     equation_str_cleaned = format_equation(p_cleaned)
#
#     # 绘制数据和拟合曲线
#     plt.figure(figsize=(10, 6))
#
#     # 绘制原始数据点
#     plt.plot(x, y, marker='o', linestyle='', color='b', label="原始数据", markersize=3,
#              markerfacecolor='white', markeredgewidth=1)
#
#     # 绘制清除异常值后的数据点
#     if remove_outliers:
#         plt.plot(x_cleaned, y_cleaned, marker='o', linestyle='', color='g', label="清除异常值后的数据", markersize=3,
#                  markerfacecolor='green', markeredgewidth=1)
#
#     # 绘制清除前的拟合曲线
#     plt.plot(x, y_fit, linestyle='--', color='r', label=f"清除前 {degree}次多项式拟合", linewidth=line_width)
#
#     # 绘制清除后的拟合曲线
#     plt.plot(x_cleaned, y_fit_cleaned, linestyle='-', color='m', label=f"清除后 {degree}次多项式拟合", linewidth=line_width)
#
#     # 添加拟合函数表达式
#     if show_equation:
#         plt.text(0.05, 0.95, f"清除前拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=10,
#                  verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))
#         plt.text(0.05, 0.85, f"清除后拟合函数: $y = {equation_str_cleaned}$", transform=plt.gca().transAxes, fontsize=10,
```

```
#                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))
#
#     # 添加 R² 值
#     if show_r_squared:
#         plt.text(0.8, 0.5, f"清除前 $R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
#                 verticalalignment='top', ha='left')
#         plt.text(0.8, 0.4, f"清除后 $R^2 = {r_squared_cleaned:.4f}$", transform=plt.gca().transAxes, fontsize=12,
#                 verticalalignment='top', ha='left')
#
#     # 图表设置
#     plt.xlabel("电压 (V)")
#     plt.ylabel("电流 (A)")
#     plt.title("电压-电流曲线及拟合（清除异常值）")
#     plt.legend()
#     plt.grid(True)
#
#     # 显示图像
#     plt.show()

# 对应使用例 5
def analyze_data_with_outlier_removal(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-',
                                      line_color='b', line_width=1.5, remove_outliers=True, target_r_squared=0.9,
                                      min_threshold=0.3, initial_threshold=3):
    """
    读取数据，进行多项式拟合并绘制结果，支持迭代去除异常值。

    参数:
    - file_path: Excel 文件的路径。
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '-'（实线）。
    - line_color: 曲线的颜色，默认为 'b'（蓝色）。
    - line_width: 曲线的线宽，默认为 1.5。
    - remove_outliers: 是否去除异常值，默认 True。
    - target_r_squared: 目标决定系数 R²，默认 0.9。
    - min_threshold: threshold 的最小值，默认 0.3。
    - initial_threshold: threshold 的初始值，默认 3。
    """
    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)

    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

    # 重新命名列名
    df_filtered.columns = ["电压 (V)", "电流 (A)"]

    # 转换数据为数值类型（防止字符串干扰）
    df_filtered = df_filtered.astype(float)
```

```
    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压 (V)"]
    y = df_filtered["电流 (A)"]

    # 定义异常值清除和拟合的函数
    def fit_and_remove_outliers(x, y, degree, threshold):
        # 第一次拟合（用于检测异常值）
        p, y_fit = np.polyfit(x, y, degree), np.poly1d(np.polyfit(x, y, degree))(x)
        residuals = y - y_fit  # 计算残差
        residual_std = np.std(residuals)  # 残差的标准差

        # 方法 1：基于残差的绝对值
        mask_residual = np.abs(residuals) <= threshold * residual_std

        # 方法 2：基于局部离群点检测（Isolation Forest）
        clf = IsolationForest(contamination=0.05)  # 假设 5% 的数据是异常值
        mask_isolation = clf.fit_predict(np.column_stack((x, y))) == 1

        # 结合两种方法
        mask = mask_residual & mask_isolation

        x_cleaned = x[mask]
        y_cleaned = y[mask]

        # 第二次拟合（使用去除异常值后的数据）
        p_cleaned, y_fit_cleaned = np.polyfit(x_cleaned, y_cleaned, degree),
np.poly1d(np.polyfit(x_cleaned, y_cleaned, degree))(x_cleaned)

        # 计算 R²
        ss_residual_cleaned = np.sum((y_cleaned - y_fit_cleaned) ** 2)  # 清除后的残差平方
和
        ss_total_cleaned = np.sum((y_cleaned - np.mean(y_cleaned)) ** 2)  # 清除后的总平方
和
        r_squared_cleaned = 1 - (ss_residual_cleaned / ss_total_cleaned)

        return x_cleaned, y_cleaned, p_cleaned, y_fit_cleaned, r_squared_cleaned

    # 初始化变量
    threshold = initial_threshold
    x_cleaned, y_cleaned = x, y
    r_squared_cleaned = 0

    # 迭代清除异常值
    if remove_outliers:
        while threshold >= min_threshold:
            x_cleaned, y_cleaned, p_cleaned, y_fit_cleaned, r_squared_cleaned =
fit_and_remove_outliers(x_cleaned, y_cleaned, degree, threshold)
            print(f"当前 threshold: {threshold:.2f}, R²: {r_squared_cleaned:.4f}")

            # 如果达到目标 R²，停止迭代
            if r_squared_cleaned >= target_r_squared:
                print(f"达到目标 R²: {target_r_squared}")
                break

            # 降低 threshold
```

```
            threshold -= 0.1
        else:
            print(f"未达到目标 R², 当前 R²: {r_squared_cleaned:.4f}, threshold 已降至最小值 {min_threshold}, 该方法不可行。")

    # 如果没有清除异常值，直接拟合
    else:
        p_cleaned, y_fit_cleaned = np.polyfit(x_cleaned, y_cleaned, degree), np.poly1d(np.polyfit(x_cleaned, y_cleaned, degree))(x_cleaned)
        ss_residual_cleaned = np.sum((y_cleaned - y_fit_cleaned) ** 2)  # 清除后的残差平方和
        ss_total_cleaned = np.sum((y_cleaned - np.mean(y_cleaned)) ** 2)  # 清除后的总平方和
        r_squared_cleaned = 1 - (ss_residual_cleaned / ss_total_cleaned)

    # 准备拟合函数的字符串表达式（优化科学计数法显示）
    def format_equation(p):
        equation_terms = []
        for i, coef in enumerate(p):
            power = degree - i
            coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
            if coef < 0:
                coef_str = f"({coef_str})"
            if power == 0:
                term = coef_str
            else:
                term = f"{coef_str}x^{{{power}}}"
            equation_terms.append(term)
        return " + ".join(equation_terms)

    equation_str_cleaned = format_equation(p_cleaned)

    # 绘制数据和拟合曲线
    plt.figure(figsize=(10, 6))

    # 绘制原始数据点
    plt.plot(x, y, marker='o', linestyle='', color='b', label="原始数据", markersize=3,
             markerfacecolor='white', markeredgewidth=1)

    # 绘制清除异常值后的数据点
    if remove_outliers:
        plt.plot(x_cleaned, y_cleaned, marker='o', linestyle='', color='g', label="清除异常值后的数据", markersize=3,
                 markerfacecolor='green', markeredgewidth=1)

    # 绘制清除后的拟合曲线
    plt.plot(x_cleaned, y_fit_cleaned, linestyle='-', color='m', label=f"清除后 {degree}次多项式拟合", linewidth=line_width)

    # 添加拟合函数表达式
    if show_equation:
        plt.text(0.05, 0.95, f"清除后拟合函数: $y = {equation_str_cleaned}$", transform=plt.gca().transAxes, fontsize=10,
                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))

```

```
    # 添加 R² 值
    if show_r_squared:
        plt.text(0.8, 0.5, f"清除后 $R^2 = {r_squared_cleaned:.4f}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合（清除异常值）")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()


# # 使用例 1: 使用 analyze_data 函数进行单文件作图和曲线拟合
# # file_path = "./datas_learn/B00.xlsx"  # 替换为你自己的文件路径(一定要是.xlsx文件!)
# file_path = "./datas_learn/test/B1.xlsx"  # 替换为你自己的文件路径(一定要是.xlsx文件!)
#
# # 分析file_path对应路径的文件，用3次多项式进行拟合，拟合多项式表达式和决定系数均展示，拟合曲线的线型为虚线，红色，宽度为1
# analyze_data(file_path, degree=3, show_equation=False, show_r_squared=True, line_style='--', line_color='r',
#              line_width=1)



# # 使用例 2: 使用 plot_multiple_files 进行多文件对比
# file_paths = [
#     "./datas_learn/compare1/B0.xlsx",
#     "./datas_learn/compare1/B0.5.xlsx",
#     # "./datas_learn/compare1/B1.xlsx",
#
#     # "./datas_learn/B0.xlsx",
#     # "./datas_learn/B0.5.xlsx",
#     # "./datas_learn/B1.xlsx",
# ]
#
# # colors = ['b', 'r', 'g']  # 每个文件的曲线颜色
# # labels = ["B0", "B0.5", "B1"]  # 每个文件的图例标签
# # line_styles = ['-', '--', ':']  # 每个文件的线型
# # line_widths = [1.5, 1.5, 1.5]  # 每个文件的线宽
#
# colors = ['b', 'r']  # 每个文件的曲线颜色
# labels = ["B0", "B0.5"]  # 每个文件的图例标签
# line_styles = ['-', '--']  # 每个文件的线型
# line_widths = [1.5, 1.5]  # 每个文件的线宽
#
# plot_multiple_files(file_paths, colors=colors, labels=labels, line_styles=line_styles, line_widths=line_widths)


# # 使用例 3: 使用 analyze_data_with_outlier_removal 进行分析
```

```
# file_path = "./datas_learn/test2/T1-5-12-21(slightly boom,900V one time).xlsx"  # 替换为你自己的文件路径
# analyze_data_with_outlier_removal(file_path, degree=3, show_equation=True, show_r_squared=True,
#                                   line_style='--', line_color='r', line_width=1, remove_outliers=True, threshold=1)

# # 使用例 4: 使用 analyze_data_with_outlier_removal 进行分析
# file_path = "./datas_learn/B00.xlsx"  # 替换为你自己的文件路径
# analyze_data_with_outlier_removal(file_path, degree=3, show_equation=True, show_r_squared=True,
#                                   line_style='--', line_color='r', line_width=1, remove_outliers=True, threshold=0.5)

# 使用例 5: 使用 analyze_data_with_outlier_removal 进行分析
file_path = "./datas_learn/B00.xlsx"  # 替换为你自己的文件路径
analyze_data_with_outlier_removal(file_path, degree=3, show_equation=False, show_r_squared=True,
                                  line_style='--', line_color='r', line_width=1, remove_outliers=True,
                                  target_r_squared=0.9, min_threshold=1, initial_threshold=3)

```

4 增加多文件画图函数 - plot_multiple_files

```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# 设置中文字体，解决字体显示问题
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 例如使用微软雅黑 (SimHei)
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

def analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
                 line_width=1.5):
    """
    读取数据，进行多项式拟合并绘制结果，同时可以控制图表的细节和拟合的显示内容。

    参数:
    - file_path: Excel 文件的路径。
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '-' (实线) 。
    - line_color: 曲线的颜色，默认为 'b' (蓝色) 。
    - line_width: 曲线的线宽，默认为 1.5。
    """

    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)
```

```python

    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列 (0-based 索引)

    # 重新命名列名
    df_filtered.columns = ["电压 (V)", "电流 (A)"]

    # 转换数据为数值类型（防止字符串干扰）
    df_filtered = df_filtered.astype(float)

    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压 (V)"]
    y = df_filtered["电流 (A)"]

    # 多项式拟合函数
    def polynomial_fit(x, y, degree=3):
        p = np.polyfit(x, y, degree)  # 拟合多项式
        poly = np.poly1d(p)
        y_fit = poly(x)
        return p, y_fit

    # 调用多项式拟合
    p, y_fit = polynomial_fit(x, y, degree)

    # 打印拟合的多项式系数
    print(f"拟合多项式的系数（从高次到低次）：{p}")

    # 准备拟合函数的字符串表达式（优化科学计数法显示）
    equation_terms = []
    for i, coef in enumerate(p):
        power = degree - i
        # 将python中 "e数字" 的科学计数法格式更改为latex形式
        coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
        # 处理负号和小数点
        if coef < 0:
            coef_str = f"({coef_str})"
        # 添加 x 的幂次
        if power == 0:
            term = coef_str
        else:
            term = f"{coef_str}x^{{{power}}}"
        equation_terms.append(term)# 使用 {{}} 包裹幂次 防止出现x^10的情况出现 (LaTex规范:x^{10})


    # 拼接拟合函数表达式
    equation_str = " + ".join(equation_terms)

    # 计算 R² (决定系数)
    ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
    ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
    r_squared = 1 - (ss_residual / ss_total)

    # 打印 R² 值
    print(f"决定系数: R^2 = {r_squared:.12f}") # 12位保证在拟合效果较好时能看出差距

```

```
    # 绘制数据和拟合曲线
    plt.figure(figsize=(8, 6))

    # 设置原始数据点的样式和连接线的样式
    plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据", markersize=3,
             markerfacecolor='white', markeredgewidth=1, linewidth=2)  # 数据点为白色，线条粗细为2

    # 绘制拟合曲线
    plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟合", linewidth=line_width)
        # 不想传参就改这里即可

    # 当 show_equation == True，在图形中添加拟合函数的表达式（使用 LaTeX 样式）
    if show_equation:
        plt.text(0.0, 0.06, f"拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))

    # 当 show_r_squared == True，在图形中添加 R² 值，使用 LaTeX 样式
    if show_r_squared:
        plt.text(0.8, 0.5, f"$R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()


def plot_multiple_files(file_paths, colors=None, labels=None, line_styles=None, line_widths=None):
    """
    绘制多个文件的数据到同一张图中，方便对比。

    参数:
    - file_paths: 文件路径列表，例如 ["./file1.xlsx", "./file2.xlsx"]。
    - colors: 每个文件的曲线颜色列表，例如 ['b', 'r', 'g']。
    - labels: 每个文件的图例标签列表，例如 ["文件1", "文件2", "文件3"]。
    - line_styles: 每个文件的线型列表，例如 ['-', '--', ':']。
    - line_widths: 每个文件的线宽列表，例如 [1.5, 1.5, 1.5]。
    """
    # 设置默认值
    if colors is None:
        colors = ['b', 'r', 'g', 'c', 'm', 'y', 'k']  # 默认颜色列表
    if labels is None:
        labels = [f"文件 {i+1}" for i in range(len(file_paths))]  # 默认标签
    if line_styles is None:
        line_styles = ['-'] * len(file_paths)  # 默认线型
```

```
    if line_widths is None:
        line_widths = [1.5] * len(file_paths)  # 默认线宽

    # 创建图表
    plt.figure(figsize=(10, 6))

    # 遍历文件路径列表
    for i, file_path in enumerate(file_paths):
        # 读取 Excel 文件
        xls = pd.ExcelFile(file_path)
        df = xls.parse(sheet_name=0)

        # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
        df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

        # 重新命名列名
        df_filtered.columns = ["电压 (V)", "电流 (A)"]

        # 转换数据为数值类型（防止字符串干扰）
        df_filtered = df_filtered.astype(float)

        # 提取自变量（电压）和因变量（电流）
        x = df_filtered["电压 (V)"]
        y = df_filtered["电流 (A)"]

        # 绘制数据
        plt.plot(x, y, linestyle=line_styles[i], color=colors[i], label=labels[i],
linewidth=line_widths[i])

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线对比")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()


# 使用例 1: 使用 analyze_data 函数进行单文件作图和曲线拟合
# file_path = "./datas_learn/B00.xlsx"  # 替换为你自己的文件路径(一定要是.xlsx文件!)
file_path = "./datas_learn/compare1/B1.xlsx"  # 替换为你自己的文件路径(一定要是.xlsx文件!)

# 使用例 1: 分析file_path对应路径的文件，用3次多项式进行拟合，拟合多项式表达式和决定系数均展
示，拟合曲线的线型为虚线，红色，宽度为1
analyze_data(file_path, degree=3, show_equation=False, show_r_squared=True, line_style='-
-', line_color='r',
             line_width=1)

# 使用例 2: 使用 plot_multiple_files 进行多文件对比
file_paths = [
    "./datas_learn/compare1/B0.xlsx",
    "./datas_learn/compare1/B0.5.xlsx",
    # "./datas_learn/compare1/B1.xlsx",

```

```
    # "./datas_learn/B0.xlsx",
    # "./datas_learn/B0.5.xlsx",
    # "./datas_learn/B1.xlsx",
]

# colors = ['b', 'r', 'g']  # 每个文件的曲线颜色
# labels = ["B0", "B0.5", "B1"]  # 每个文件的图例标签
# line_styles = ['-', '--', ':']  # 每个文件的线型
# line_widths = [1.5, 1.5, 1.5]  # 每个文件的线宽

colors = ['b', 'r']  # 每个文件的曲线颜色
labels = ["B0", "B0.5"]  # 每个文件的图例标签
line_styles = ['-', '--']  # 每个文件的线型
line_widths = [1.5, 1.5]  # 每个文件的线宽

plot_multiple_files(file_paths, colors=colors, labels=labels, line_styles=line_styles, line_widths=line_widths)

```

批量处理时代

1 传文件夹, 批量画所有表格的图.(batch_analyze_data.py)

```python
import os
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt


# 设置中文字体，解决字体显示问题
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 例如使用微软雅黑 (SimHei)
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

def analyze_data(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='--', line_color='r',
                 line_width=1.5, connect_points=True):
    """
    读取数据，进行多项式拟合并绘制结果，同时可以控制图表的细节和拟合的显示内容。

    参数:
    - file_path: Excel 文件的路径。
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '--'（虚线）。
    - line_color: 曲线的颜色，默认为 'r'（红色）。
    - line_width: 曲线的线宽，默认为 1.5。
    - connect_points: 是否连接原始数据点，默认连接。
    """

    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)

```

```python
    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列（0-based 索引）

    # 重新命名列名
    df_filtered.columns = ["电压 (V)", "电流 (A)"]

    # 转换数据为数值类型（防止字符串干扰）
    df_filtered = df_filtered.astype(float)

    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压 (V)"]
    y = df_filtered["电流 (A)"]

    # 多项式拟合函数
    def polynomial_fit(x, y, degree=3):
        p = np.polyfit(x, y, degree)  # 拟合多项式
        poly = np.poly1d(p)
        y_fit = poly(x)
        return p, y_fit

    # 调用多项式拟合
    p, y_fit = polynomial_fit(x, y, degree)

    # 打印拟合的多项式系数
    print(f"拟合多项式的系数（从高次到低次）：{p}")

    # 准备拟合函数的字符串表达式（优化科学计数法显示）
    equation_terms = []
    for i, coef in enumerate(p):
        power = degree - i
        # 将python中 "e数字" 的科学计数法格式更改为latex形式
        coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
        # 处理负号和小数点
        if coef < 0:
            coef_str = f"({coef_str})"
        # 添加 x 的幂次
        if power == 0:
            term = coef_str
        else:
            term = f"{coef_str}x^{{{power}}}"
        equation_terms.append(term)# 使用 {{}} 包裹幂次 防止出现x^10的情况出现 (LaTex规范:x^{10})

    # 拼接拟合函数表达式
    equation_str = " + ".join(equation_terms)

    # 计算 R² (决定系数)
    ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
    ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
    r_squared = 1 - (ss_residual / ss_total)

    # 打印 R² 值
    print(f"决定系数: R^2 = {r_squared:.12f}") # 12位保证在拟合效果较好时能看出差距

    # 绘制数据和拟合曲线
    plt.figure(figsize=(8, 6))
```

```
    # 设置原始数据点的样式和连接线的样式
    if connect_points:
        plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据", markersize=3,
                 markerfacecolor='white', markeredgewidth=1, linewidth=2)  # 数据点为白色，线条粗细为2
    else:
        plt.plot(x, y, marker='o', linestyle='', color='b', label="原始数据", markersize=3,
                 markerfacecolor='white', markeredgewidth=1)  # 数据点为白色，不连接

    # 绘制拟合曲线
    plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟合", linewidth=line_width)

    # 当 show_equation == True，在图形中添加拟合函数的表达式（使用 LaTeX 样式）
    if show_equation:
        plt.text(0.0, 0.06, f"拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))

    # 当 show_r_squared == True，在图形中添加 R² 值，使用 LaTeX 样式
    if show_r_squared:
        plt.text(0.8, 0.5, f"$R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合")
    plt.legend()
    plt.grid(True)

    # 显示图像
    plt.show()

def analyze_data_no_display(file_path, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
                            line_width=1.5, connect_points=True):
    """
    与 analyze_data 功能相同，但不显示图像。
    """
    # 读取 Excel 文件
    xls = pd.ExcelFile(file_path)
    df = xls.parse(sheet_name=0)

    # 提取 C35 到 D536 的数据（假设列名在第 35 行开始）
    df_filtered = df.iloc[34:536, [2, 3]].dropna()  # 选择 C 和 D 列 (0-based 索引)

    # 重新命名列名
    df_filtered.columns = ["电压 (V)", "电流 (A)"]

    # 转换数据为数值类型（防止字符串干扰）
```

```
    df_filtered = df_filtered.astype(float)

    # 提取自变量（电压）和因变量（电流）
    x = df_filtered["电压 (V)"]
    y = df_filtered["电流 (A)"]

    # 多项式拟合函数
    def polynomial_fit(x, y, degree=3):
        p = np.polyfit(x, y, degree)  # 拟合多项式
        poly = np.poly1d(p)
        y_fit = poly(x)
        return p, y_fit

    # 调用多项式拟合
    p, y_fit = polynomial_fit(x, y, degree)

    # 准备拟合函数的字符串表达式（优化科学计数法显示）
    equation_terms = []
    for i, coef in enumerate(p):
        power = degree - i
        # 将python中 "e数字" 的科学计数法格式更改为latex形式
        coef_str = f"{coef:.2e}".replace("e", "\\cdot10^{") + "}"
        # 处理负号和小数点
        if coef < 0:
            coef_str = f"({coef_str})"
        # 添加 x 的幂次
        if power == 0:
            term = coef_str
        else:
            term = f"{coef_str}x^{{{power}}}"
        equation_terms.append(term)# 使用 {{}} 包裹幂次 防止出现x^10的情况出现 (LaTex规范:x^{10})

    # 拼接拟合函数表达式
    equation_str = " + ".join(equation_terms)

    # 计算 R² (决定系数)
    ss_residual = np.sum((y - y_fit) ** 2)  # 残差平方和
    ss_total = np.sum((y - np.mean(y)) ** 2)  # 总平方和
    r_squared = 1 - (ss_residual / ss_total)

    # 绘制数据和拟合曲线
    plt.figure(figsize=(8, 6))

    # 设置原始数据点的样式和连接线的样式
    if connect_points:
        plt.plot(x, y, marker='o', linestyle='-', color='b', label="原始数据", markersize=3,
                 markerfacecolor='white', markeredgewidth=1, linewidth=2)  # 数据点为白色，线条粗细为2
    else:
        plt.plot(x, y, marker='o', linestyle='', color='b', label="原始数据", markersize=3,
                 markerfacecolor='white', markeredgewidth=1)  # 数据点为白色，不连接

    # 绘制拟合曲线
```

```
    plt.plot(x, y_fit, linestyle=line_style, color=line_color, label=f"{degree}次多项式拟合", linewidth=line_width)

    # 当 show_equation == True，在图形中添加拟合函数的表达式（使用 LaTeX 样式）
    if show_equation:
        plt.text(0.0, 0.06, f"拟合函数: $y = {equation_str}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left', bbox=dict(facecolor='white', alpha=0.8))

    # 当 show_r_squared == True，在图形中添加 R² 值，使用 LaTeX 样式
    if show_r_squared:
        plt.text(0.8, 0.5, f"$R^2 = {r_squared:.4f}$", transform=plt.gca().transAxes, fontsize=12,
                 verticalalignment='top', ha='left')

    # 图表设置
    plt.xlabel("电压 (V)")
    plt.ylabel("电流 (A)")
    plt.title("电压-电流曲线及拟合")
    plt.legend()
    plt.grid(True)

    # 返回图像对象
    return plt.gcf()

def batch_analyze_data(folder_path, output_folder, degree=3, show_equation=True, show_r_squared=True, line_style='-', line_color='b',
                       line_width=1.5, connect_points=False):
    """
    批量处理文件夹中的 Excel 文件，进行多项式拟合并保存图像。

    参数:
    - folder_path: 包含 Excel 文件的文件夹路径。
    - output_folder: 保存图像的文件夹路径。
    - degree: 多项式拟合的次数，默认3。
    - show_equation: 是否在图中显示拟合函数表达式，默认显示。
    - show_r_squared: 是否显示决定系数R²，默认显示。
    - line_style: 曲线的线型，默认为 '-'（实线）。
    - line_color: 曲线的颜色，默认为 'b'（蓝色）。
    - line_width: 曲线的线宽，默认为 1.5。
    - connect_points: 是否连接原始数据点，默认不连接。
    """
    # 创建输出文件夹
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # 遍历文件夹中的所有 Excel 文件
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.xlsx') or file_name.endswith('.xls'):
            file_path = os.path.join(folder_path, file_name)
            # 调用 analyze_data_no_display 函数生成图像
            fig = analyze_data_no_display(file_path, degree, show_equation, show_r_squared, line_style, line_color,
                                          line_width, connect_points)
            # 保存图像
```

```
            output_file_path = os.path.join(output_folder, f"{os.path.splitext(file_name)[0]}.png")
            fig.savefig(output_file_path, dpi=240)
            plt.close(fig)
            print(f"已保存图像: {output_file_path}")
    print('分析完成.')

# 示例调用
# batch_analyze_data('./datas', './datas/datas_img')

batch_analyze_data('./datas_learn/old_34', './datas_learn/old_34/old34_img', line_style='--', line_color='r', connect_points=False)


```