3.3.3. Опыт Милликена. Рассчеты с использованием Python

Пазов Тенгиз

16.09.2024

1 Код

Данный код рассчитывает по формуле для заряда сам заряд для каждой пары t и t', то есть для падения и подъема одной и той же частицы на протяжении пяти повторений. Посчитанные данные записываются в один массив зарядов. Далее мы смотрим на кучность измеренных зарядов, и если в окрестности точки кучность оказывается большой, то ищем среднее значение зарядов из данной кучи. Таких средних значений, как видно из предыдущего файла, получилось 5. Также, вначале было рассчитано $U_{min} = 217.123$, данный рассчет также прописан в коде.

```
import math
import matplotlib.pyplot as plt
from functools import reduce
# Определим данные установки
eyepiece_division_price = 0.25 * (10 ** -2)
oil_density = 898  # кг/м^3
U = 200  # напряжение конденсатора в вольтах
l = 0.725 * (10 ** -2)  # расстояние между пластинами

# Определим минимальное напряжение:
h_min = 0.001  # м
t_min = 20  # с
q_min = 5 * 1.6 * (10 ** -19)  # Кл
#формула так записана для удобства, при запуске интерпритатора необходимо вернуть
#все слагаемые в одну строчку
U_min = 9 * 3.14 * math.sqrt(((((2 * 1.85 * (10 ** -5) * h_min) / (t_min)) ** 3)) /
(9.81 * oil_density)) * (l / q_min)

# Время падения и подъема частиц
t_values = [
    [15.62, 18.815, 21.91, 22.08, 28.68],
    [30.65, 31.10, 28.77, 46.49, 43.52],
    [30.85, 30.02, 27.78, 46.15, 35.67],
    [31.08, 32.22, 31.68, 28.46, 33.34],
    [18.51, 18.59, 32.47, 36.32, 31.33],
]
t_up_values = [
    [13.33, 10.61, 12.41, 19.73, 21.21],
    [7.46, 8.91, 9.23, 7.87, 8.51],
    [8.59, 5.21, 7.67, 7.59, 7.87],
    [8.97, 8.07, 7.19, 7.66, 10.37],
    [6.71, 6.54, 7.44, 11.06, 9.98],
]
h = [(1/4) * (10 ** -3)] * 5
q_values = []

def calculate_q(t, t_up):
    q = []
```

```
    for i in range(len(t)):
        element = (9 * math.pi * (l / U) * math.sqrt(2 / (oil_density * 9.81)) *
                   ((1.85 * (10 ** -5)) * h[i]) ** (3/2) *
                   ((t[i] + t_up[i]) / ((t[i]) ** (3/2) * t_up[i])))
        q.append(element)
    return q

for i in range(5):
    q_values.extend(calculate_q(t_values[i], t_up_values[i]))

print("Q значения:", q_values)

def calculate_group_averages(data, group_size):
    averages = []
    for i in range(0, len(data), group_size):
        group = data[i:i + group_size]
        if group:
            averages.append(sum(group) / len(group))
    return averages

group_size = 5
average_q_values = calculate_group_averages(q_values, group_size)
print("Средние значения по группам:", average_q_values)
print("Минимальное напряжение для подъема капель", U_min)
# Найдем НОД списка average_q_values
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def find_gcd_of_list(numbers):
    return reduce(gcd, numbers)

gcd_of_averages = find_gcd_of_list(average_q_values)
print("НОД списка average_q_values:", gcd_of_averages)
# Определим массив погрешностей для каждого заряда
delta_q = []
# Погрешность U, взяв ее за 5% от исходной величины
delta_U = U * 0.05
# Погрешность измерения времени
delta_t = 0.2

# Отдельно посчитаем погрешности измерений для каждого заряда
for i in range(len(q_values)):
    idx = i // len(t_up_values[0])
    jdx = i % len(t_up_values[0])

    element_1 = q_values[i] * math.sqrt((delta_U / U) ** 2 +
```

```
    (5 * (delta_t / (t_up_values[idx][jdx] + t_values[idx][jdx]))) ** 2)

    delta_q.append(element_1)

print("Список погрешностей для каждой частицы:", delta_q)

# Теперь также рассчитаем погрешности для средних значений в каждой куче.
def calculate_average_error(delta_q_list, group_size):
    average_errors = []
    for i in range(0, len(delta_q_list), group_size):
        group_errors = delta_q_list[i:i + group_size]
        if group_errors:
            total_error = math.sqrt(sum(error ** 2 for error in group_errors))
            average_errors.append(total_error)
    return average_errors

average_error_q_values = calculate_average_error(delta_q, group_size)
print("Погрешности средних значений по группам:", average_error_q_values)

# Найдем установившуюся скорость для каждого значения заряда:
v_ust = []
for i in range(len(t_values[0])):
    v_ust.append(h[0] / t_values[0][i])

# Определим радиус частиц и другие параметры
r = []
k = []
t_ust = []
v_sr = []

for i in range(len(t_values[0])):
    element_3 = math.sqrt((9 * 1.85 * (10 ** -5) * h[0]) /
    (2 * oil_density * 9.8 * t_values[0][i]))
    r.append(element_3)

for i in range(len(r)):
    element_4 = 6 * math.pi * 1.85 * (10 ** -5) * r[i]
    k.append(element_4)

for i in range(len(v_ust)):
    element_6 = v_ust[i] / 9.8
    t_ust.append(element_6)
print("радиусы", r)
print("Время релаксации:", t_ust)
# Теперь найдем среднее значение от v
for i in range(len(t_ust)):
    element_5 = 1 - math.exp(-(k[i] * t_ust[i] / ((4/3) *
    math.pi * oil_density * (r[i] **3))))
```

```
    v_sr.append(element_5)

v_chastnoe = []
for i in range(len(v_ust)):
    element_7 = v_ust[i] - v_sr[i]
    v_chastnoe.append(element_7)
print("Установившаяся скорость для каждой частицы:", v_ust)
print("Средняя скорость за время релаксации:", v_sr)
print("Разность скоростей:", v_chastnoe)
print("Погрешности для каждого среднего заряда из кучи:", average_error_q_values)
#время релаксации мы знаем, это массив t_ust
#теперь найдем расстояние s(t_ust), на которое сместится частица за это время:
#s = v_ust[i]^2 / g
s = []
for i in range(len(v_ust)):
    element_8 = (v_ust[i])**(2) / 9.8
    s.append(element_8)
y = [0] * len(q_values)
print("Время релаксации:", t_ust)
print("Расстояние пройденное за это время:", s)
#посчитаем погрешность установившейся скорости
sigma_h = 0.00001
sigma_v = []
for i in range(len(v_ust)):
    element_9 = v_ust[i]*math.sqrt((sigma_h / h[0])**2 +
    (delta_t / t_values[0][i])**2)
    sigma_v.append(element_9)
print("Погрешности установившейся скорости:", sigma_v)
#посчитаем погрешность S:
sigma_s = []
for i in range(len(sigma_v)):
    element_10 = 2 * s[i] * (sigma_v[i] / v_ust[i])
    sigma_s.append(element_10)
#посчитаем погрешность измерения времени релаксации
sigma_t = []
for i in range(len(t_ust)):
    element_11 = t_ust[i] * sigma_v[i] / v_ust[i]
    sigma_t.append(element_11)
print("Погрешности измерения пути:", sigma_s)
print("ПОгрешности времени релаксации:", sigma_t)
#посчитаем погрешности радиусов частиц
sigma_r = []
for i in range(5):
    element_12 = 0.5 * r[i] * math.sqrt((sigma_h / h[0])**2 + (delta_t / t_values[0][i])
    sigma_r.append(element_12)
print("Погрешности радиусов частиц:", sigma_r)
#теперь зная погрешности радиусов, посчитаем погрешности средних скоростей:
sigma_v_sr = []
```

```
for i in range(len(t_ust)):
    element_13 = math.sqrt(20.25 * ((1.85 * (10**(-5)))**2 / r[i]**4 * oil_density**2) *
    t_ust[i] / r[i]**2 * oil_density) *
    sigma_t[i]**2 + 81 * ( (1.85 * 10**(-5) * t_ust[i])**2 / oil_density**2 * r[i]**6) *
    math.exp(-9 * 1.85 * 10**(-5) * t_ust[i] / r[i]**2 * oil_density) * sigma_r[i]**2)
    sigma_v_sr.append(element_13)
plt.plot(q_values, y, 'o')
plt.xlabel('Заряд (q)')
plt.yticks([])
plt.savefig('my_graph.png')
```

Данный код высчитывает как установившуюся скорость, так и разность между усредненной за время релаксации. А также расчитывает минимальное напряжение, время релаксации для каждой частиы, погрешность измерения каждой величины заряда, а также располагает значения по кучности на горизонтальной оси и далее проходя по всей оси выбирает наиболее скученные места и ищет средние значения в данных кучах. Таким образом и был определен элементарный заряд(взяв НОД из всех средних в кучах).