

本数据处理中  $N$  均为计数率  $N/t$ ，与原始数据记录  $N$  不同!!!

## 计算 $\alpha$ 粒子束的强度与能量 $E$

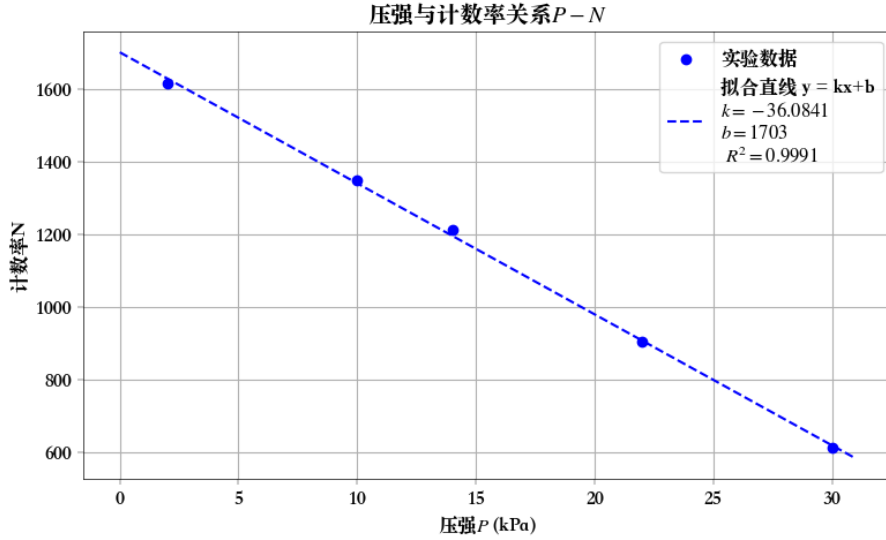


图 1:  $P - N$  曲线.

其中拟合曲线:

$$N = -36.0841P + 1703. \quad (1)$$

线性相关程度  $R^2 = 0.9991$ . 由此可知  $N_0 = 1703$ .

### 1.1 $\alpha$ 粒子射程计算

当计数  $N = \frac{N_0}{2} = 851$  时的粒子运动距离即为射程:

$$R' = l_2 = 7.25 \text{ cm}.$$

对应的压强可以通过公式 1 算出, 结果为:

$$P' = 23.583 \text{ kPa}.$$

同截面情况下我们有  $\frac{R_1}{R_2} = \frac{\rho_2}{\rho_1}$ ，且考虑空气密度公式：

$$\rho = 1.293 \times \frac{P}{1.325} \times \frac{273}{T}.$$

可以算出温度相同时  $\frac{R_1}{R_2} = \frac{P_2}{P_1}$ ，因此我们将  $P' = 23.583 \text{ kPa}$  下的射程换算到  $P = 101.235 \text{ kPa}$  下得到：

$$R = 1.69 \text{ cm}.$$

## 1.2 $\alpha$ 粒子能量计算

考虑公式：

$$R = (0.285 + 0.005E)E^{1.5}.$$

为超越方程（其中  $R$  单位为  $\text{cm}$ ， $E$  单位为  $\text{MeV}$ ）。我们可以考虑利用 Newton 迭代法或弦截法 (Secant Method) 来求解非线性超越方程的根：

$$f(x) = (0.285 + 0.005x)x^{1.5} - 1.69.$$

代码及结果见附录，最终计算结果为：

$$E = 3.16 \text{ MeV}.$$

## 验证 $N \propto \sin^{-4} \frac{\theta}{2}$ 关系

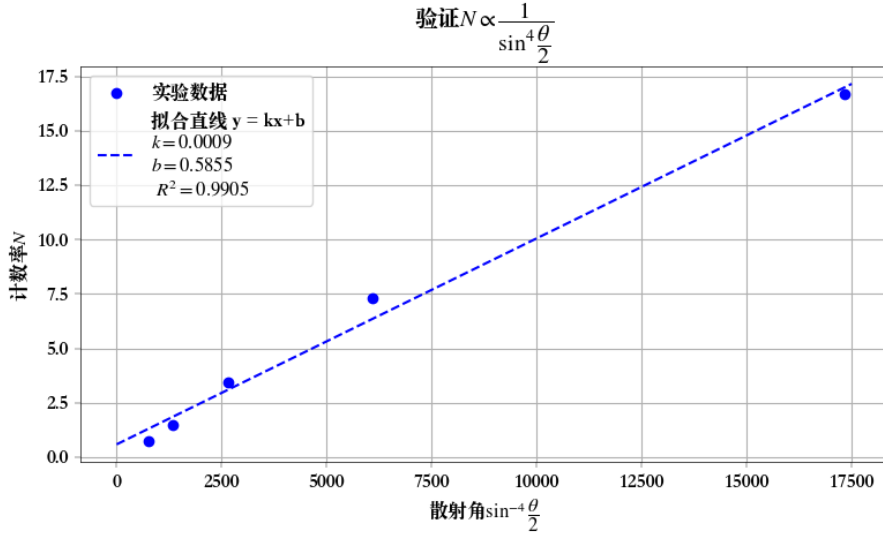


图 2:  $N - \sin^{-4} \frac{\theta}{2}$  曲线.

其中拟合曲线:

$$N = 0.0009 \sin^{-4} \frac{\theta}{2} + 0.5855. \quad (2)$$

线性相关程度  $R^2 = 0.9905$ .

计算  $K = N \sin^4 \frac{\theta}{2}$  理论值:

$$\begin{aligned}
 K_{\text{theoretical}} &= 4.8065 \times 10^{-34} \frac{N_0}{E^2 l_1^2} \\
 &= 4.8065 \times 10^{-34} \frac{1703}{(3.16 \times 10^6 \times 1.6 \times 10^{-19})^2 \times 0.0417^2} \\
 &= 0.0018
 \end{aligned}$$

其中 120 是测量  $N_0$  时所用时间.

## $K$ 值的实验与理论对比

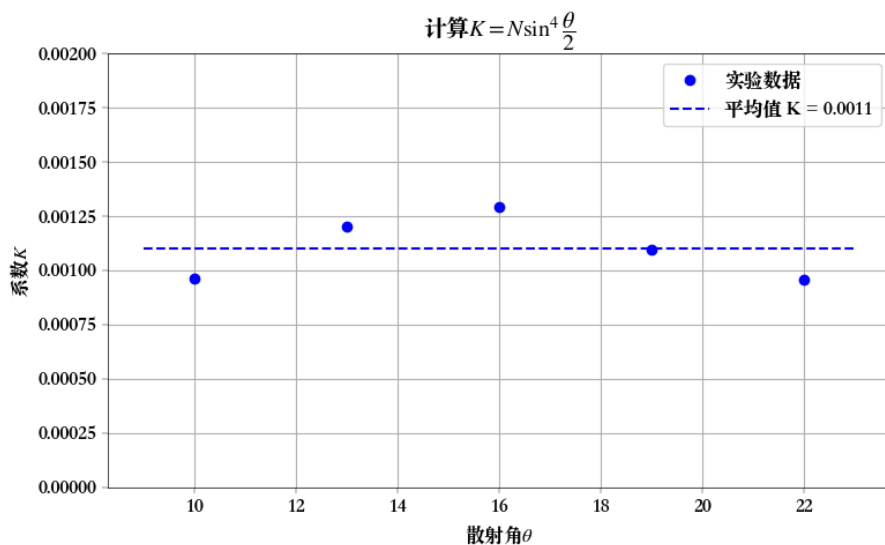


图 3:  $K - \theta$  曲线.

相比于理论计算 0.0018 误差为  $\frac{0.0018 - 0.0011}{0.0018} = 38.9\%$  .

## Newton 法与弦截法解非线性方程组

---

代码:

```
1 def Newton(f, df, x0, epsilon ,MAXSTEP = 1000):
2     iterate = [x0]
3     for _ in (range(MAXSTEP+1)):
4         if df(x0) != 0:
5             x1 = x0 - f(x0)/df(x0)
6         else:
7             raise ValueError("0 cannot be divided")
8         # print(x1)
9         iterate.append(x1)
10        if abs(x1 - x0) < epsilon:
11            return iterate
12        else:
13            x0 = x1
14    return "There's no root around x0.", None
15
16 def Secant(f, x1, x2, epsilon, MAXSTEP = 1000):
17     iterate = [x1, x2]
18     f1 = f(x1)
19     for _ in (range(MAXSTEP+1)):
20         f2 = f(x2)
21         if f2 - f1 != 0:
22             x = x2 - f2*(x2-x1)/(f2-f1)
23         else:
24             raise ValueError("0 cannot be divided")
25         # print(x1)
26         iterate.append(x)
27         if abs(x - x2) < epsilon:
28             return iterate
29         else:
30             f1 = f2
```

```
31         x1 = x2
32         x2 = x
33         return "There's no root around x0.", None
34
35 def f(x):
36     return (0.285 + 0.005 * x) * pow(x, 1.5) - 1.69
37
38 def df(x):
39     return 0.4275 * pow(x, 0.5) + 0.0125 * pow(x, 1.5)
40
41 if __name__ == "__main__":
42     x0_newton = [0.1, 0.2, 0.9, 9.0]
43     for i in range(len(x0_newton)):
44         res = Newton(f, df, x0_newton[i], 1e-8)
45         print(f"Newton: x0 = {x0_newton[i]}, root =
46               {res[-1]}, steps = {len(res) - 1}.")
47     x_secant = [[0, 0.1], [0, 0.2], [0, 1.0], [0, 9.0]]
48     for j in range(len(x0_newton[:])):
49         res = Secant(f, *x_secant[j], 1e-8)
50         print(f"Secant: x1 = {x_secant[j][0]}, x2 =
51               {x_secant[j][1]}, root = {res[-1]}, steps =
52               {len(res) - 1}.")
```

输出:

Newton: x0 = 0.1, root = 3.1603667461938865, steps = 7.

Newton: x0 = 0.2, root = 3.1603667461938865, steps = 7.

Newton: x0 = 0.9, root = 3.160366746193886, steps = 6.

Newton: x0 = 9.0, root = 3.1603667461938865, steps = 6.

Secant: x1 = 0, x2 = 0.1, root = 3.1603667461938842, steps = 10.

Secant: x1 = 0, x2 = 0.2, root = 3.1603667461938865, steps = 10.

Secant: x1 = 0, x2 = 1.0, root = 3.160366746193886, steps = 9.

Secant: x1 = 0, x2 = 9.0, root = 3.1603667461938865, steps = 9.