文件 编辑 查看 运行 内核 标签页 设置 帮助

Untitled.ipynb

Markdown

笔记本 Python 3 (ipykernel)

Rutherford Scattering (short project for PHYS 1113)

Coded by WANG, Xuechi

Analysed by LUO, Yueyang

[2]
```
import numpy as np
import matplotlib.pyplot as plt
import tqdm
```

[3]
```
from scipy import constants
```

Initialize some constants

[4]
```
Z1 = 2  # Atomic number of alpha particle
Z2 = 79  # Atomic number of gold nucleus
e = constants.e  # Elementary charge (C)
epsilon_0 = constants.epsilon_0  # Permittivity of free space (F/m)
(m, _, _) = constants.physical_constants['alpha particle mass']  # Mass of alpha particle (kg)
k = 10  # Initial distance in X direction (sufficiently large)
pi = constants.pi
```

Set initial condition

[5]
```
b = 2  # Impact parameter (can be varied)
X_0 = -k
Y_0 = b
V_X0 = 1.0
V_Y0 = 0.0
delta_tau = 0.001
```

[6]
```
V_X = np.array([V_X0])
V_Y = np.array([V_Y0])
X = np.array([X_0])
Y = np.array([Y_0])
```

Calculate motion by iteration

The following code may not use in the real experiment due to performance issue (np.append) but it's ok for small stimulation

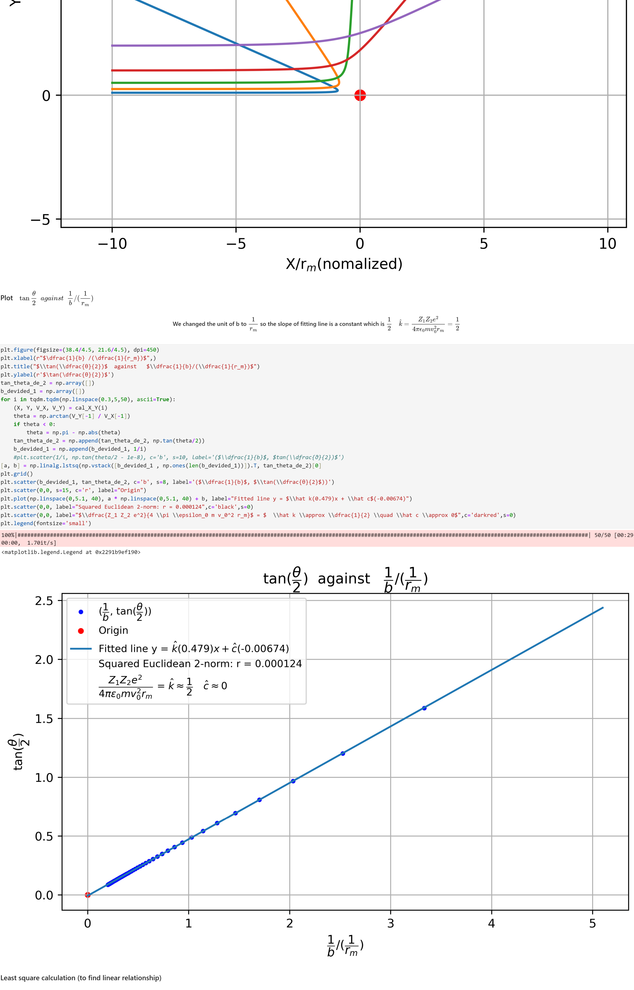
[7]
```
def cal_X_Y(b):
    X_0 = -k
    Y_0 = b
    V_X0 = 1.0
    V_Y0 = 0.0
    delta_tau = 0.001
    V_X = np.array([V_X0])
    V_Y = np.array([V_Y0])
    X = np.array([X_0])
    Y = np.array([Y_0])
    while (np.sqrt(X[-1]**2 + Y[-1]**2) < 12):  # R < 15
        X = np.append(X, X[-1] + delta_tau * V_X[-1])
        Y = np.append(Y, Y[-1] + delta_tau * V_Y[-1])
        A_X = X[-1]/(2*(np.sqrt(X[-1]**2 + Y[-1]**2)**3))
        A_Y = Y[-1]/(2*(np.sqrt(X[-1]**2 + Y[-1]**2)**3))
        V_X = np.append(V_X, V_X[-1] + delta_tau * A_X)
        V_Y = np.append(V_Y, V_Y[-1] + delta_tau * A_Y)
    return (X, Y, V_X, V_Y)
```

Plot figures of particle motion

[8]
```
plt.figure(figsize=(38.4/5.5, 38.4/5.5), dpi=550)
plt.scatter([0], [0], marker='o', linewidths=2, label='Gold particle', c='r')
plt.axis('equal')
plt.grid()
plt.title(f"Motion of alpha particle (b = 0.1, 0.25, 0.5, 1, 2)")
for i in [0.1,0.25,0.5,1,2]:
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    theta = np.arctan(V_Y[-1] / V_X[-1])
    if theta < 0:
        theta = np.pi - np.abs(theta)
    plt.plot(X, Y, label='b/r$_m$ = {:.2f}    '.format(i) + 'θ = {:.2f}$^o$'.format(theta*180/pi))
    print(i)
plt.xlabel("X/r$_m$(nomalized)")
plt.ylabel("Y/r$_m$ (nomalized)")
plt.legend(fontsize="small")
```

```
0.1
0.25
0.5
1
2
```

[8]
```
<matplotlib.legend.Legend at 0x204ffe43e10>
```



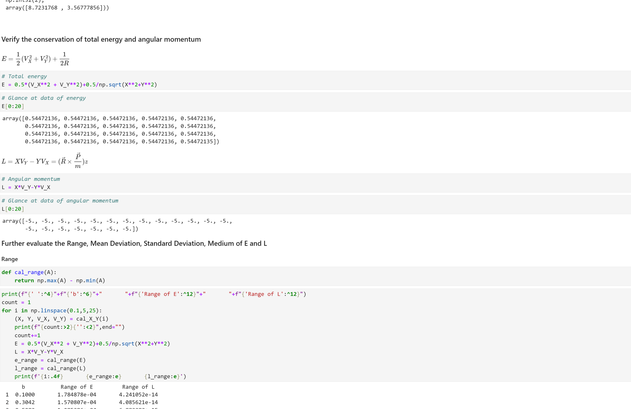
Plot $\tan\frac{\theta}{2}$ *against* $\frac{1}{b}/(\frac{1}{r_m})$

We changed the unit of b to $\frac{1}{r_m}$ so the slope of fitting line is a constant which is $\frac{1}{2}$ $\hat{k} = \frac{Z_1 Z_2 e^2}{4\pi\varepsilon_0 m v_0^2 r_m} = \frac{1}{2}$

[9]
```
plt.figure(figsize=(38.4/5.5, 21.6/4.5), dpi=450)
plt.xlabel("$\dfrac{1}{b} /(\dfrac{1}{r_m})$",)
plt.title("$\tan(\dfrac{θ}{2})$  against   $\dfrac{1}{b}/(\\dfrac{1}{r_m})$")
plt.ylabel("$\tan(\dfrac{θ}{2})$")
tan_theta_de_2 = np.array([])
b_devided_1 = np.array([])
for i in tqdm.tqdm(np.linspace(0.3,5,50), ascii=True):
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    theta = np.arctan(V_Y[-1] / V_X[-1])
    if theta < 0:
        theta = np.pi - np.abs(theta)
    tan_theta_de_2 = np.append(tan_theta_de_2, np.tan(theta/2))
    b_devided_1 = np.append(b_devided_1, 1/i)
    #plt.scatter(1/i, np.tan(theta/2 - 1e-6), c='b', s=10, label='($\\dfrac{1}{b}$, $tan(\\dfrac{θ}{2})$)')
[a, b] = np.linalg.lstsq(np.vstack([b_devided_1 , np.ones(len(b_devided_1))]).T, tan_theta_de_2)[0]
plt.grid()
plt.scatter(b_devided_1, tan_theta_de_2, c='b', s=8, label='($\\dfrac{1}{b}$$, $\\tan(\\dfrac{θ}{2})$)')
plt.scatter(0,0, s=15, c='r', label="Origin")
plt.plot(np.linspace(0,5.1, 40), a * np.linspace(0,5.1, 40) + b, label="Fitted line y = $\\hat k(0.479)x + \\hat c$(-0.00674)")
plt.scatter(0,0, label="Squared Euclidean 2-norm: r = 0.000124",c='black',s=0)
plt.scatter(0,0, label="$\\dfrac{Z_1 Z_2 e^2}{4 \\pi \\epsilon_0 m v_0^2 r_m}$ =  $  \\hat k \\approx \\dfrac{1}{2} \\quad \\hat c \\approx 0$",c='darkred',s=0)
plt.legend(fontsize='small')
```

```
100%|##########################################################################################################################################| 50/50 [00:29<
00:00,  1.70it/s]
```

[7]
```
<matplotlib.legend.Legend at 0x2291b9ef190>
```



Least square calculation (to find linear relationship)

$mx + c = y$

return m (Slope), c (Intercept), r (Residual)

[9]
```
np.linalg.lstsq(np.vstack([ b_devided_1 , np.ones(len(b_devided_1))]).T, tan_theta_de_2)
```

[9]
```
(array([ 0.47931386, -0.00674272 ]),
 array([0.00012417]),
 np.int32(2),
 array([8.7231768 , 3.56777856]))
```

Verify the conservation of total energy and angular momentum

$$E = \frac{1}{2}(V_X^2 + V_Y^2) + \frac{1}{2R}$$

[10]
```
# Total energy
E = 0.5*(V_X**2 + V_Y**2)+0.5/np.sqrt(X**2+Y**2)
```

[11]
```
# Glance at data of energy
E[0:20]
```

[11]
```
array([0.54472136, 0.54472136, 0.54472136, 0.54472136, 0.54472136,
       0.54472136, 0.54472136, 0.54472136, 0.54472136, 0.54472136,
       0.54472136, 0.54472136, 0.54472136, 0.54472136, 0.54472136,
       0.54472136, 0.54472136, 0.54472136, 0.54472136, 0.54472135])
```

$$L = XV_Y - YV_X = (\vec{R} \times \frac{\vec{P}}{m})_z$$

[12]
```
# Angular momentum
L = X*V_Y-Y*V_X
```

[13]
```
# Glance at data of angular momentum
L[0:20]
```

[13]
```
array([-5., -5., -5., -5., -5., -5., -5., -5., -5., -5., -5., -5., -5.,
       -5., -5., -5., -5., -5., -5., -5.])
```

Further evaluate the Range, Mean Deviation, Standard Deviation, Medium of E and L

Range

[14]
```
def cal_range(A):
    return np.max(A) - np.min(A)
```

[15]
```
print(f'{' ':4}{'b':6}{"+"      +f'{'Range of E':^12}{"+"      +f'{'Range of L':^12}')
count = 1
for i in np.linspace(0.1,5,25):
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    print(f"{count:>2}{' ':>2}",end="")
    count+=1
    E = 0.5*(V_X**2 + V_Y**2)+0.5/np.sqrt(X**2+Y**2)
    L = X*V_Y-Y*V_X
    e_range = cal_range(E)
    l_range = cal_range(L)
    print(f'{i:.4f}                {e_range:e}        {l_range:e}')
```

```
     b       Range of E       Range of L
 1  0.1000    1.784878e-04     4.241052e-14
 2  0.3042    1.570007e-04     4.885621e-14
 3  0.5083    1.285386e-04     6.883383e-15
 4  0.7125    1.028133e-04     4.218847e-14
 5  0.9167    8.239112e-05     6.128431e-14
 6  1.1208    6.677031e-05     3.463896e-14
 7  1.3250    5.486576e-05     2.997602e-14
 8  1.5292    4.571204e-05     4.618528e-14
 9  1.7333    3.857933e-05     1.083178e-13
10  1.9375    3.294126e-05     6.528111e-14
11  2.1417    2.843192e-05     3.330669e-14
```

12	2.3458	2.475193e-05	4.973799e-14
13	2.5500	2.173541e-05	5.240253e-14
14	2.7542	1.922875e-05	5.906386e-14
15	2.9583	1.712492e-05	5.062617e-14
16	3.1625	1.534312e-05	5.107026e-14
17	3.3667	1.382158e-05	3.597123e-14
18	3.5708	1.251247e-05	5.728751e-14
19	3.7750	1.137838e-05	5.062617e-14
20	3.9792	1.038970e-05	4.884981e-14
21	4.1833	9.522801e-06	1.101341e-13
22	4.3875	8.758609e-06	6.750156e-14
23	4.5917	8.081635e-06	8.348877e-14
24	4.7958	7.479185e-06	6.661338e-14
25	5.0000	6.940780e-06	8.704149e-14

Mean Deviation

[16…
```
def cal_mean_deviation(A):
    return np.sum(np.abs(A - np.mean(A)))
```

[17…
```
print(f"{' ':^4}"+f"{'b':^6}"+"+"        "+f"{'Mean Deviation of E':^0}"+"+"    "+f"{'Mean Deviation of L'}")
count = 1
for i in np.linspace(0.1,5,25):
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    print(f"{count:>2}{'':<2}",end="")
    count+=1
    E = 0.5*(V_X**2 + V_Y**2)+0.5/np.sqrt(X**2+Y**2)
    L = X*V_Y-Y*V_X
    e_mean_deviation = cal_mean_deviation(E)
    l_mean_deviation = cal_mean_deviation(L)
    print(f"{i:.4f}          {e_mean_deviation:e}            {l_mean_deviation:e}")
```

	b	Mean Deviation of E	Mean Deviation of L
1	0.1000	4.978914e-01	2.389738e-10
2	0.3042	4.569606e-01	1.371497e-10
3	0.5083	4.006224e-01	2.777334e-11
4	0.7125	3.469121e-01	8.983947e-11
5	0.9167	3.011266e-01	1.551100e-10
6	1.1208	2.632905e-01	1.009788e-10
7	1.3250	2.321179e-01	9.522916e-11
8	1.5292	2.062585e-01	2.056466e-10
9	1.7333	1.845882e-01	4.028620e-10
10	1.9375	1.662367e-01	3.088014e-10
11	2.1417	1.505349e-01	1.008265e-10
12	2.3458	1.369735e-01	1.952327e-10
13	2.5500	1.251587e-01	1.929070e-10
14	2.7542	1.147883e-01	2.146714e-10
15	2.9583	1.056208e-01	1.877849e-10
16	3.1625	9.746669e-02	2.218705e-10
17	3.3667	9.017099e-02	1.300369e-10
18	3.5708	8.361313e-02	1.777560e-10
19	3.7750	7.768778e-02	2.882787e-10
20	3.9792	7.231439e-02	1.960654e-10
21	4.1833	6.742057e-02	4.840572e-10
22	4.3875	6.294803e-02	2.823439e-10
23	4.5917	5.884750e-02	4.195675e-10
24	4.7958	5.507561e-02	2.517915e-10
25	5.0000	5.159796e-02	3.247038e-10

Standard Deviation

[18…
```
def cal_standard_deviation(A):
    return np.std(A)
```

[19…
```
print(f"{' ':^4}"+f"{'b':^6}"+"        "+f"{'STD of E':^12}"+"        "+f"{'STD of L':^12}")
count = 1
for i in np.linspace(0.1,5,25):
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    print(f"{count:>2}{'':<2}",end="")
    count+=1
    E = 0.5*(V_X**2 + V_Y**2)+0.5/np.sqrt(X**2+Y**2)
    L = X*V_Y-Y*V_X
    e_std = cal_standard_deviation(E)
    l_std = cal_standard_deviation(L)
    print(f"{i:.4f}          {e_std:e}          {l_std:e}")
```

	b	STD of E	STD of L
1	0.1000	3.320740e-05	1.194140e-14
2	0.3042	2.996694e-05	7.959646e-15
3	0.5083	2.556366e-05	1.515659e-15
4	0.7125	2.146605e-05	5.906410e-15
5	0.9167	1.808092e-05	1.183860e-14
6	1.1208	1.537878e-05	6.196091e-15
7	1.3250	1.322903e-05	5.641177e-15
8	1.5292	1.150501e-05	1.175675e-14
9	1.7333	1.010644e-05	2.438029e-14
10	1.9375	8.957156e-06	1.613814e-14
11	2.1417	8.001361e-06	5.976566e-15
12	2.3458	7.197113e-06	1.043728e-14
13	2.5500	6.513935e-06	1.154586e-14
14	2.7542	5.927313e-06	1.223040e-14
15	2.9583	5.419515e-06	1.120235e-14
16	3.1625	4.976414e-06	1.202829e-14
17	3.3667	4.587130e-06	7.435204e-15
18	3.5708	4.242674e-06	1.124232e-14
19	3.7750	3.936271e-06	1.507648e-14
20	3.9792	3.662050e-06	1.103614e-14
21	4.1833	3.415511e-06	2.693135e-14
22	4.3875	3.192811e-06	1.621267e-14
23	4.5917	2.990764e-06	2.264887e-14
24	4.7958	2.806762e-06	1.432514e-14
25	5.0000	2.638512e-06	2.016989e-14

Medium

[20…
```
def cal_medium(A):
    return np.median(A)
```

[22…
```
print(f"{' ':^4}"+f"{'b':^6}"+"        "+f"{'Medium of E':^11}"+"        "+f"{'Medium of L':^11}")
count = 1
for i in np.linspace(0.1,5,25):
    (X, Y, V_X, V_Y) = cal_X_Y(i)
    print(f"{count:>2}{'':<2}",end="")
    count+=1
    E = 0.5*(V_X**2 + V_Y**2)+0.5/np.sqrt(X**2+Y**2)
    L = X*V_Y-Y*V_X
    e_mid = cal_medium(E)
    l_mid = cal_medium(L)
    print(f"{i:.4f}          {e_mid:e}          {l_mid:e}")
```

	b	Medium of E	Medium of L
1	0.1000	5.500021e-01	-1.000000e-01
2	0.3042	5.499815e-01	-3.041667e-01
3	0.5083	5.499401e-01	-5.083333e-01
4	0.7125	5.498781e-01	-7.125000e-01
5	0.9167	5.497958e-01	-9.166667e-01
6	1.1208	5.496934e-01	-1.120833e+00
7	1.3250	5.495712e-01	-1.325000e+00
8	1.5292	5.494299e-01	-1.529167e+00
9	1.7333	5.492698e-01	-1.733333e+00
10	1.9375	5.490915e-01	-1.937500e+00
11	2.1417	5.488956e-01	-2.141667e+00
12	2.3458	5.486828e-01	-2.345833e+00
13	2.5500	5.484538e-01	-2.550000e+00
14	2.7542	5.482092e-01	-2.754167e+00
15	2.9583	5.479500e-01	-2.958333e+00
16	3.1625	5.476768e-01	-3.162500e+00
17	3.3667	5.473905e-01	-3.366667e+00
18	3.5708	5.470918e-01	-3.570833e+00
19	3.7750	5.467817e-01	-3.775000e+00
20	3.9792	5.464608e-01	-3.979167e+00
21	4.1833	5.461301e-01	-4.183333e+00
22	4.3875	5.457901e-01	-4.387500e+00
23	4.5917	5.454419e-01	-4.591667e+00
24	4.7958	5.450862e-01	-4.795833e+00
25	5.0000	5.447239e-01	-5.000000e+00

According Equations

$$m\frac{d^2\vec{r}}{dt^2} = \frac{Z_1 Z_2 e^2}{4\pi\epsilon_0 r^2}\hat{e}_r$$

$$\hat{e}_r = \frac{\vec{r}}{r} = \frac{x\hat{i} + y\hat{j}}{r}$$

$$\frac{1}{2}mv_0^2 = \frac{Z_1 Z_2 e^2}{4\pi\epsilon_0 r_m}\hat{e}_r$$

$$v_0 = \frac{r_m}{T}$$

$$T = \sqrt{\frac{2\pi m\epsilon_0 r_m^3}{Z_1 Z_2 e^2}}$$

$$t = T\tau \quad \vec{r} = r_m\vec{R}$$

$$\frac{d^2\vec{R}}{d\tau^2} = \frac{1}{2R^2}\hat{e}_r$$

$$A_X = \frac{X}{2R^3} \quad A_Y = \frac{Y}{2R^3}$$

$$R = \sqrt{X^2 + Y^2}$$

¶

$$X(n+1) = X(n) + V_X(n)\Delta\tau \quad V(n+1) = V(n) + A_X(n)\Delta\tau$$

$$\theta = \tan^{-1}\frac{V_Y}{V_X} \quad \theta = \pi - \tan^{-1}\frac{V_Y}{|V_X|}$$

$$E = \frac{1}{2}(V_X^2 + V_Y^2) + \frac{1}{2R} \quad L = XV_Y - YV_X = (\vec{R} \times \frac{\vec{P}}{m})z$$

$$\tan\frac{\theta}{2} = \frac{Z_1 Z_2 e^2}{4\pi\epsilon_0 m v_0^2}\frac{1}{b}$$