09/14/20 04:14:04 /home/ana/Documents/uni/PHS3000/code/alpha_particles.py

```python
# PHS3000
# alpha particles - Range and energy loss
# Ana Fabela, 09/09/2020
import os
from pathlib import Path
import monashspa.PHS3000 as spa
import numpy as np
import matplotlib.pyplot as plt
from scipy import special
import scipy.optimize

plt.rcParams['figure.dpi'] = 150

folder = Path('spectra')
os.makedirs(folder, exist_ok=True)

# Global prefixes and values SI units
x = np.linspace(0,1036,1024) # bins array

kilo = 1e3
cm = 1e-2 # [m]
g = 1e-3 # [kg]

eV = 1.602e-19 # [J]
MeV = eV * 1e6 # [J]
keV = eV * 1e3 # [J]

p_zero = 36.1 * kilo # [Pa]
u_p_zero = 0.5 * kilo

A_air = 14.5924
A_Au = 196.966570

rho_atm = 1.284 # [kg m**-3]
rho_gold = 19.30 * (g / cm**3) # [kg/ m3]

T = 294.15 # [K]
u_T = 0.5 # [K]

x_0  = 6.77 # [cm]
u_x_0 = 0.1 # [cm]

alpha_energy = 5.4857 # [MeV]


def read_files():
    read_folder = Path('log2')
    files = list(os.listdir(path=read_folder))
    data_files = []
    p_values = []

    files.sort(key=lambda name: int(name.split('_')[0]) if name[0].isdigit() else -1)
    for i, file in enumerate(files):
        # print(i, file)
        if i >= 1:
            p_value = float(file.rstrip('mbar.mca').replace('_', '.'))
            p_values.append(p_value)
        header, data = spa.read_mca_file(read_folder/file)
        data_files.append(data)
    p_values = np.asarray(p_values)
    u_p = np.sqrt((0.0025 * p_values)**2 + 0.01**2) # resolution and accuracy [kPa]
    return p_values, u_p, data_files

def calibrate_axis(x, E_0, u_E_0):
    # calibration factor for x-axis
    _1bin = 5 * keV # [J]
    u_1bin = (u_E_0 / E_0) * _1bin
    # therefore x -> E
    E = x * _1bin # [J]
    u_E = (u_1bin / _1bin) * E
    return E, u_E

def plot_data(x, y, i, average, ax=None):
    xmax = np.argmax(y)
```

```python
    ymax = y.max()
    half_y_max = ymax / 2
    L = np.argmin((y[:xmax] - half_y_max)**2)
    R = np.argmin((y[xmax:] - half_y_max)**2) + xmax
    peak_width = R - L

    # if i >= 1:
    #     text= "bin={:.0f}, count={:.0f}".format(xmax, ymax)
    #     if not ax:
    #         ax=plt.gca()
    #     ax.annotate(text, xy=(xmax, ymax), xytext=(0.7, 1.02), textcoords='axes fraction')

    # plt.bar(x, y, color='tomato',label="Detected alphas")
    # plt.plot(x[:41], average, label="Extrapolation") # extrapolation cuve
    # # plt.axhline(y=half_y_max, linestyle=':', alpha=0.3, label="Half max")
    # # plt.fill_betweenx([0, ymax + 20], [L, L], [R, R], alpha=0.3, zorder=10)
    # plt.xlim([0, 1024])
    # plt.xlabel('Bins')
    # plt.ylabel('Counts')
    # plt.title(f'file[{i}].mca')
    # plt.legend()

    # spa.savefig(folder/f'file{i}.png')
    # plt.show()
    # plt.clf()
    return xmax, ymax, peak_width

def energy_peaks(p_values, data_files):
    max_counts = []
    max_positions = []
    peak_widths = []
    total_events = []

    for i, signal in enumerate(data_files):
        # curve extrapolation for the threshold region
        average_list = [np.mean(signal[42:92])] * 41
        # sum of all events including extrapolation
        total = np.around(np.sum(signal) +  np.sum(average_list), decimals=0)
        total_events.append(total)

        # print(f"{np.sum(signal)} + {np.sum(average_list)} = {total}")

        # barcharts to visualise our files
        xmax, countmax, peak_width = plot_data(x, signal, i, average_list)

        max_positions.append(xmax)
        max_counts.append(countmax)
        peak_widths.append(peak_width)
    peak = max_positions[1]
    return peak, signal, total_events, max_positions, max_counts, peak_widths


def plot_pressure_vs_energy(p_values, max_positions):
    # pressure vs Energy peak
    plt.plot(p_values, max_positions[1:],'o', color='tomato', markersize=2.5, label="peak")
    plt.xlabel('pressure / kPa')
    plt.ylabel('Bin number')
    plt.title(f' Position of peak vs pressure ')
    plt.legend()
    spa.savefig(f'peak_position_vs_pressure.png')
    plt.show()

def Task_2(x_0, u_x_0):
    # Calculating E_0
    # from equation (5)
    R_0 = x_0
    u_R_0 = u_x_0
    # print(f"\n{T = :.2f} K, R = {R_0} cm, p = {p_zero / kilo} kPa")
    E_0 = (R_0 * p_zero / (64.31 * T))**(1 / 1.73)
    u_E_0 = (E_0 / 1.73) * np.sqrt((u_R_0 / R_0)**2 + (u_p_zero / p_zero)**2 + (u_T / T)**2)
    print(f"\nE_0 = {E_0:.2f} ± {u_E_0:.2f} MeV")
    # # difference in energy
    diff_range_E = alpha_energy - E_0
    u_diff_range_E = u_E_0
    print(f"\n{diff_range_E = :.2f} ± {u_diff_range_E:.2f}")
    return R_0, u_R_0, E_0, u_E_0

```

```python
def Task_3(E_0, u_E_0, rho_atm, rho_gold, A_Au, A_air, alpha_energy):
    # Calculating R_Au:
    # equation (4)
    R_atm = 0.186 * E_0**1.73
    u_R_atm = R_atm * 1.73* u_E_0 / E_0
    # print(f"\nTheoretical Range in 1 atm: {R_atm = :.2f} ± {u_R_atm:.2f} cm")

    # Calculating anticipated range for particles travelling through gold
    # equation (13)
    R_Au = R_atm * (rho_atm / rho_gold) * np.sqrt(A_Au / A_air)
    u_R_Au = R_Au * (u_R_atm / R_atm)
    # print(f"{R_Au = } ± {u_R_Au} cm")

    # Calculating the thickness (delta_x) of the gold coating
    # rearranged equation (14)
    delta_x = np.sqrt(A_Au) * (E_0**1.73 -alpha_energy**(1/0.578))  / (4.464 * rho_gold * np.sqrt(A_air))
    u_delta_x = 1.73 * u_E_0 / E_0
    # print(f"\n{delta_x = } ± {u_delta_x} cm")
    return R_atm, u_R_atm, R_Au, u_R_Au, delta_x, u_delta_x


def Task_4(E_0, u_E_0, R_atm, u_R_atm):
    # theoretical plot of range vs energy of alpha particles in air at atm pressure
    x, y = [E_0], [R_atm]
    E_th =  np.linspace(0,6,1024) # (0 - 6) MeV array
    R_th = 0.186 * (E_th**1.73)

    plt.plot(
        E_th, R_th, marker="None",
        linestyle="-", label=r"$R_{th}(E_0)$"
        )
    plt.errorbar(
            x, y, xerr=u_E_0, yerr=u_R_0,
            marker="None", linestyle="None", ecolor="tomato",
            label=r'$R_{atm}$', color="tomato", barsabove=True
        )

    plt.xlabel(r'$E_0$ / MeV')
    plt.ylabel('R / cm')
    plt.title(r'Theoretical plot of R vs $E_0$ of alpha particles in air')
    plt.legend()
    spa.savefig(f'theoretical_R_vs_E.png')
    plt.show()

def Task_5(peak_widths, p_values):
    # plot of FWHM vs pressure
    plt.plot(
        p_values, peak_widths[1:], 'o', color='tomato', markersize=2.5, label=r"$FWHM(p)$"
        )
    plt.axvline(x=p_zero/kilo, linestyle='--', alpha=0.5, label="37 kPa" )
    plt.grid(linestyle=':')
    plt.xlabel(r'$p$ / kPa')
    plt.ylabel('FWHM')
    plt.title(r'Width of energy spectra vs pressure')
    plt.legend()
    spa.savefig(f'FWHM_vs_pressure.png')
    plt.show()

def Task_6(total_events):
    # The effect  of  range  straggling.
    # How does the number of surviving particles vary with pressure.

    plt.plot(
        p_values, total_events[1:], 'o', markersize=2.5, color='tomato', label=r"detected $alphas$"
        )
    # plt.grid(linestyle=':')
    plt.xlabel(r'$p$ / kPa')
    plt.ylabel('detected particles')
    plt.title(r'Surviving particles vs pressure')
    plt.legend()
    spa.savefig(f'alphas_vs_pressure.png')
    plt.show()


def f(p_values, p_R, α):
    # model for optimize.curve_fit()
    return (1 / 2) * (1 - special.erf((p_values - p_R) / α))
```

```python

def Task_7_8(f, x, y):
    p_0 = p_zero / kilo # [kPa]
    u_p_0 = u_p_zero / kilo
    # print(x)
    # print(f"{p_0}")

    # determining straggling parameter α
    popt, pcov = scipy.optimize.curve_fit(f, x, y)
    # To compute one standard deviation errors on parameter α
    perr = np.sqrt(np.diag(pcov))

    p_R, pα = popt
    u_p_R, u_pα = perr

    p = np.linspace(x[0], x[-1], 25)
    optimal_fit = f(p, p_R, pα)

    plt.plot(
            x, y, marker='o', linestyle='None', markersize=2.5, color='tomato',
            label=r"detected $alphas$"
    )
    plt.plot(
            p, optimal_fit, marker="None",
            linestyle="-",
            label="fit"
    )

    plt.grid(linestyle=':')
    plt.xlabel(r'p / kPa')
    plt.ylabel('Detected particles')
    plt.title(r'Number alpha particles as a function of pressure')
    plt.legend()
    spa.savefig(f'alphas_vs_pressure.png')
    # plt.show()
    return p_R, pα, u_p_R, u_pα, p_0, u_p_0

def compare(p_R, pα, u_p_R, u_pα, p_0, u_p_0):
    # conversion to distance units
    p_atm = 101.325 # [kPa]
    u_p_atm = np.sqrt((0.0025 * p_atm)**2 + 0.01**2) # [kPa]

    xα = (pα / p_atm) * R_atm
    u_xα = xα * np.sqrt((u_pα / pα)**2 + (u_p_atm / p_atm)**2 + (u_R_atm /R_atm)**2)
    # comparison to theory
    k = 0.015
    diff_range = R_atm - xα
    how_many_sigmas = diff_range / u_xα
    print(f"\nEXPECTED RESULT α ≅ {k * R_atm = :.4f} ± {k * u_R_atm:.4f} cm")
    print(f"Experimental α = {xα:.2f} ± {u_xα:.2f} cm")
    # print(f"difference {diff_range:.3f}")
    print(f"number of σ away from true result: {abs(how_many_sigmas):.3f}")

    diff_p = p_0 - p_R
    how_many_sigmas = diff_p / u_p_R
    print(f"\nPrevious p_0: {p_0:.2f} ± {u_p_0:.2f} kPa")
    print(f"fit p_0 {p_R:.2f} ± {u_p_R:.2f} kPa")
    # print(f"difference {diff_p:.3f}")
    print(f"number of σ away from true result: {abs(how_many_sigmas):.3f}")


### * FUNCTION CALLS *###

p_values, u_p, data_files = read_files()

peak, signal, total_events, max_positions, max_counts, peak_widths = energy_peaks(p_values, data_files)

plot_pressure_vs_energy(p_values, max_positions)

R_0, u_R_0, E_0, u_E_0 = Task_2(x_0, u_x_0)

R_atm, u_R_atm, R_Au, u_R_Au, delta_x, u_delta_x = Task_3(E_0, u_E_0, rho_atm, rho_gold, A_Au, A_air,
alpha_energy)

E, u_E = calibrate_axis(x, E_0, u_E_0)

# Task_4(E_0, u_E_0, R_atm, u_R_atm)
```

```

# Task_5(peak_widths, p_values)

# Task_6(total_events)

y = total_events[1:] / total_events[1]
p_R, pα, u_p_R, u_pα, p_0, u_p_0 = Task_7_8(f, p_values, y)
# The straggling parameter can be expressed either as a pressure or a distance
# (at 1 atm pressure) it's just proportional to the range value in the units
 # you've expressed it in.
print(f"\n{pα = :.1f} ± {u_pα:.1f} kPa")

compare(p_R, pα, u_p_R, u_pα, p_0, u_p_0)

# Which  result do you think is more accurate and why?


```