```python
# -*- coding: utf-8 -*-
"""
File Name: compton_analysis.py
Purpose:
Author: Samuel Wong
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.integrate import simps

#================basic laod and plots functions=====================
def load_raw(file):
    data = np.loadtxt(file)
    channel = data[:,0]
    counts = data[:,1]
    return channel, counts

def plot(channel,counts,title):
    plt.figure()
    plt.title(title)
    plt.scatter(channel,counts)
    #x=np.linspace(450,575,10000)
    #plt.plot(x,gaussian(x,160,510,10,0),color='black')
    plt.ylabel('Counts')
    plt.xlabel('Channels')
    plt.savefig(title+".png")
    plt.show()

def load_and_plot_raw(file,title):
    channel, counts = load_raw(file)
    plot(channel,counts,title)
    return channel, counts

#================Restrict array range=====================
#because most of the channel (energy) range only contain noise, we cut both the
#channel and counts array to only restrict to 450 to 575, approximately the
#the channels where signal are present
def restrict_array(array):
    return array[450:575]

#===============Adjusted load function ==========================
def load_adjusted(file):
    channel, counts = load_raw(file)
    #adjust counts of a sample by restricting range and subtracting noise
    channel = restrict_array(channel)
    counts = restrict_array(counts)
    counts = np.abs(counts - noise) #counts must be non-negative
    return channel, counts

#================Fitted load and plot functions ==========================
#fit the adjusted counts and then plots
def load_fitted_plot_integrate(file_name,title,p0=None):
    #load restricted and noise adjusted values
    channel, counts = load_adjusted("data/"+file_name)
    #fit to gaussian
    popt, pcov = curve_fit(gaussian, xdata=channel, ydata=counts,p0=p0)
```

```
    #standard deviation of parameters
    pstd = np.sqrt(np.diag(pcov))
    #maximum parameters with uncertainties
    popt_max = popt + pstd
    popt_min = popt - pstd
    #plot fitted gaussian and max/min gaussian
    plot_fitted(counts, popt, popt_max, popt_min, title)
    #integrate for optimal and max/min parameters
    x=np.linspace(450,575,1000)
    integral = simps(gaussian(x,*popt),x)
    max_integral = simps(gaussian(x,*popt_max),x)
    min_integral = simps(gaussian(x,*popt_min),x)
    #estimate uncertainty by averaging the max/min difference
    uncertainty = (max_integral - min_integral)/2
    return integral, uncertainty

def plot_fitted(counts,popt,popt_max,popt_min,title,):
    plt.figure()
    plt.title(title)
    plt.scatter(channel,counts)
    x=np.linspace(450,575,1000)
    plt.plot(x,gaussian(x,*popt),color='black',label = "optimal fit")
    plt.plot(x,gaussian(x,*popt_max),color='green', ls='--', label = "max/min fit")
    plt.plot(x,gaussian(x,*popt_min),color='green', ls='--')
    plt.ylabel('Counts')
    plt.xlabel('Channels')
    plt.legend()
    plt.savefig(title+".png")
    plt.show()

#================Gaussian model function ==========================
def gaussian(x,a,b,c,d):
    return a*np.exp(-((x-b)**2)/(2*c**2))+d

#=================Load measurements of sample mass and dimensions==========
def get_measurements(title):
    data = np.loadtxt("data/"+title)
    m = data[:,1]
    dm = data[:,2]
    x = data[:,3]
    dx = data[:,4]
    diam = data[:,5]
    d_diam = data[:,6]
    return m,dm,x,dx,diam,d_diam

#===================Plot photon counts as a function of thickness ============

def plot_I_vs_x(Element_name,x,dx,I,dI,y_max):
    plt.figure()
    plt.errorbar(x=x,y=I,xerr=dx,yerr=dI,fmt='o')
    plt.title("I vs x for {}".format(Element_name))
    plt.xlabel('absorber thickness, x (mm)')
    plt.ylabel('photon counts, I (over 600s)')
    plt.ylim(0,y_max)
    plt.savefig("I vs x for {}.png".format(Element_name))
    plt.show()
```

```
def fit_and_plot_I_ratio_vs_x(Element_name,x,dx,I_ratio,d_I_ratio,y_max):
    #fit the photon ratio I/I_0 to exponential
    #since x is in mm, coeff is in mm^-1. Corresponds to mm^-3 of number
    #desnity times mm^2 of cross section
    popt, pcov = curve_fit(exponential,xdata=x,ydata=I_ratio,sigma=d_I_ratio,
                           p0=(0.1))
    coeff = popt[0]
    d_coeff = np.sqrt(pcov[0][0])
    plt.figure()
    plt.errorbar(x=x,y=I_ratio,xerr=dx,yerr=d_I_ratio,fmt='o')
    x_array = np.linspace(0,np.max(x)*1.1,1000)
    plt.plot(x_array,exponential(x_array,coeff),label='exp(-n*sigma_tot*x)')
    plt.title("$\\dfrac{I(x)}{I(0)}$" + " vs x for {}".format(Element_name))
    plt.xlabel('absorber thickness, x (mm)')
    plt.ylabel("$\\dfrac{I(x)}{I(0)}$")
    plt.ylim(0,y_max)
    plt.legend()
    plt.savefig("I ratio vs x for {}.png".format(Element_name))
    plt.show()
    return coeff, d_coeff

def get_atomic_number_density(molar_mass,m,dm,diam,d_diam,x,dx):
    r = diam/2
    dr = d_diam/2
    V = x*np.pi*(r**2) #volume
    #using the computation dV = V sqrt( (d(r^2)/r^2)^2 + (dx/x)^2)
    #and d(r^2) = 2r dr, we have dV = V sqrt( 4dr^2/r^2 + (dx/x)^2 )
    dV = V * np.sqrt(4*(dr**2)/(r**2) + (dx**2)/(x**2))
    avagadro = 6.02 * (10**23)
    #using the identity n = (m/M)*N_a/V
    n = ((m/molar_mass)*avagadro)/V #the unit of gram matches; in mm^-3
    dn = n* np.sqrt( (dm/m)**2 + (dV/V)**2 )
    #average all the atomic density of different samples
    n = np.average(n)
    dn = np.average(dn)
    return n, dn

def exponential(x,coeff):
    return np.exp(-coeff*x)

def get_total_cross_section(coeff,d_coeff,n,dn):
    sigma_tot = coeff/n
    d_sigma_tot = sigma_tot * np.sqrt( (d_coeff/coeff)**2 + (dn/n)**2 )
    return sigma_tot, d_sigma_tot

def get_final_cross_section_low_Z(sigma_tot,d_sigma_tot,Z):
    #using the formula sigma_tot = Z*sigma
    sigma = sigma_tot/Z
    d_sigma = sigma* (d_sigma_tot/sigma_tot)
    return sigma, d_sigma

def get_final_cross_section_high_Z(sigma_tot,d_sigma_tot,Z):
    #using the formula sigma_tot = Z*sigma + p*Z**(4.2)
    sigma = (sigma_tot - p*(Z**4.2))/Z
    d_sigma = sigma* (d_sigma_tot/sigma_tot)
    return sigma, d_sigma
```

```
def extract_electron_radius(sigma,d_sigma):
    #use the formula sigma = 2*pi*(r**2)*Klein_Ninshima_factor
    #gamma is energy of gamma ray in unit of m_electron*c**2
    #since gamma in energy = 0.6617 MeV and m_electron*c**2=0.511 MeV
    gamma = 0.6617/0.511 #energy of gamma ray in unit of electron mass
    A = (1+gamma)/gamma**2
    B = 2*(1+gamma)/(1+2*gamma)
    L = np.log(1+2*gamma)/gamma
    C = (1+3*gamma)/(1+2*gamma)**2
    Klein_Ninshima_factor = A*(B-L)+L/2-C
    #extract classical electron radius
    r_electron = np.sqrt((sigma/Klein_Ninshima_factor)/(2*np.pi))
    #using the formula dr = dr/d_sigma * d_sigma
    d_r_electron = 1/(r_electron*4*np.pi*Klein_Ninshima_factor) * d_sigma
    return r_electron, d_r_electron

def extract_fine_structure(r, dr):
    #extract fine structure constant from the electron radius using the formula
    #r= alpha* (hbar * c)/(m_e*c**2)
    hbar_times_c = 197.32*(10**(-15)) #in MeV*m
    electron_mass = 0.511 #in MeV
    alpha = r*electron_mass/hbar_times_c
    d_alpha = dr*electron_mass/hbar_times_c
    return alpha, d_alpha

def get_inverse_alpha(alpha,d_alpha):
    inverse_alpha = 1/alpha
    #negative sign taken out by square and squareroot
    d_inverse_alpha = (1/alpha**2)*d_alpha
    return inverse_alpha, d_inverse_alpha

def sigma_total_model(Z,p):
    gamma = 0.6617/0.511 #energy of gamma ray in unit of electron mass
    A = (1+gamma)/gamma**2
    B = 2*(1+gamma)/(1+2*gamma)
    L = np.log(1+2*gamma)/gamma
    C = (1+3*gamma)/(1+2*gamma)**2
    Klein_Ninshima_factor = A*(B-L)+L/2-C
    r = 2.812*(10**(-15)) #classical electron radius given in lab manual
    sigma = 2*np.pi*(r**2)*Klein_Ninshima_factor
    sigma_tot = p*Z**(4.2) + Z*sigma
    return sigma_tot

def analyze_element(I,dI,measurement_file,I_plot_max,Element_name,
                    molar_mass,Z,high_Z=False):
    #get all experimental measurements from textfile
    #in units of gram, mm, and mm, respectively
    m,dm,x,dx,diam,d_diam = get_measurements(measurement_file)
    #plot photon counts vs absorber thickness
    plot_I_vs_x(Element_name=Element_name,x=x,dx=dx,I=I,dI=dI,y_max=I_plot_max)
    #get uncertianty of I/I_0
    I_ratio = I/I_0
    d_I_ratio = (I/I_0) * np.sqrt((dI/I)**2 + (dI_0/I_0)**2)
    #plot photon ratio I/I_0
    coeff, d_coeff = fit_and_plot_I_ratio_vs_x(Element_name=Element_name,x=x,dx=dx,
                                    I_ratio=I_ratio,d_I_ratio=d_I_ratio,y_max=1.1)
    print('n*sigma_tot = ',coeff,'+/-',d_coeff)
```

```
    #compute atomic number density of absorbers, n, by measuring the mass,
    #thickness, and diameter of each absorbers. As an approximation, we assume
    #oll absorbers of the same material have the same n, and in practice, we
    #average the various density measurements
    #molar mass is in gram/mol
    n, dn = get_atomic_number_density(molar_mass,m,dm,diam,d_diam,x,dx)
    print('n_{}='.format(Element_name),n,'+/-',dn, 'mm^-3')
    #extract total cross section from coefficient in exponential
    #unit is mm^2
    sigma_tot, d_sigma_tot = get_total_cross_section(coeff,d_coeff,n,dn)
    print('sigma_tot =',sigma_tot,'+/-',d_sigma_tot, 'mm^2')
    #convert total cross section into unit of m^2
    sigma_tot = sigma_tot*(10**-6)
    d_sigma_tot = d_sigma_tot*(10**-6)
    print('sigma_tot =',sigma_tot,'+/-',d_sigma_tot, 'm^2')
    #use formula for low Z, where we can ignore photoelectric effect
    if high_Z:
        sigma, d_sigma = get_final_cross_section_high_Z(sigma_tot,d_sigma_tot,Z)
    else:
        sigma, d_sigma = get_final_cross_section_low_Z(sigma_tot,d_sigma_tot,Z)
    print('sigma = ',sigma,'+/-',d_sigma,'m^2')
    #extract classical electron radius from Klein_Nishima formula
    r_electron, d_r_electron = extract_electron_radius(sigma,d_sigma)
    print('r_electron = ', r_electron,'+/-', d_r_electron, 'm')
    #extract fine structure constant from the electron radius
    alpha, d_alpha = extract_fine_structure(r_electron, d_r_electron)
    print('alpha = ',alpha,'+/-',d_alpha)
    #get inverse of alpha, which is supposed to be 137
    inverse_alpha, d_inverse_alpha = get_inverse_alpha(alpha,d_alpha)
    print('1/alpha = ', inverse_alpha, '+/-', d_inverse_alpha)

#load and plot the raw (unrestricted) noise counts
channel_raw, noise_raw = load_and_plot_raw("data/noise.txt","noise")
#global variables: create restricted array range  of channel and noise array
channel = restrict_array(channel_raw) #only need one channel array from now on
noise = restrict_array(noise_raw)

#load and get the total counts and its unertainty for I(0), the photons count
#for when there is no absorber but with a source
I_0, dI_0 = load_fitted_plot_integrate("I(0).txt","I(0)",(160,510,10,0))
print(I_0, '+/-', dI_0)

print("====================Aluminum Analysis============================")
#load and plot different absorbers with noise subtracted and integrate the
#total counts, get the total counts and its uncertainty
alum1, dalum1 = load_fitted_plot_integrate(
        "aluminum#1.txt","aluminum#1",(160,510,10,0))
print(alum1, '+/-', dalum1)
alum2, dalum2 = load_fitted_plot_integrate(
        "aluminum#2.txt","aluminum#2",(160,510,10,0))
print(alum2, '+/-', dalum2)
alum3, dalum3 = load_fitted_plot_integrate(
        "aluminum#3.txt","aluminum#3",(160,510,10,0))
print(alum3, '+/-', dalum3)
alum4, dalum4 = load_fitted_plot_integrate(
        "aluminum#4.txt","aluminum#4",(160,510,10,0))
print(alum4, '+/-', dalum4)
```

```
alum5, dalum5 = load_fitted_plot_integrate(
        "aluminum#5.txt","aluminum#5",(160,510,10,0))
print(alum5, '+/-', dalum5)
#collect all integral photon counts into an array, ordered by their
#experimental label number
I = np.array([alum1,alum2,alum3,alum4,alum5])
dI = np.array([dalum1,dalum2,dalum3,dalum4,dalum5])

analyze_element(I,dI,measurement_file = "aluminum_measurements.txt",
                I_plot_max=6100,Element_name="Aluminum",molar_mass=26.982,
                Z=13)

print("==================Carbon Analysis============================")
#load and plot different absorbers with noise subtracted and integrate the
#total counts, get the total counts and its uncertainty
carbon1, dcarbon1 = load_fitted_plot_integrate(
        "carbon#1.txt","carbon#1",(160,510,10,0))
print(carbon1, '+/-', dcarbon1)
carbon2, dcarbon2 = load_fitted_plot_integrate(
        "carbon#2.txt","carbon#2",(160,510,10,0))
print(carbon2, '+/-', dcarbon2)
carbon3, dcarbon3 = load_fitted_plot_integrate(
        "carbon#3.txt","carbon#3",(160,510,10,0))
print(carbon3, '+/-', dcarbon3)
carbon4, dcarbon4 = load_fitted_plot_integrate(
        "carbon#4.txt","carbon#4",(160,510,10,0))
print(carbon4, '+/-', dcarbon4)
carbon5, dcarbon5 = load_fitted_plot_integrate(
        "carbon#5.txt","carbon#5",(160,510,10,0))
print(carbon5, '+/-', dcarbon5)
carbon6, dcarbon6 = load_fitted_plot_integrate(
        "carbon#6.txt","carbon#6",(160,510,10,0))
print(carbon6, '+/-', dcarbon6)
#collect all integral photon counts into an array, ordered by their
#experimental label number
I = np.array([carbon1,carbon2,carbon3,carbon4,carbon5,carbon6])
dI = np.array([dcarbon1,dcarbon2,dcarbon3,dcarbon4,dcarbon5,dcarbon6])
analyze_element(I,dI,measurement_file = "carbon_measurements.txt",
                I_plot_max=5000,Element_name="Carbon",molar_mass=12.0107,
                Z=6)

print("==================Copper Analysis============================")
#load and plot different absorbers with noise subtracted and integrate the
#total counts, get the total counts and its uncertainty
copper1, dcopper1 = load_fitted_plot_integrate(
        "copper#1.txt","copper#1",(160,510,10,0))
print(copper1, '+/-', dcopper1)
copper2, dcopper2 = load_fitted_plot_integrate(
        "copper#2.txt","copper#2",(160,510,10,0))
print(copper2, '+/-', dcopper2)
copper3, dcopper3 = load_fitted_plot_integrate(
        "copper#3.txt","copper#3",(160,510,10,0))
print(copper3, '+/-', dcopper3)
copper4, dcopper4 = load_fitted_plot_integrate(
        "copper#4.txt","copper#4",(160,510,10,0))
print(copper4, '+/-', dcopper4)
copper5, dcopper5 = load_fitted_plot_integrate(
```

```
        "copper#5.txt","copper#5",(160,510,10,0))
print(copper5, '+/-', dcopper5)
#collect all integral photon counts into an array, ordered by their
#experimental label number
I = np.array([copper1,copper2,copper3,copper4,copper5])
dI = np.array([dcopper1,dcopper2,dcopper3,dcopper4,dcopper5])
analyze_element(I,dI,measurement_file = "copper_measurements.txt",
                I_plot_max=5000,Element_name="Copper",molar_mass=63.546,
                Z=29)

print("====================Photoelectric Coefficient Analysis================")
#we are given that for large Z, the photoelectric contribution to the cross
#section starts to matter, and that sigma_photoelectric is proportioanl to
#Z^(4.2).
#Then we can write sigma_pe = p Z^(4.2), where 'p' is the photoelectric
#coefficient for its contribution to the cross section. To analyze the data
#for high Z element such as lead, we need to find p first. I will now proceed
#to estimate 'p' from the data of copper, carbon, and aluminum, using the known
#ideal value of alpha=1/137
#First, note that if we set alpha=1/137, then the Compton cross section is the
#same for all 3 elements. In the equation,
#sigma_total = p Z^(4.2) + Z sigma,
#if sigma is a constant, and we have 3 pairs of (Z,sigma_total) for 3 elements,
#we can get a fit to extract p. This is exactly what I will do in the following.
Z = np.array([6,13,29,82]) #carbon, aluminum, copper, lead
#sigma_tot in m^2, copied from output of above sections
sigma_tot = np.array([1.463*10**(-28),3*10**(-28),7.0699*10**(-28),3.05*10**(-27)])
popt, pcov = curve_fit(sigma_total_model,xdata=Z,ydata=sigma_tot)
plt.figure()
plt.title("Total Cross Section Versus Atomic Number")
plt.scatter(Z,sigma_tot)
Z_array = np.linspace(0,90,1000)
plt.plot(Z_array,sigma_total_model(Z_array,*popt))
plt.ylim(0,5*10**(-27))
plt.ylabel("sigma_tot")
plt.xlabel("Z")
plt.savefig("Total Cross Section Versus Atomic Number.png")
plt.show()
#set global constant p
p = popt[0]

print("====================Lead Analysis=============================")
#load and plot different absorbers with noise subtracted and integrate the
#total counts, get the total counts and its uncertainty
#lead1, dlead1 = load_fitted_plot_integrate(
#        "lead#1.txt","lead#1",(20,510,10,0))
#print(lead1, '+/-', dlead1)
lead2, dlead2 = load_fitted_plot_integrate(
        "lead#2.txt","lead#2",(160,510,10,0))
print(lead2, '+/-', dlead2)
lead3, dlead3 = load_fitted_plot_integrate(
        "lead#3.txt","lead#3",(160,510,10,0))
print(lead3, '+/-', dlead3)
lead4, dlead4 = load_fitted_plot_integrate(
        "lead#4.txt","lead#4",(160,510,10,0))
print(lead4, '+/-', dlead4)
#collect all integral photon counts into an array, ordered by their
```

```
#experimental label number
I = np.array([lead2,lead3,lead4])
dI = np.array([dlead2,dlead3,dlead4])
analyze_element(I,dI,measurement_file = "lead_measurements.txt",
                I_plot_max=4000,Element_name="lead",molar_mass=207.2,
                Z=82,high_Z=True)
```