Neutron Penetrations through Mediums of Finite Thickness: Water, Lead and Graphite

10090371
Jiachen Guo

School of Physics and Astronomy
The University of Manchester

Second Year Computational Physics Report

May 2018

Abstract

The transmission, reflection and absorption of thermal neutrons incident normally on mediums of finite thickness were investigated in this experiment using Monte Carlo simulations. The transmission, reflection and absorption fractions of 4000-6000 neutrons fired at 10 cm slabs of water, lead and graphite were measured. The transmission, reflection and absorption fractions for 10 cm of water were measured to be $(0.35 \pm 0.03)\%$, $(79.81 \pm 0.12)\%$ and $(19.84 \pm 0.10)\%$ respectively. The corresponding transmission, reflection and absorption fractions measured for 10 cm slabs of lead and graphite were $(27.98 \pm 0.11)\%$, $(61.68 \pm 0.11)\%$, $(10.34 \pm 0.08)\%$ and $(30.74 \pm 0.12)\%$, $(68.37 \pm 0.12)\%$, $(0.89 \pm 0.02)\%$ respectively. The variation of the fractions with slab thickness for each medium was also investigated and the characteristic attenuation lengths of thermal neutrons in each medium were determined. They were calculated to be (2.08 ± 0.04) cm for water, (12.95 ± 0.37) cm for lead and (40.00 ± 2.77) cm for graphite. The values were significantly larger than the mean free path values, which are 0.29 cm for water, 2.67 cm for lead and 2.52 cm for graphite.

1 Introduction

Free neutrons are produced in large numbers in nuclear power plants. The study of the transport of neutrons and their interaction with matter are of paramount importance for the calculations of nuclear reactors. In the case of thermal neutrons, a neutron can either be scattered or absorbed when collided with a target nucleus or particle. For scattering processes, the target nucleus emits a single neutron after the neutron-nucleus interaction. For absorption processes, the neutron is completely absorbed by the target nucleus and results in either the formation of a compound nucleus with emission of gamma rays or a nuclear-induced fission reaction [1]. The study of neutrons transport in matter helps develop better technology in radiation shielding that protects people and the environment from harmful ionizing radiation [2].

2 Theory

In nuclear physics, the probability of occurrence of the neutron-nucleus interactions that happen when a neutron collides with a target nucleus is described in terms of quantities known as cross sections.

Consider a thin film of material, the microscopic cross-section of a particular neutron-nucleus interaction for a this material is defined as

$$\sigma = \frac{C}{N_a I_0}, \tag{1}$$

where C is the number of interactions per unit area per second, N_a is the nuclei density per unit area of the material and I_0 is the intensity of the neutron beam incident on the material.

For a neutron beam of intensity I_0 incident on a slab of material, the resulting beam intensity at a distance x from the incident surface of the material is given by

$$I(x) = I_0 e^{-N\sigma x}, \tag{2}$$

where N is the nuclei density per unit volume of the material. The term $N\sigma$ is also called the macroscopic cross-section Σ of the neutron-nucleus interaction of the material.

In the case of multiple neutron-nucleus interactions possible, the total macroscopic cross-section Σ_T is just the sum of all the macroscopic cross-sections, as in

$$\Sigma_T = \Sigma_1 + \Sigma_2 + \Sigma_3 + \cdots = N_1\sigma_1 + N_2\sigma_2 + N_3\sigma_3 + \cdots, \tag{3}$$

where the integer numbers represent each of the possible neutron-nucleus interaction.

The mean free path λ of the neutron in the slab of material is then given by

$$\lambda = \frac{1}{\Sigma_T}. \tag{4}$$

Equation 2 can then also be rewritten as

$$I(x) = I_0 e^{-\frac{x}{\lambda}}. \tag{5}$$

For the case of thermal neutrons, only two neutron-nucleus interactions are possible: scattering and absorption. The probability of scattering σ_s and the probability of absorption σ_a when a neutron collides with a nucleus are then given by

$$p_s = \sigma_s/\sigma_t, \tag{6}$$

$$p_a = \sigma_a/\sigma_t, \tag{7}$$

where σ_s is the microscopic cross-section for scattering, σ_a is the microscopic cross-section for absorption and $\sigma_t = \sigma_s + \sigma_a$.

Equation 5 suggests that in a slab of material, if a neutron undergoes a scattering process, the probability P of finding it at a distance x from its original position is

$$P(x) = e^{-\frac{x}{\lambda}}, \tag{8}$$

where λ is the mean free path of neutron in the material.

To find an exponential distribution of the form of Equation 8 that gives the possible step lengths of a neutron after a scattering process, the method of inverse distribution can be used so that a random step length is be given by

$$s_i = \lambda \log(u_i), \tag{9}$$

where u_i are random numbers distributed uniformly between 0 and 1.

If the scattering processes are isotropic, uniformly distributed displacement vectors of the scattered neutrons described by spherical polar coordinates length r, ϕ and θ can be found by using the sampling rule. This gives

$$r = s_i, \phi = 2\pi u_i, cos\theta = -u_i, \theta = arccos(cos\theta). \tag{10}$$

3 Method

3.1 Preliminary Preparation for Simulation

An exponential random number generator (ERNG) whose generated distribution is that of Equation 8 was first constructed by making use of Equation 9.

To verify the correctness of the ERNG, 1000 random lengths were generated using the ERNG with an input λ of 45 cm and a histogram of the generated numbers binned into 50 bins of uniform width was plotted. The expected frequencies N_i of each bin is given by

	Water	Lead	Graphite
Density / gcm^{-3}	1.00	11.35	1.67
σ_a / barn	0.6652	0.158	0.0045
σ_s / barn	103.0	11.221	4.74
λ / cm	0.29	2.67	2.52
p_a	0.006	0.014	0.001
p_s	0.994	0.986	0.999

Table 1: Table showing the characteristic quantities for water, lead and graphite.

$$N_i = N_0 \int_x^{x+\Delta x} P(x) = N_0 \int_x^{x+\Delta x} e^{-\frac{x}{\lambda}} = N_0 P_i,$$

where $N_0 = 1000$ is the total frequency of generated lengths, x are the numbers generated, Δx is the width of the bins and P_i is the probability corresponding to each individual bin. Linearising this gives

$$x = \lambda ln N_0 - \lambda ln N_i$$

so plotting x against lnN_i gives an expected gradient of $-\lambda$. For the ERNG to work correctly, taking the negative of the gradient should be give back approximately 45 cm. The ERNG was run 100 times to give 100 sets of unique distributions each containing 1000 random lengths. 100 graphs of x was plotted against lnN_i to obtain a mean value for λ. This value of λ was compared the original input of 45 cm.

Using the ERNG, an isotropic exponential steps generator (IESG) was constructed using the methods discussed in Equation 10. A graph showing 10000 generated steps using the IESG with a λ of 45 cm was plotted and visually inspected to verify the correctness of the IESG.

3.2 Simulation of Neutrons in Water, Lead and Graphite

Water, lead and graphite slabs of thickness 10 cm and infinite in length and height were used. The cross-sections, mean free paths, probability of scattering and probability of absorption for each material is shown in Table 1.

For each slab of water, lead and graphite, n number of neutrons were fired to be incident normally on the slab. Neutrons that exit the slab through the incident surface are recorded as 'reflected'. Neutrons that exit the slab through the opposite surface are recorded as 'transmitted'. Neutrons that are absorbed within the slab are recorded as 'absorbed'.

The transmission, reflection and absorption fractions were calculated after all n neutrons are simulated. The entire simulation of n neutrons was performed 10 times to obtain mean values for transmission, reflection and absorption fractions and their corresponding errors through the standard deviation.

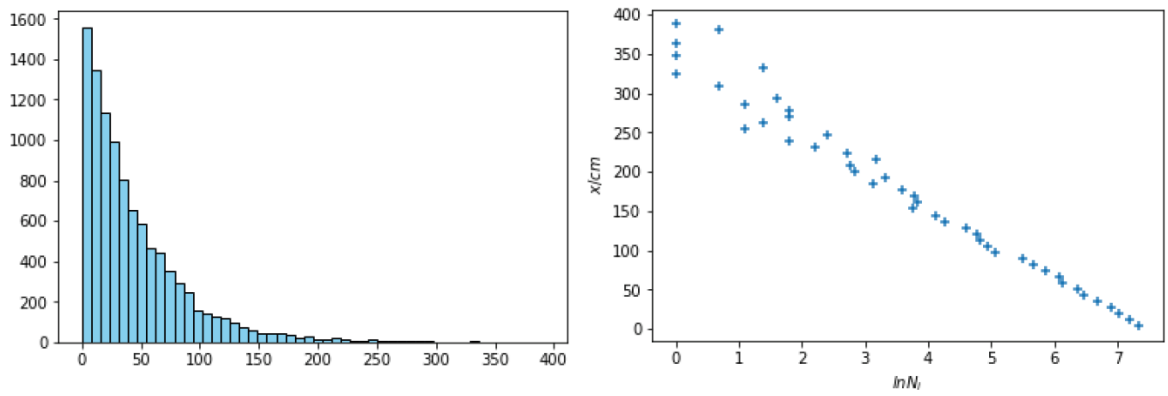

Figure 1: (Left) Histogram of 1000 exponentially distributed random lengths with a mean free path of 45 cm. (Right) Graph of exponentially distributed lengths against the logarithmic of the frequency at each bin.

To investigate how the percentage errors of the fractions vary with n for each material, simulations of n values of 100, 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000 were performed for all three 10 cm slabs of water, lead and graphite. Graphs of percentage errors of the transmission, reflection and absorption fractions were plotted against n to observe the relationship. An optimum n value that minimizes both the error and simulation time was chosen for each material and used for subsequent simulations.

The variation of the transmission, reflection and absorption fractions with slab thickness was then investigated for water, lead and graphite. The thickness used for water was 1 to 10 cm in increments of 1 cm. The thickness used for lead was 10 to 100 cm in increments of 10 cm. The thickness used for graphite was 10 to 150 cm in increments of 20 cm. Three graphs of fractions against thickness were obtained for water, lead and graphite. Values of characteristic attenuation lengths of neutrons in each material were deduced from these graphs and compared with the expected mean free path values.

4 Results

4.1 Determining Accuracy of ERNG and IESG

One of the 100 exponential distributions generated by the ERNG plotted as a histogram and its corresponding linearised plot of x against lnN_i is shown in Figure 1. It was generated using an input mean free path of 45 cm.

The mean gradient of the 100 linearised graphs were determined to be (-47.58 ± 0.21) cm, giving a calculated mean free path of (47.58 ± 0.21) cm. This disagrees with the expected value of 45 cm. This may be caused by the inaccuracies in assigning an average x value to each bin of the histogram in Figure 1 and using those x values to produce the linearised graph. However, this discrepancy is not too large for the purpose of this experiment. The ERNG is subsequently used in later parts of the experiment.

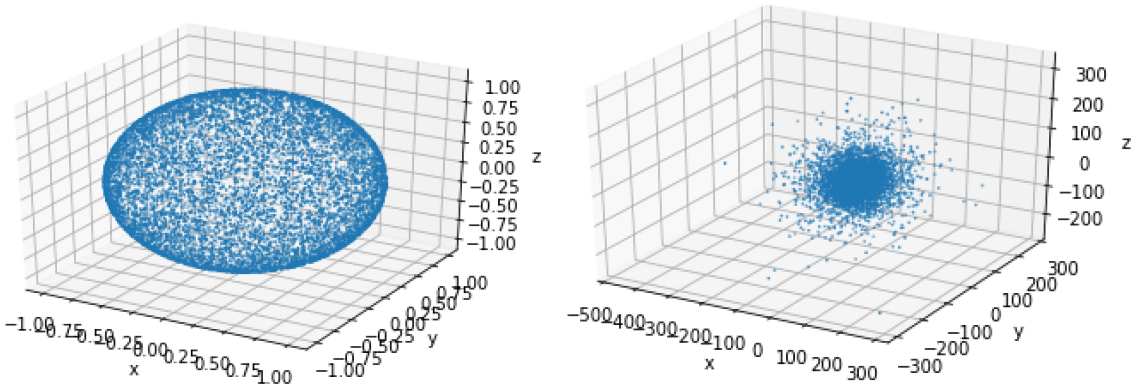

Figure 2: (Left) Isotropic unit vectors displayed as uniform points on a sphere of unit radius 1. (Right) Isotropic displacment vectors with exponentially distributed lengths displayed as collection of points.

Figure 2 shows 10000 points generated by an isotropic unit vector generator on the left and 10000 points generated by the IESG on the right. The isotropic unit vector generator combined with the ERNG makes the IESG. It can be seen that points generated are by the isotropic unit vector generator are indeed uniformly distributed in all directions, as proven by the uniform density of points on the unit sphere. The points generated by the IESG are also seen to be isotropic and their lengths are distributed exponentially as well.

4.2 Simulation of Neutrons in Water, Lead and Graphite

Figure 3 shows a few examples of the transport of neutrons in 10 cm of water, lead and graphite. It can be seen that the transport of neutrons in these three materials is a stochastic process. The neutron undergoes consecutive scattering processes (represented by blue dots) until it is transmitted, reflected or absorbed.

The variation of the percentage errors of transmission, reflection and absorption fractions for 10 cm of water with no. of simulated neutrons n is shown in Figure 4. The percentage errors of the fractions decreased significantly with larger values of n and gradually reached saturation. The point of saturation was around 6000 neutrons. At this point, the transmission, reflection and absorption fractions recorded were $(0.35 \pm 0.03)\%$, $(79.81 \pm 0.12)\%$ and $(19.84 \pm 0.10)\%$ respectively. The transmission fraction was low because water has a mean free path of 0.29 cm that is much smaller than 10 cm. Neutrons are unlikely to reach the opposite slab surface before they encounter an absorption event or are reflected. The neutron number of 6000 was used for subsequent simulations of involving water.

Figure 5 shows the variation of percentage errors of the transmission, reflection and absorption fractions with n for 10 cm of lead. The smallest percentage errors were obtained when 4000 neutrons were used for simulation. The corresponding transmission, reflection and absorption fractions measured were $(27.98 \pm 0.11)\%$, $(61.68 \pm 0.11)\%$ and $(10.34 \pm 0.08)\%$. 10 cm of lead has a much larger transmission fraction than 10 cm of water as the

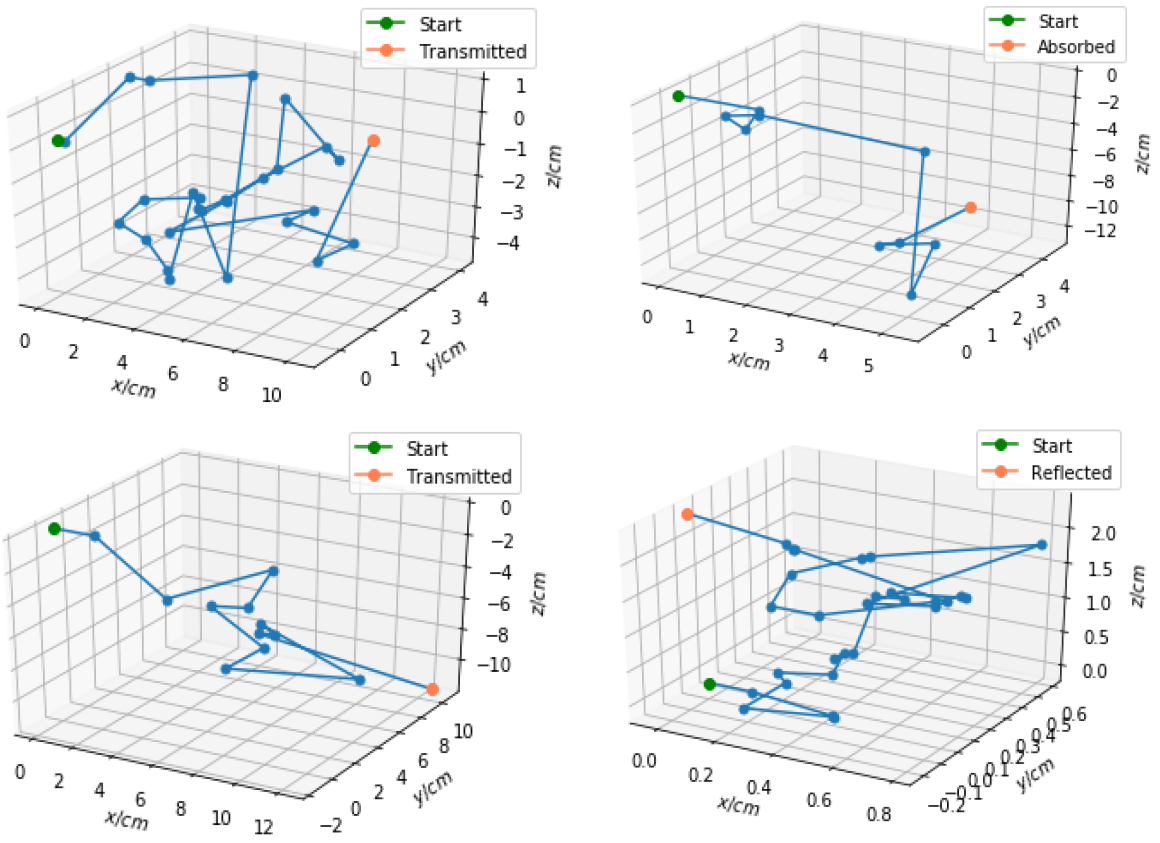

Figure 3: (Top Left) Random walk of a transmitted neutron in 10 cm of graphite. (Top Right) Random walk of an absorbed neutron in 10 cm of lead. (Bottom Left) Random walk of a transmitted neutron in 10 cm of lead. (Bottom Right) Random walk of a reflected neutron in 10 cm of water.

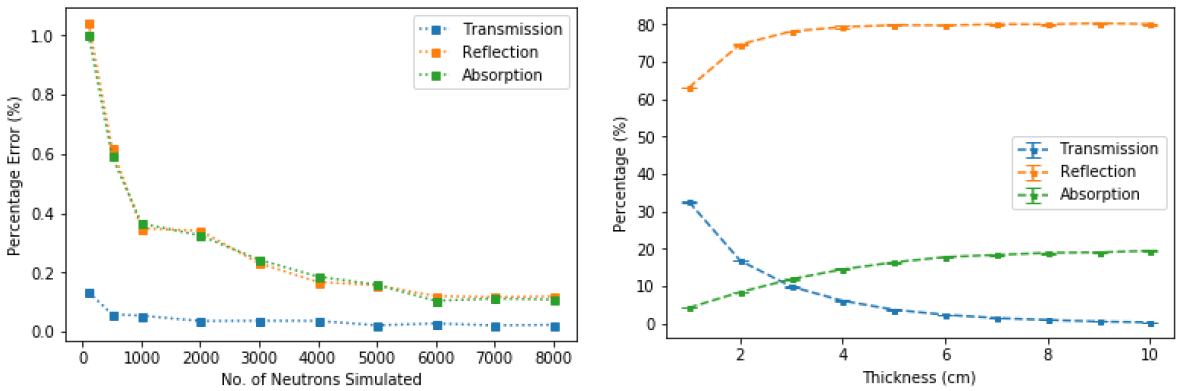

Figure 4: (Left) Graph showing variation of percentage errors of transmission, reflection and absorption fractions with no. of neutrons simulated for 10 cm of water. (Right) Graph showing variation of transmission, reflection and absorption fractions with thickness of water from 1 to 10 cm with error bars.

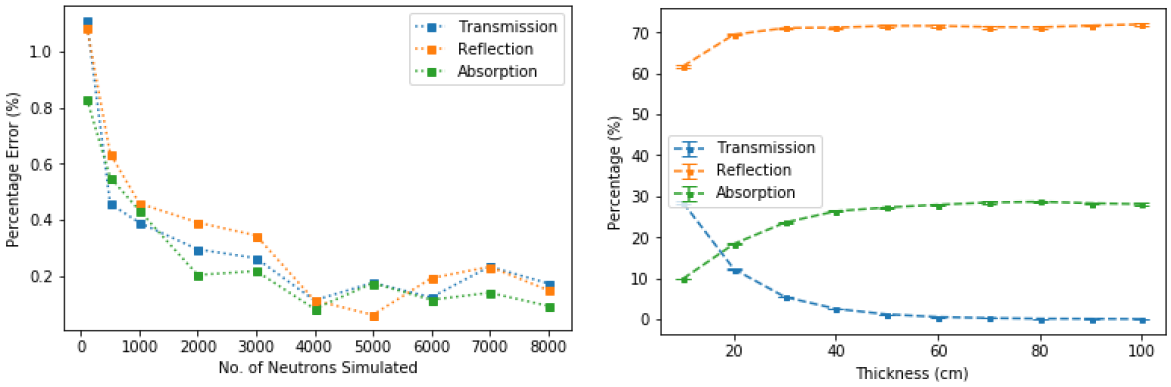

Figure 5: (Left) Graph showing variation of percentage errors of transmission, reflection and absorption fractions with no. of neutrons simulated for 10 cm of lead. (Right) Graph showing variation of transmission, reflection and absorption fractions with thickness of lead from 10 to 100 cm with error bars.

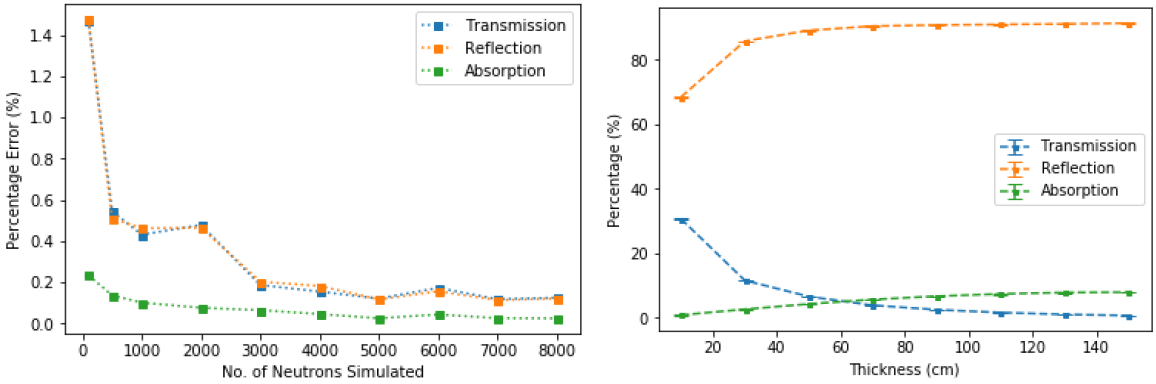

Figure 6: (Left) Graph showing variation of percentage errors of transmission, reflection and absorption fractions with no. of neutrons simulated for 10 cm of graphite. (Right) Graph showing variation of transmission, reflection and absorption fractions with thickness of graphite from 10 to 150 cm with error bars.

mean free path of lead is 2.67 cm and is of comparable size to the thickness of the slab. The neutron number of 4000 was used for subsequent simulations of involving lead.

Figure 6 shows the variation of percentage errors of the transmission, reflection and absorption fractions with n for 10 cm of graphite. Saturation of the errors occurred at around 5000 neutrons simulated and had the smallest errors. The corresponding transmission, reflection and absorption fractions measured were $(30.74 \pm 0.12)\%$, $(68.37 \pm 0.12)\%$ and $(0.89 \pm 0.02)\%$. 10 cm of graphite had the smallest absorption fraction as its probability of absorption of 0.001 is the smallest out of the three materials. The neutron number of 5000 was used for subsequent simulations of involving graphite.

A summary of the transmission, reflection and absorption fractions for 10 cm of water, lead and graphite is given in Table 2.

	10 cm of Water	10 cm of Lead	10 cm of Graphite
Transmission (%)	0.35 ± 0.03	27.98 ± 0.11	30.74 ± 0.12
Reflection (%)	79.81 ± 0.12	61.68 ± 0.11	68.37 ± 0.12
Absorption (%)	19.84 ± 0.10	10.34 ± 0.08	0.89 ± 0.02

Table 2: Table showing the transmission, reflection and absorption fractions for neutrons through 10 cm of water, lead and graphite.

	Water	Lead	Graphite
Gradient	-0.481 ± 0.010	-0.077 ± 0.002	-0.026 ± 0.002
$\boldsymbol{\lambda}$ **(cm)**	2.08 ± 0.04	12.95 ± 0.37	40.00 ± 2.77
$\boldsymbol{\chi^2_{red}}$	0.006	0.03	0.04

Table 3: Table showing the gradients, reduced chi-squred values and calculated attenuation lengths for the graphs in Figure 7.

Figures 4, 5 and 6 also shows the variation of transmission, reflection and absorption fractions with slabs of different thickness for each material. In general the percentage of reflected and absorbed neutrons increased with thickness and the percentage of transmitted neutrons decreased until all three fractions are at saturation. The saturation of the fractions for water, lead and graphite happened after thickness of 8 cm, 60 cm and 110 cm respectively.

Using the transmission fraction values for water, lead and graphite in Figures of 4, 5 and 6 for different thickness, graphs of ln(transmission %) against distance were plotted for each material to obtain the characteristic attenuation length. These are shown in Figure 7. The negative of the gradients in Figure 7 gives the characteristic attenuation length of each material.

The results obtained from the graphs in Figure 7 are presented in Table 3. The characteristic attenuation lengths of water and graphite determined from the graphs were significantly larger than the expected mean free paths of 0.29 cm and 2.52 cm. Both graphs for water and graphite had suscpiciously small χ^2_{red} values of 0.006 and 0.04 respectively. Inspection on the plotted graphs also show the plotted points deviating from the fitted line in a periodic fashion, suggesting that a linear fit is not appropriate. No conclusive results can be drawn for water and graphite.

For lead, plotted points of the graph in Figure 7 shows a good linear fit. However, the χ^2_{red} value obtained for the fit was too small. The characteristic attenuation length was also very large. Thus no conclusive results can be drawn for lead.

5 Conclusion

The transmission, reflection and absorption fractions for 10 cm slabs of water, lead and graphite were measured in this experiment, accurate to errors smaller than 0.13%. The

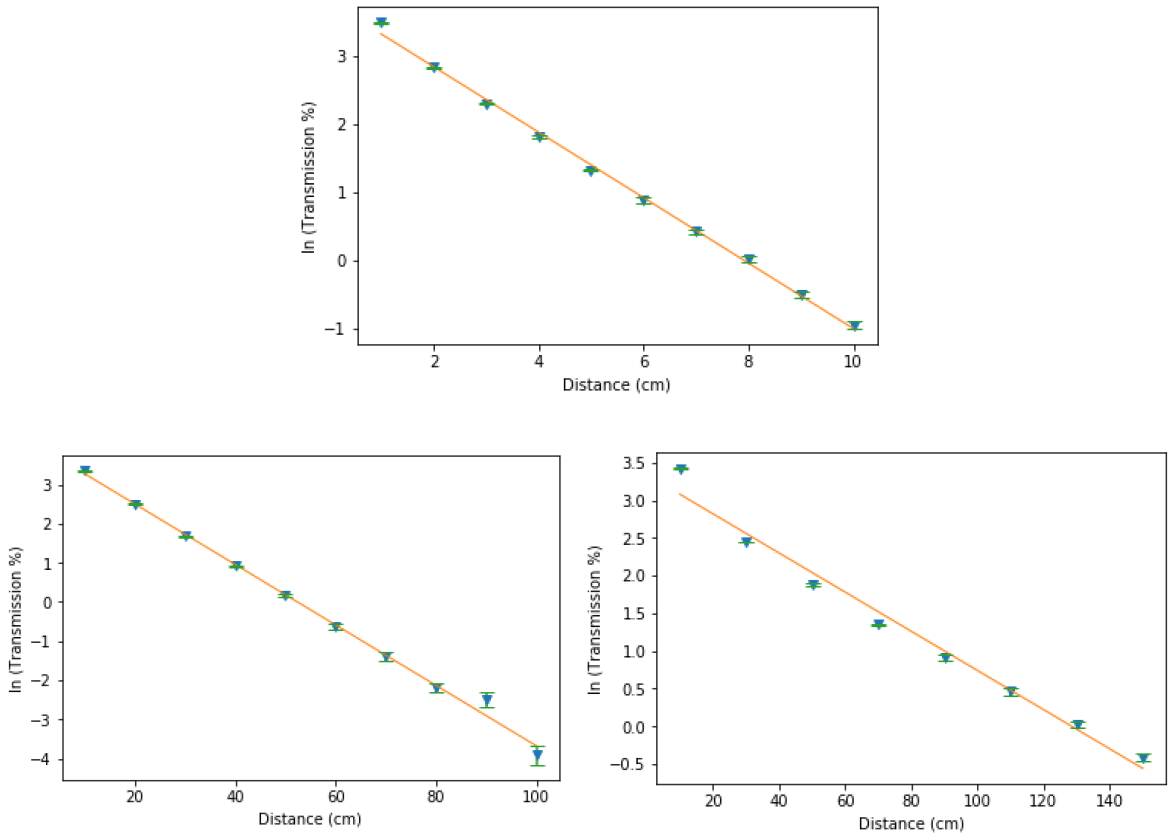

Figure 7: (Top) Graph of ln(transmission %) against distance for water from 1 to 10 cm. (Bottom Left) Graph of ln(transmission %) against distance for lead from 10 to 100 cm. (Bottom Right) Graph of ln(transmission %) against distance for graphite from 10 to 150 cm.

largest transmission fraction was observed for graphite at $(30.74 \pm 0.12)\%$. Water had the smallest transmission fraction of $(0.35 \pm 0.03)\%$ due to it having the shortest mean free path. Water also had the largest absorption fraction of $(19.84 \pm 0.10)\%$, again due to its short mean free path. Neutrons in 10cm of water are more likely to spend more time in it and thus result in greater chances of being absorbed. All three slabs were observed to have reflection fractions of over 60%.

Calculations of the attenuation lengths for water, lead and graphite were significantly longer than the corresponding mean free paths and no conclusive results were drawn.

References

[1] Alonso, M. and Finn, E. (1967). Fundamental university physics. Reading, Mass.: Addison-Wesley Pub. Co.

[2] Nuclear Power. (2018). Microscopic Cross-section - Nuclear Power. [online] Available at: https://www.nuclear-power.net/nuclear-power/reactor-physics/nuclear-engineering-fundamentals/neutron-nuclear-reactions/microscopic-cross-section/ [Accessed 17 May 2018].

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math


#================================
# Uniform Random Number Generator
#================================
s = np.random.uniform(size=60000) # size divisible by 2 and 3

# show uniform frequency of s
plt.hist(s,bins=50,color='skyblue',ec='black')
plt.ylabel('$Frequency$')
plt.xlabel('$x$')
plt.show()

# group values in s into groups of twos to get (x,y) coordinates
x = s[0::2]
y = s[1::2]
plt.scatter(x,y,s=0.2)
plt.ylabel('y')
plt.xlabel('x')
plt.show()

# group values in s into groups of threes to get (x,y,z) coordinates
x = s[0::3]
y = s[1::3]
z = s[2::3]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x,y,z,s=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()

#================================================
# Function of exponential random number generator
#================================================
def randomExpo(n,lumda):
    # generates n numbers distributed according to exponential function
    # e^(-x/lumda)
    # returns array of n numbers
    i = np.random.uniform(size=n)
    s = -lumda*np.log(i)
    return s

# code to test the function
s = randomExpo(1000,45)
(n, bins, patches) = plt.hist(s,bins=50,color='skyblue',ec='black')
plt.ylabel('Frequency')
plt.xlabel('x / cm')
plt.show()

# code to test the function more rigorously
```

```
# by verifying that characteristic attenuation length in water is about
45cm
# if there is no scattering (only absorption) and mean free path input is
45cm

# array to store calculated attenuation length values
attenLengthValues = np.array([])

# run 100 times to get a mean value for attenution length and its estimated
error
for x in range(100):
    s = randomExpo(10000,45)
    (n, bins, patches) = plt.hist(s,bins=50,color='skyblue',ec='black')
    plt.show()

    # store values of wanted counts and middle x-value of wanted bins
    n_new = np.array([])
    x_new = np.array([])

    # discard bins with zero counts and take middle x-value of bins
    for i in range(len(n)):
        if n[i] != 0:
            n_new = np.append(n_new, n[i])
            x_value = (bins[i] + bins[i+1]) / 2
            x_new = np.append(x_new, x_value)

    # plot x_new against ln(n_new)
    (m, c) = np.polyfit(np.log(n_new), x_new, 1)
    plt.scatter(np.log(n_new), x_new, marker='+')
    plt.ylabel('$x / cm$')
    plt.xlabel('$ln N(x)$')
    plt.show()

    attenLengthValues = np.append(attenLengthValues, -m)

# calculate mean, standard deviation, and standard error of mean for
attenLengthValues
std = np.std(attenLengthValues)
mean = np.mean(attenLengthValues)
mean_error = std/math.sqrt(100)

#=====================================================
# Function that generates uniformly distributed points
# on the surface of unit sphere
#=====================================================
def randomSpherePoints(n):
    # generates n points on a unit sphere of radius 1
    # returns tuple of (x_array,y_array,z_array) coordinates of each point
    r = 1
    phi = np.random.uniform(0,2*math.pi,size=n)
    costheta = np.random.uniform(-1,1,size=n)
    theta = np.arccos( costheta )

    x = r * np.sin( theta) * np.cos( phi )
    y = r * np.sin( theta) * np.sin( phi )
    z = r * np.cos( theta )

    return (x,y,z)

# code to test the function
(x,y,z) = randomSpherePoints(10000)
```

```
fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.scatter(x,y,z,s=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()

#=========================================================
# Function that generates 3D isotropic steps distributed
# exponentially with a mean free path lumda
#========================================== ==============
def randomIsoSteps(n,lumda):
    # generates n 3D isotropic steps with exponentially distributed length
    # and a mean free path of lumda
    # returns tuple of (x_array,y_array,z_array) for the steps
    r = randomExpo(n,lumda)
    phi = np.random.uniform(0,2*math.pi,size=n)
    costheta = np.random.uniform(-1,1,size=n)
    theta = np.arccos( costheta )

    x = r * np.sin( theta) * np.cos( phi )
    y = r * np.sin( theta) * np.sin( phi )
    z = r * np.cos( theta )

    return (x,y,z)

# code to test the function
(x,y,z) = randomIsoSteps(10000,45)
fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.scatter(x,y,z,s=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()

#=========================================================================
# Function that simulates firing neutrons at a medium of finite thickness
#=========================================================================
# medium of finite thickness with existing between x=0 and x=L and
# infinite length in y and z

def fireNeutrons(n,L,mfp,aProb,sProb):
    # simulate firing 'n' neutrons at a medium of thickness 'L' in cm with
mean free path 'mfp' in cm,
    # abosrption probabilty 'aProb' in decimal and scattering probability
'sProb' in decimal
    # returns (no. of neutrons transmitted, no. of neutrons reflected, no.
of neutrons absorbed)

    aCount = 0    # parameter to count number of neutrons absorbed
    rCount = 0    # parameter to count number of neutrons reflected
    tCount = 0    # parameter to count number of neutrons transmitted

    while n > 0:
        # create arrays to store position of neutron and give initial
position of (0,0,0)
        (xPos,yPos,zPos) = (np.array([0]),np.array([0]),np.array([0]))
```

```
        # create temporary variables to store temporary x,y,z component
values of each step
        (xVect,yVect,zVect) = (0,0,0)

        aCheck = 0      # parameter to check whether absorption has happened
        rCheck = 0      # parameter to check whether reflection has happened
        tCheck = 0      # parameter to check if transmitted
        i = 0           # parameter to count number of steps
        dice = 0        # parameter to determine absorbed or scattered after
every step
        result = ''     # parameter to announce result of the neutron in the
end

        while aCheck == 0 and rCheck == 0 and tCheck == 0 :
            if i == 0:
                # first step always travels in x direction
                (xVect,yVect,zVect) = (randomExpo(1,mfp),0,0)
                (xPos, yPos, zPos) =
(np.append(xPos,xVect),np.append(yPos,yVect),np.append(zPos,zVect))
                i += 1
            else:
                (xVect,yVect,zVect) = randomIsoSteps(1,mfp)
                (xPos,yPos,zPos) =
(np.append(xPos,xPos[i]+xVect),np.append(yPos,yPos[i]+yVect),np.append(zPos
,zPos[i]+zVect))
                i += 1

            if xPos[i] < 0:
                # check if reflected
                rCheck = 1

            if xPos[i] > L:
                # check if transmitted
                tCheck = 1

            if rCheck == 0 and tCheck == 0:
                # if not reflected and not transmitted
                dice = np.random.uniform()    # 'roll a dice'
                if dice <= aProb:
                    # check if aborbed
                    aCheck = 1

        # record final result of this neutron
        if rCheck == 1:
            rCount += 1
            result = 'Reflected'
        if tCheck == 1:
            tCount += 1
            result = 'Transmitted'
        if aCheck == 1:
            aCount += 1
            result = 'Absorbed'

        n -= 1    # done with 1 neutron simultaion

        # plot an example 3D random walk history of the last neutron fired
        if n == 1:
            fig = plt.figure()
            ax = fig.add_subplot(111,projection='3d')
            ax.plot(xPos[:1],yPos[:1],zPos[:1],'o-
',c='green',label="Start",zorder=2)
```

```
            ax.plot(xPos[-1:],yPos[-1:],zPos[-1:],'o-
',c='coral',label=result,zorder=2)
            ax.plot(xPos,yPos,zPos,'o-',markersize=5,zorder=1)
            ax.set_xlabel('$x / cm$')
            ax.set_ylabel('$y / cm$')
            ax.set_zlabel('$z / cm$')
            ax.legend()
            plt.show()

    # optional code to print results
    print('No. of neutrons fired',tCount+rCount+aCount)
    print('No. transmitted', tCount)
    print('No. reflected', rCount)
    print('No. absorbed', aCount)

    return (tCount,rCount,aCount)

#=============================================================================
============
# Function that calculates percentage of transmission/reflection/absorption
with errors
#=============================================================================
============
def runSimulations(n,runs,L,mfp,aProb,sProb):
    # runs fireNeutrons() simulaion 'runs' times to get percentage of
transmission/reflection/absorption
    # with corresponding percentage errors
    # returns and print the calculated values
    tResults = np.array([])     # array to store transmission number of all
runs
    rResults = np.array([])     # array to store reflection number of all
runs
    aResults = np.array([])     # array to store absorption number of all
runs

    runsCount = 0      # parameter to count no. of runs elapsed

    while runsCount < runs:
        # fire the neutrons
        (tCount,rCount,aCount) = fireNeutrons(n,L,mfp,aProb,sProb)
        # record the results
        (tResults,rResults,aResults) =
(np.append(tResults,tCount),np.append(rResults,rCount),np.append(aResults,a
Count))
        # done with 1 run
        runsCount += 1

    # calculate mean
    (tMean,rMean,aMean) =
(np.mean(tResults),np.mean(rResults),np.mean(aResults))
    # calculate standard deviation
    (tStd,rStd,aStd) = (np.std(tResults),np.std(rResults),np.std(aResults))
    # calculate standard error of mean
    (tMeanErr,rMeanErr,aMeanErr) = (tStd/math.sqrt(runs-
1),rStd/math.sqrt(runs-1),aStd/math.sqrt(runs-1))
    # calculate percentage of transmission/reflection/absorption
    (tPercent,rPercent,aPercent) = (tMean/n*100,rMean/n*100,aMean/n*100)
    # calculate percentage errors of transmission/reflection/absorption
    (tPercentErr,rPercentErr,aPercentErr) =
(tMeanErr/n*100,rMeanErr/n*100,aMeanErr/n*100)
```

```
    print('Transmission %.2f +- %.2f %.2f +- %.2f' %
(tMean,tMeanErr,tPercent,tPercentErr),'%')
    print('Reflection %.2f +- %.2f %.2f +- %.2f' %
(rMean,rMeanErr,rPercent,rPercentErr),'%')
    print('Absorption %.2f +- %.2f %.2f +- %.2f' %
(aMean,aMeanErr,aPercent,aPercentErr),'%')
    return (
[tMean,tMeanErr,tPercent,tPercentErr],[rMean,rMeanErr,rPercent,rPercentErr]
, [aMean,aMeanErr,aPercent,aPercentErr])

#=====================================================
# Simulations For Medium of Water (Thickness 10cm )
#=====================================================
# vary no. of neutrons simulated to see the variation of errors

# array to store the n values to use
nValues = np.array([100,500,1000,2000,3000,4000,5000,6000,7000,8000])
# arrays to store transmission/reflection/absorption data collected
(tDataWater,rDataWater,aDataWater) = ([],[],[])

# simulate for all the nValues
for i in nValues:
    data =
runSimulations(n=i,runs=10,L=10,mfp=0.29,aProb=0.006,sProb=0.994)
    tDataWater.append(data[0])
    rDataWater.append(data[1])
    aDataWater.append(data[2])

# convert into numpy arrays
(tDataWater,rDataWater,aDataWater) =
(np.array(tDataWater),np.array(rDataWater),np.array(aDataWater))

plt.errorbar(nValues,tDataWater[:,3],label='Transmission',linestyle='dotted
',marker='s',markersize=5)
plt.errorbar(nValues,rDataWater[:,3],label='Reflection',linestyle='dotted',
marker='s',markersize=5)
plt.errorbar(nValues,aDataWater[:,3],label='Absorption',linestyle='dotted',
marker='s',markersize=5)
plt.xlabel('No. of Neutrons Simulated')
plt.ylabel('Percentage Error (%)')
plt.legend()

#=========================================================
# Simulations for Medium of Medium of Lead (Thickness 10cm)
#=========================================================
# vary no. of neutrons simulated to see the variation of errors

# array to store the n values to use
nValues = np.array([100,500,1000,2000,3000,4000,5000,6000,7000,8000])
# arrays to store transmission/reflection/absorption data collected
(tDataLead,rDataLead,aDataLead) = ([],[],[])

# simulate for all the nValues
for i in nValues:
    data =
runSimulations(n=i,runs=10,L=10,mfp=2.67,aProb=0.014,sProb=0.986)
    tDataLead.append(data[0])
    rDataLead.append(data[1])
    aDataLead.append(data[2])

# convert into numpy arrays
```

```
(tDataLead,rDataLead,aDataLead) =
(np.array(tDataLead),np.array(rDataLead),np.array(aDataLead))

plt.errorbar(nValues,tDataLead[:,3],label='Transmission',linestyle='dotted'
,marker='s',markersize=5)
plt.errorbar(nValues,rDataLead[:,3],label='Reflection',linestyle='dotted',m
arker='s',markersize=5)
plt.errorbar(nValues,aDataLead[:,3],label='Absorption',linestyle='dotted',m
arker='s',markersize=5)
plt.xlabel('No. of Neutrons Simulated')
plt.ylabel('Percentage Error (%)')
plt.legend()
plt.show()

#============================================================
# Simulations for Medium of Medium of Graphite (Thickness 10cm)
#============================================================
# vary no. of neutrons simulated to see the variation of errors

# array to store the n values to use
nValues = np.array([100,500,1000,2000,3000,4000,5000,6000,7000,8000])
# arrays to store transmission/reflection/absorption data collected
(tDataGraphite,rDataGraphite,aDataGraphite) = ([],[],[])

# simulate for all the nValues
for i in nValues:
    data =
runSimulations(n=i,runs=10,L=10,mfp=2.52,aProb=0.001,sProb=0.999)
    tDataGraphite.append(data[0])
    rDataGraphite.append(data[1])
    aDataGraphite.append(data[2])

# convert into numpy arrays
(tDataGraphite,rDataGraphite,aDataGraphite) =
(np.array(tDataGraphite),np.array(rDataGraphite),np.array(aDataGraphite))

plt.errorbar(nValues,tDataGraphite[:,3],label='Transmission',linestyle='dot
ted',marker='s',markersize=5)
plt.errorbar(nValues,rDataGraphite[:,3],label='Reflection',linestyle='dotte
d',marker='s',markersize=5)
plt.errorbar(nValues,aDataGraphite[:,3],label='Absorption',linestyle='dotte
d',marker='s',markersize=5)
plt.xlabel('No. of Neutrons Simulated')
plt.ylabel('Percentage Error (%)')
plt.legend()
plt.show()

#=====================================================
# Simulations for Water of Varying Thickness (1-10cm)
#=====================================================
# fire 6000 neutrons, run 10 times for each thickness to get t,r,a values

# array to store the thickness values of water to use (1-10cm)
thickWater = np.array([1,2,3,4,5,6,7,8,9,10])

# arrays to store transmission/reflection/absorption values calculated
(tValuesWater,rValuesWater,aValuesWater) = ([],[],[])

# simulate for all the thickWater values
for i in thickWater:
```

```
    data =
runSimulations(n=6000,runs=10,L=i,mfp=0.29,aProb=0.006,sProb=0.994)
    tValuesWater.append(data[0])
    rValuesWater.append(data[1])
    aValuesWater.append(data[2])

# convert into numpy arrays
(tValuesWater,rValuesWater,aValuesWater) =
(np.array(tValuesWater),np.array(rValuesWater),np.array(aValuesWater))

plt.errorbar(thickWater,tValuesWater[:,2],yerr=tValuesWater[:,3],label='Tra
nsmission',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickWater,rValuesWater[:,2],yerr=rValuesWater[:,3],label='Ref
lection',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickWater,aValuesWater[:,2],yerr=aValuesWater[:,3],label='Abs
orption',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.legend()
plt.xlabel('Thickness (cm)')
plt.ylabel('Percentage (%)')


#=====================================================
# Simulations for Lead of Varying Thickness (10-100cm)
#=====================================================
# fire 4000 neutrons, run 10 times for each thickness to get t,r,a values

# array to store the thickness values of lead to use (10-100cm)
thickLead = np.array([10,20,30,40,50,60,70,80,90,100])

# arrays to store transmission/reflection/absorption values calculated
(tValuesLead,rValuesLead,aValuesLead) = ([],[],[])

# simulate for all the thickLead values
for i in thickLead:
    data =
runSimulations(n=4000,runs=10,L=i,mfp=2.67,aProb=0.014,sProb=0.986)
    tValuesLead.append(data[0])
    rValuesLead.append(data[1])
    aValuesLead.append(data[2])

# convert into numpy arrays
(tValuesLead,rValuesLead,aValuesLead) =
(np.array(tValuesLead),np.array(rValuesLead),np.array(aValuesLead))

plt.errorbar(thickLead,tValuesLead[:,2],yerr=tValuesLead[:,3],label='Transm
ission',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickLead,rValuesLead[:,2],yerr=rValuesLead[:,3],label='Reflec
tion',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickLead,aValuesLead[:,2],yerr=aValuesLead[:,3],label='Absorp
tion',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.legend()
plt.xlabel('Thickness (cm)')
plt.ylabel('Percentage (%)')

#=====================================================
# Simulations for Graphite of Varying Thickness (10-150)
#=====================================================
# fire 5000 neutrons, run 10 times for each thickness to get t,r,a values

# array to store the thickness values of graphite to use (10-150cm)
thickGraphite = np.array([10,30,50,70,90,110,130,150])
```

```
# arrays to store transmission/reflection/absorption values calculated
(tValuesGraphite,rValuesGraphite,aValuesGraphite) = ([],[],[])

# simulate for all the thickGraphite values
for i in thickGraphite:
    data =
runSimulations(n=5000,runs=10,L=i,mfp=2.52,aProb=0.001,sProb=0.999)
    tValuesGraphite.append(data[0])
    rValuesGraphite.append(data[1])
    aValuesGraphite.append(data[2])

# convert into numpy arrays
(tValuesGraphite,rValuesGraphite,aValuesGraphite) =
(np.array(tValuesGraphite),np.array(rValuesGraphite),np.array(aValuesGraphi
te))

plt.errorbar(thickGraphite,tValuesGraphite[:,2],yerr=tValuesGraphite[:,3],l
abel='Transmission',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickGraphite,rValuesGraphite[:,2],yerr=rValuesGraphite[:,3],l
abel='Reflection',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.errorbar(thickGraphite,aValuesGraphite[:,2],yerr=aValuesGraphite[:,3],l
abel='Absorption',capsize=5,linestyle='dashed',marker='s',markersize=3)
plt.legend()
plt.xlabel('Thickness (cm)')
plt.ylabel('Percentage (%)')

#================================================
# Find Characteristic Attenuation Length for Water
#================================================
# plot ln(transmission percentage) against thickness
# gradient is -1/lumda where lumda is attenuation length
x = thickWater
y = np.log(tValuesWater[:,2])
yErr = tValuesWater[:,3] / tValuesWater[:,2]

# fit linear relationship
(p, V) = np.polyfit(x,y,1,cov=True)
(m, c) = (p[0],p[1])        # m is gradient, c is intercept
(mErr, cErr) = (np.sqrt(V[0][0]),np.sqrt(V[1][1]))

# calculate chi-sq
chiSq = np.sum((np.polyval(p,x)-y)**2)
chiSqRed = chiSq / (len(x)-2)

# plot and print results
plt.plot(x,y,marker='v',linestyle='none')
plt.plot(x,np.polyval(p,x),linewidth=1)
plt.errorbar(x,y,yerr=yErr,fmt='none',capsize=5)
plt.ylabel('ln (Transmission %)')
plt.xlabel('Distance (cm)')
plt.show()

print('Chi Sq Reduced',chiSqRed)
print('Gradient',m,'+-',mErr)

#================================================
# Find Characteristic Attenuation Length for Lead
#================================================
# plot ln(transmission percentage) against thickness
# gradient is -1/lumda where lumda is attenuation length
```

```
x = thickLead
y = np.log(tValuesLead[:,2])
yErr = tValuesLead[:,3] / tValuesLead[:,2]

# fit linear relationship
(p, V) = np.polyfit(x,y,1,cov=True)
(m, c) = (p[0],p[1])        # m is gradient, c is intercept
(mErr, cErr) = (np.sqrt(V[0][0]),np.sqrt(V[1][1]))

# calculate chi-sq
chiSq = np.sum((np.polyval(p,x)-y)**2)
chiSqRed = chiSq / (len(x)-2)

# plot and print results
plt.plot(x,y,marker='v',linestyle='none')
plt.plot(x,np.polyval(p,x),linewidth=1)
plt.errorbar(x,y,yerr=yErr,fmt='none',capsize=5)
plt.ylabel('ln (Transmission %)')
plt.xlabel('Distance (cm)')
plt.show()

print('Chi Sq Reduced',chiSqRed)
print('Gradient',m,'+-',mErr)

#=====================================================
# Find Characteristic Attenuation Length for Graphite
#=====================================================
# plot ln(transmission percentage) against thickness
# gradient is -1/lumda where lumda is attenuation length
x = thickGraphite
y = np.log(tValuesGraphite[:,2])
yErr = tValuesGraphite[:,3] / tValuesGraphite[:,2]

# fit linear relationship
(p, V) = np.polyfit(x,y,1,cov=True)
(m, c) = (p[0],p[1])        # m is gradient, c is intercept
(mErr, cErr) = (np.sqrt(V[0][0]),np.sqrt(V[1][1]))

# calculate chi-sq
chiSq = np.sum((np.polyval(p,x)-y)**2)
chiSqRed = chiSq / (len(x)-2)

# plot and print results
plt.plot(x,y,marker='v',linestyle='none')
plt.plot(x,np.polyval(p,x),linewidth=1)
plt.errorbar(x,y,yerr=yErr,fmt='none',capsize=5)
plt.ylabel('ln (Transmission %)')
plt.xlabel('Distance (cm)')
plt.show()

print('Chi Sq Reduced',chiSqRed)
print('Gradient',m,'+-',mErr)
```