Penetration of Neutrons Through Shielding

Parampreet Singh

10137114

School of Physics and Astronomy

The University of Manchester

Project 3: Monte Carlo Techniques

May 2019

Abstract

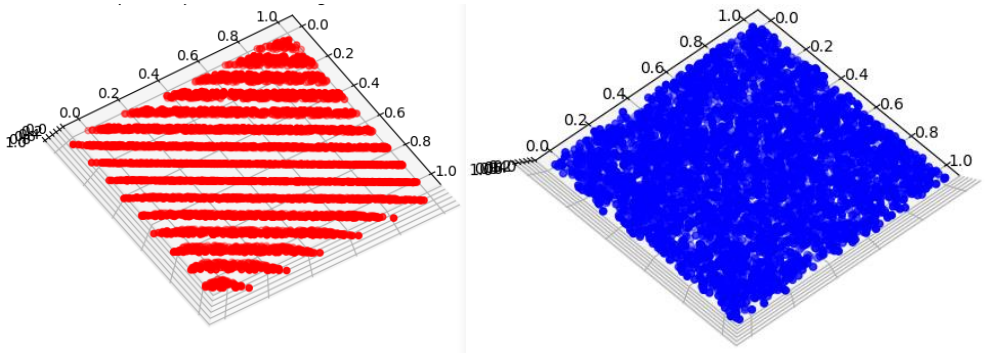
In this report Monte Carlo techniques are discussed and used to determine the fraction of neutrons transmitted in three different slabs of thickness 10 cm. The slabs were assumed to be infinitely wide and only thermal neutrons were considered. The percentage of neutrons transmitted in the slabs were found to be (0.335 ± 0.04), (28.025 ± 0.35) and (30.325 ± 0.40) % for water, lead and graphite respectively. The attenuation length for each slab was also calculated from the linear fits and was found to be (2.0991 ± 0.1495) cm for water, (10.9768 ± 0.0456) cm for lead and (15.6083 ± 0.1258) cm for graphite.

1. Introduction

Integration is a method used to find the area of a function. Monte Carlo techniques is a form of integration where random samples are used to find an approximation; the utilization of increasing sample numbers gives a more accurate approximation. More precisely Monte Carlo techniques can determine the value of an unknown quantity using the principle of inferential statistics. For example, to compute the area of a circle using Monte Carlo techniques one would place the circle in a square and then generate random points within the square, if a point was inside the circle it would return true, else it would return false. As the number of points increases the number of true and false values being returned would increase and allow an approximation for the area of the circle to be calculated. If the process is repeated several times the error on the approximation can be found by taking the standard deviation of all approximations [1].

In this report Monte Carlo techniques were used to infer the percentage of neutrons transmitted through slabs of water, lead and graphite. The effect of varying neutrons and the thickness of the slab on the percentage of neutrons transmitted was also observed. The respective attenuation lengths of each slab were also calculated and compared to their respective mean free path.

2. Theory



(a) *(b)*

Figure 1: *Spectral test on Linear Congruential Generator (a) and Python in-built number generator (b).*

Monte Carlo techniques rely on samples being truly random. Figure 1 shows the spectral test on two different methods for generating random numbers, method one which is shown on the left is a Linear Congruential Generator (LCG) and method two uses Pythons in-built random number generator. In 2D both graphs would look to show true statistical randomness, however in 3D LCG's (see Figure 1a) create points which lie on hyper planes. This method would not generate true random samples and hence method two was then used to create random uniformly distributed samples.

In order to produce a neutron penetration simulation, numbers needed to be randomly generated from a distribution according to an exponential function Eq. (1). To do this, cumulative distribution sampling was used. Cumulative distribution sampling requires an initial probability density function (PDF) (see Eq. (1)), which is then integrated to form the cumulative distribution function (CDF). The CDF is then inverted to form the inverse CDF and then multiplied by a uniformly distributed variable to get a distribution distributed according to the original PDF (see Eq. (2)). In this case random uniformly distributed numbers between zero and one were inserted into Eq. (2) to obtain the step length of each neutron (randomly generated numbers from a distribution according to an exponential function).

$$P[x] = e^{\frac{-x}{\lambda}} \tag{1}$$

$$s_i = -\lambda \log(u_i) \tag{2}$$

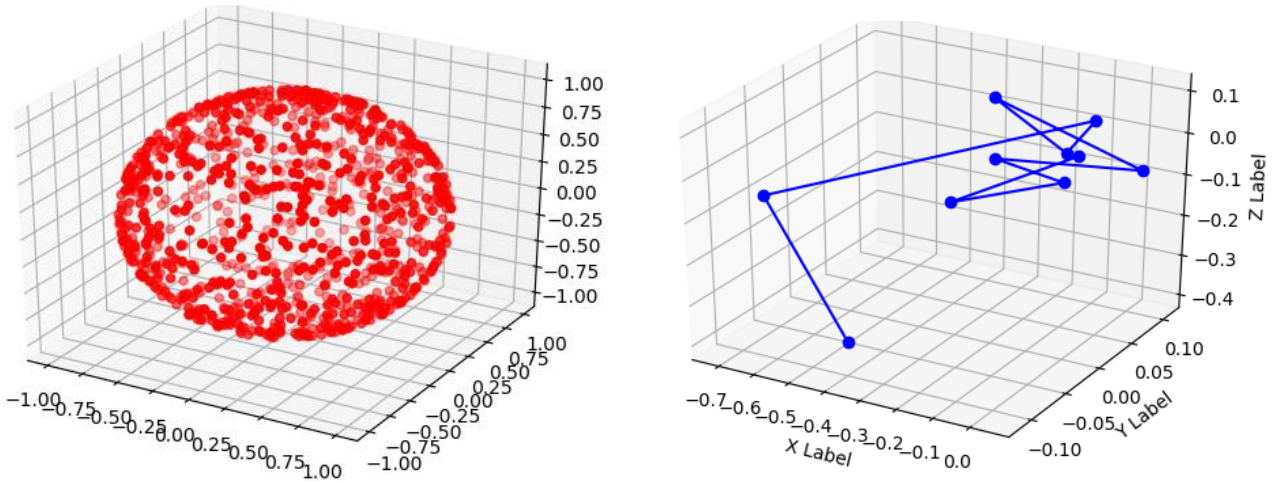
In Eq. (1) and (2) λ is the mean free path of the material and x and u_i are random uniformly distributed numbers between zero and one [2].

To simulate the random movement of neutrons within the slab isotropic steps with lengths distributed according to the exponential function needed to be generated. To do this, isotropic unit vectors were generated by finding random uniformly distributed numbers between negative pi and pi for theta and zero and one for phi. To ensure there was uniform distribution over the sphere and no concentration of points near the poles, the random uniformly distributed points were transformed using Eq. (3) to find values for phi [3].

$$\phi = \cos(1 - 2u_i) \tag{3}$$

To convert these unit vectors from spherical polar coordinates to cartesian coordinates Eq. (4) [4] was used. Simulations of the sphere to obtain unit vectors and the resulting isotropic steps can be seen in Figure 2.

$$x = \sin(\phi)\cos(\theta) \tag{4}$$
$$y = \sin(\phi)\sin(\theta)$$
$$z = \cos(\theta)$$



(a) *(b)*

Figure 2: *Graph(a) is an example of random uniformly distributed numbers on a sphere. Graph(b) is an example of ten isotropic steps with lengths distributed to an exponential function.*

To calculate the attenuation length which is essentially the mean free path of a material, the number of absorbing molecules needed to be calculated. This can be done using Eq. (5), where ρ is density and *M* is molar mass of the material.

$$n = \frac{\rho N_A}{M} \tag{5}$$

Mean free path is then given by Eq. (6), where σ is the absorption cross-section [5].

$$\lambda = \frac{1}{n\sigma} \tag{6}$$

3. Experimental Method

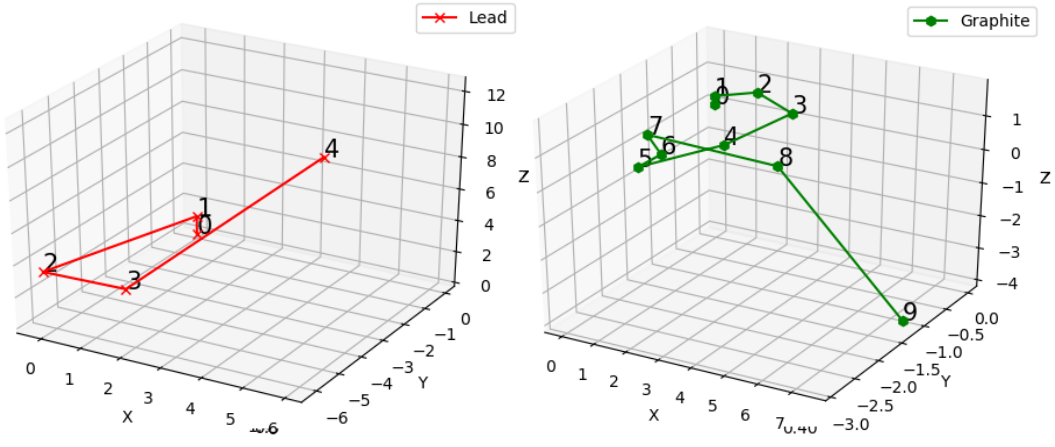
The materials chosen for this simulation were water, lead and graphite and each slab had a thickness of ten centimetres (cm). Initially, all neutrons were assumed to move perpendicular to the slab to guarantee initial penetration, this way no neutrons would be initially discarded due to reflection. This meant that each neutron initially only moved in one direction whilst all other positions were zero, in this case x and y were initially zero and penetration occurred in the z-direction. This first step was then calculated using Eq. (2) and was not multiplied by an isotropic unit vector.

Once the neutron had penetrated, movement in three dimensions could now be considered, thus another exponentially distributed step length was calculated and multiplied by its corresponding isotropic unit vector, this ensured the step now had magnitude and direction. This step could either cause the neutron to leave the slab from where it came from, resulting in reflection or leave from the other side of the slab meaning the neutron was transmitted. If the neutron left the slab it was assumed to be 'dead', and hence its movement was no longer tracked under the assumption it didn't re-enter the slab. The slabs were assumed to be infinite in the x and y direction and hence only movement in the z-direction needed to be considered when deciding if the neutron had been reflected, transmitted or absorbed.

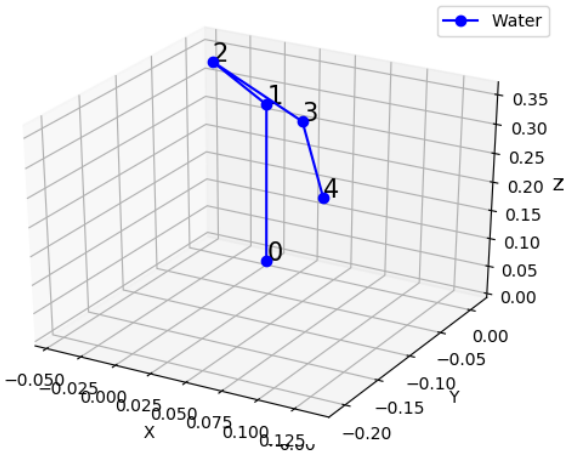
If the neutron was neither reflected or transmitted it meant that it was still present within the slab, which then implies it was either scattered or absorbed. In the latter case the neutron would again be assumed to be 'dead' and therefore no longer be tracked. To decide whether the neutron was absorbed or scattered the ratio of macroscopic cross-sections for absorption and scattering were calculated. However, in this simulation the number of absorbing molecules were assumed to be the same as the number of scattering molecules, therefore reducing this ratio down to the ratio of their respective cross-sections.

Once this limit was found, a random uniformly distributed number between zero and one was found. If this number was then lower than the cross-section ratio it was assumed that the neutron was absorbed. If this number was however greater than the ratio it was assumed to be scattered and would subsequently take another step. This cycle was then repeated until the neutron was either absorbed, transmitted or reflected.

Examples of these step cycles can be seen in Figure 3, all neutrons initially start at a position of (0, 0, 0). The final z-position for water is within in the slab, this means it must have been absorbed (see Figure 3c). The final z-position for the lead neutron is past 10 (the angle of the picture isn't representative of the position) and as the slabs were only 10 cm thick, this must have meant that this neutron had been transmitted (see Figure 3a). The final z-position for graphite is below 0 and hence implies that it was reflected out of the slab (see Figure 3b).



(a) *(b)*

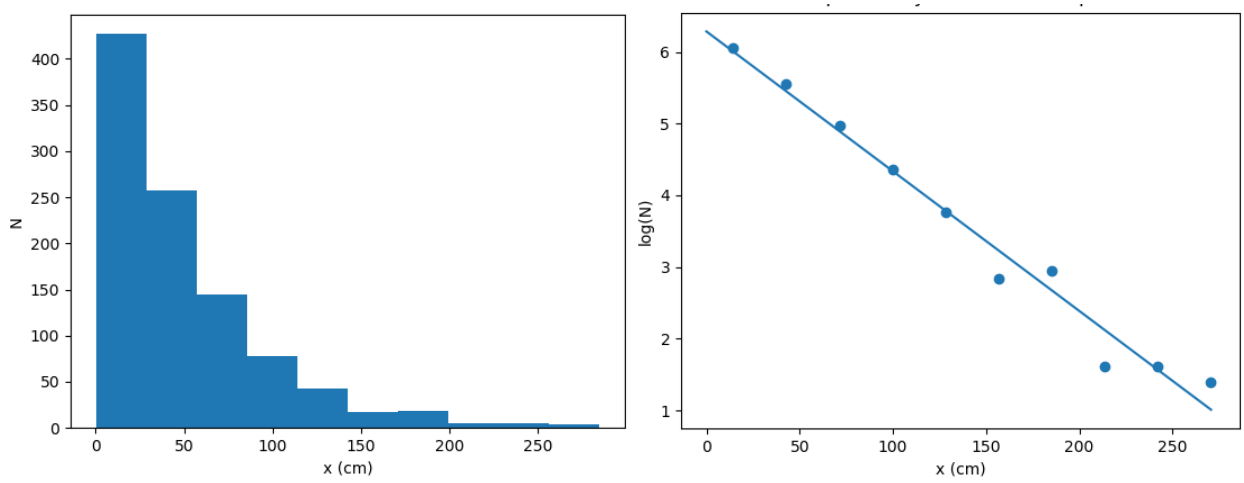


(c)

Figure 3: *Simulated random walks for the three slabs of thickness 10 cm*

Figure 4a shows the distribution of random uniformly generated numbers according to the exponential function. The mean free path was set to 45 cm and scattering was assumed to be negligible (including reflection). Logging Eq. (1) we can form a new equation in the form of a straight line, $y = mx + c$, and hence plot a straight line as seen in Figure 4b. This straight line will now have a gradient equal to the negative reciprocal of the mean free path.

Finding the gradient and therefore finding the attenuation length of this graph gave a value of 44.59 cm, close to the original value of the mean free path. The error on the attenuation length can be calculated by taking the standard deviation of multiple attenuation lengths, to see how much one deviates from the other. It was found in this case that the attenuation length was (44.59 ± 0.56) cm. The true answer of 45 cm therefore lies within one standard deviation of the estimated value.



(a) *(b)*

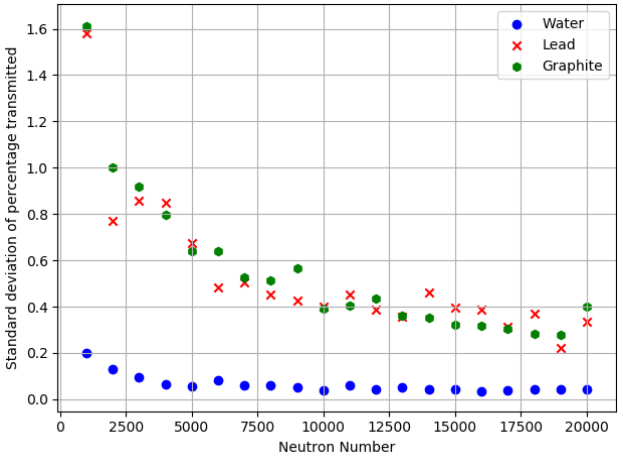
Figure 4: *Graph (a) shows the random depth of neutrons in water. Graph (b) shows the log of the number of neutrons against the depth, to determine the attenuation length.*

	Water	Lead	Graphite
Absorbing cross-section (barn)	0.6652	0.1580	0.0045
Scattering cross-section (barn)	103.0	11.221	14.74
Density (g/cm^3)	1.00	11.35	1.67
Molar mass (g/mol)	18.015	207.2	12.0107
Absorbing limit	0.00641	0.0139	0.000948
Mean free path (cm) (Experimental)	0.289	2.66	2.52

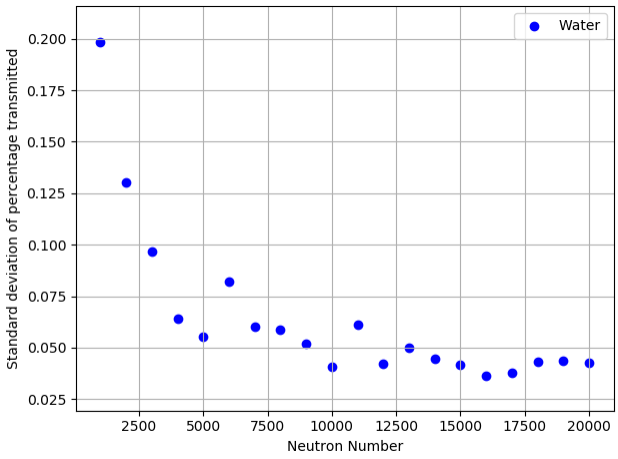
Table 1: *Parameters of water, lead and graphite [6]*

As seen in Table 1 water has the greatest absorption and scattering cross-section. Whereas the mean free path is greatest for lead and lowest for water. Similarly, the absorbing limit is greatest for lead and lowest for graphite.

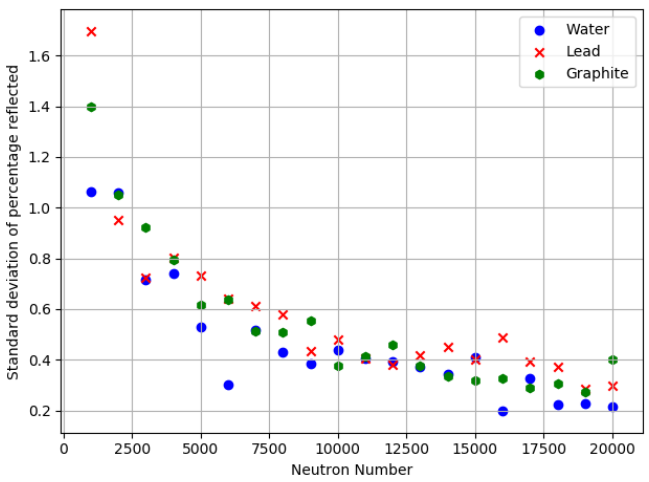
4. Results



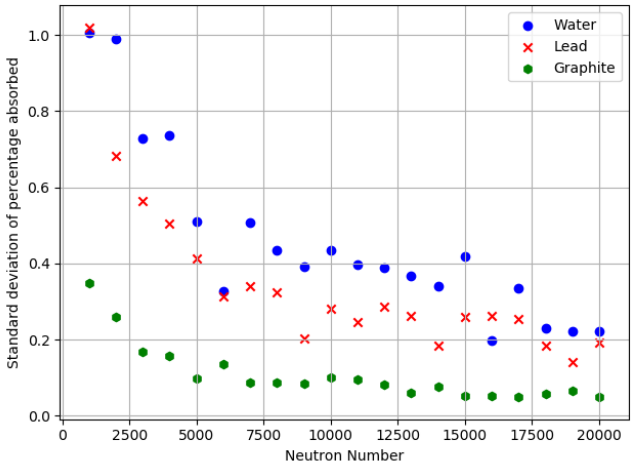
(a)



(b)



(c)



(d)

Figure 5: *Percentage error by standard deviation for water, lead and graphite against neutron number for transmission (a) and (b), reflection (c) and absorption (d) derived from 20 repeats.*

In order to calculate the percentage of neutrons transmitted in the slabs, the error associated with neutron number needed to be calculated. All the graphs in Figure 5 show there is a clear trend of weak exponential decay as the neutron number is increasing. In Figure 5a there is a great percentage drop for lead and graphite compared to water, therefore water has been plotted on its own in Figure 5b to show its weak exponential decay more clearly. For Figure 5a all slabs have a percentage error below 1% at around 2500 neutrons, this drops to half its value of 0.5% at around 10,000 neutrons for lead and graphite. Beyond 12,500 neutrons it can be seen that the percentage error for lead and graphite tends to around 0.3%. Although water looks like it has hit a steady point at around 7500 neutrons in Figure 5a, it is clear when looking at Figure 5b that there is still a steady drop in percentage error past this point.

Figure 5c shows that lead has the greatest drop in percentage error compared to the other slabs, it drops from a value of around 1.7% to 0.3%. Unlike for transmission all slabs seem to converge to a similar percentage error of 0.3%. Figure 5d shows a distinct difference in percentage error for graphite compared to lead and water. Lead and water start at a percentage error of around 1.0% and drop to a value of around 0.2%, whereas graphite starts at around 0.4% and drops to around 0.05%

	Water	**Lead**	**Graphite**
Thickness (cm)	10	10	10
Total Neutrons	20,000	20,000	20,000
Neutrons Reflected	15,972	12,421	13,782
Neutrons Absorbed	3961	1974	153
Neutrons Transmitted	67	5605	6065
Percentage Reflected (%)	79.86 ± 0.21	62.105 ± 0.30	68.91 ± 0.40
Percentage Absorbed (%)	19.805 ± 0.22	9.87 ± 0.19	0.765 ±0.05
Percentage Transmitted (%)	0.335 ± 0.04	28.025 ± 0.35	30.325 ± 0.40

Table 2: *Number of neutrons absorbed, transmitted and reflected at a thickness 10 cm for 20,000 neutrons*

From Figure 5 it was clear that increasing neutron number would decrease the variation in results and hence give a lower standard deviation resulting in more precise results. Therefore 20,000 neutrons were chosen to be tested in the simulation. From Table 2 it is seen that graphite had the greatest proportion of neutrons transmitted which was closely followed by lead, whilst water had the greatest number of neutrons absorbed and reflected.

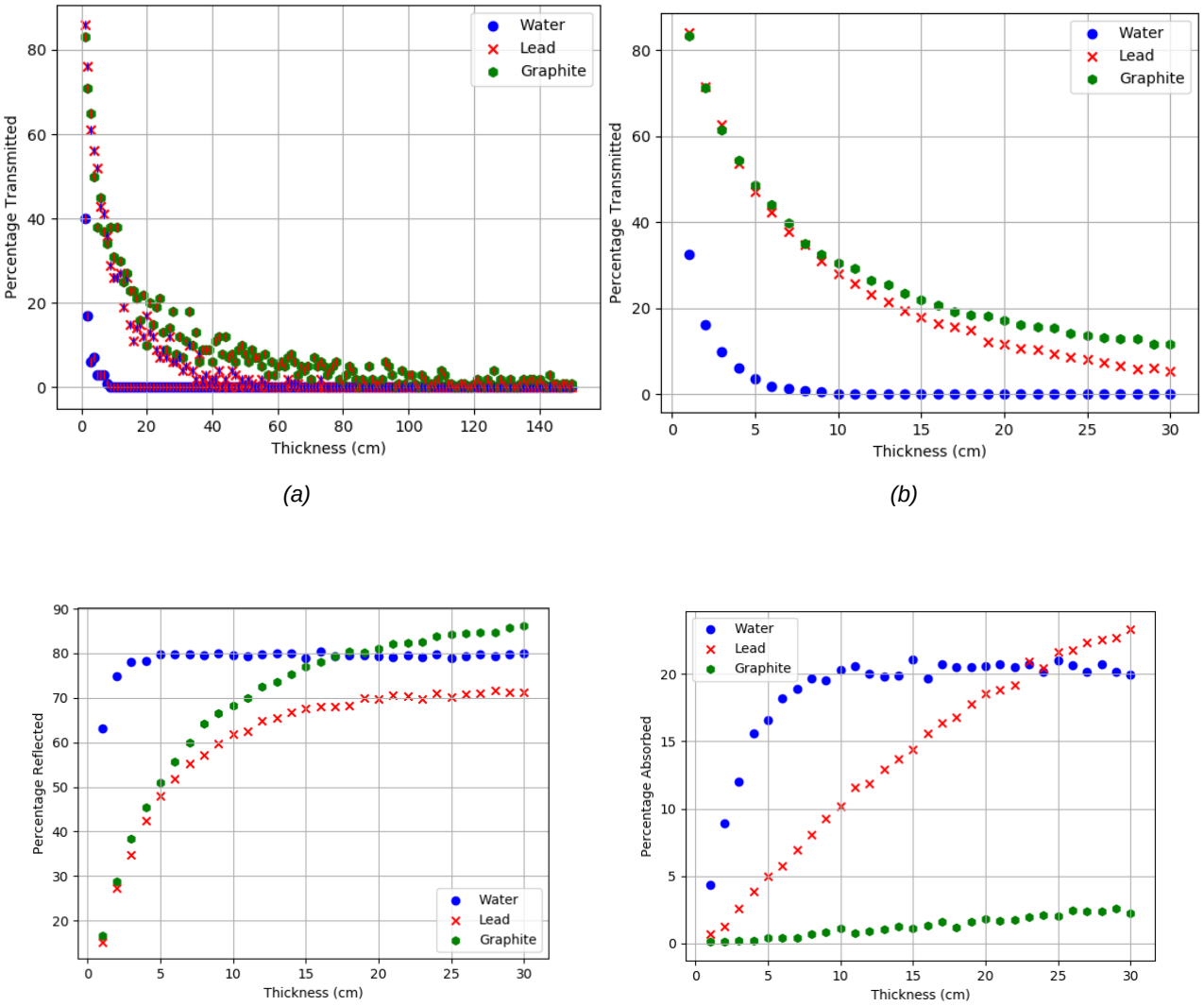
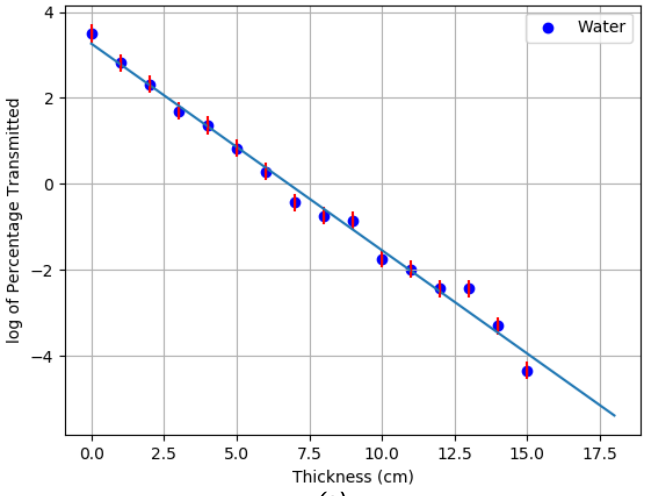


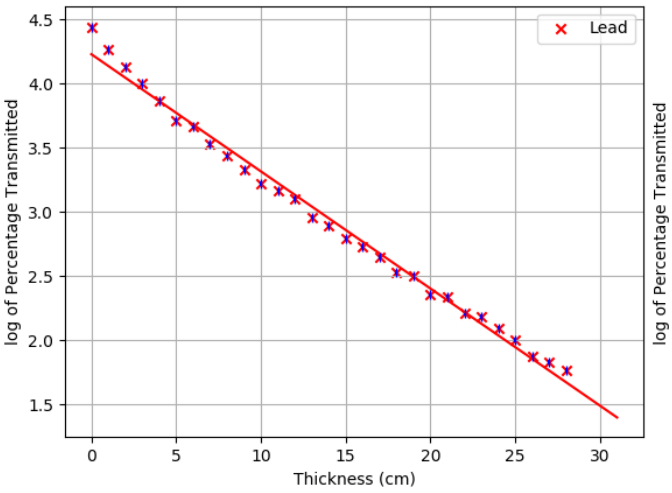
Figure 6: *Preliminary test to find thickness at which there is no transmission for 100 neutrons (a) and percentage transmitted, reflected and absorbed against thickness of 30 cm for 8000 neutrons (b), (c), and (d)*

To understand how thickness affected transmission of neutrons a preliminary test was initially simulated to see at what thickness no transmission would be seen for all slabs. Figure 6a summarises the results for each slab, the general trend of the graph is again of weak exponential decay which is to be expected. There is approximately no transmission for water at around 10 cm whereas lead starts to converge to zero transmission at around 40 cm, graphite however begins to converge at around 100 cm. Figure 5b is coherent with Figure 5a, showing a clearer and more precise weak exponential decay for percentage transmitted against thickness. It can be seen that initial percentage transmitted is much greater for graphite and lead compared to water.

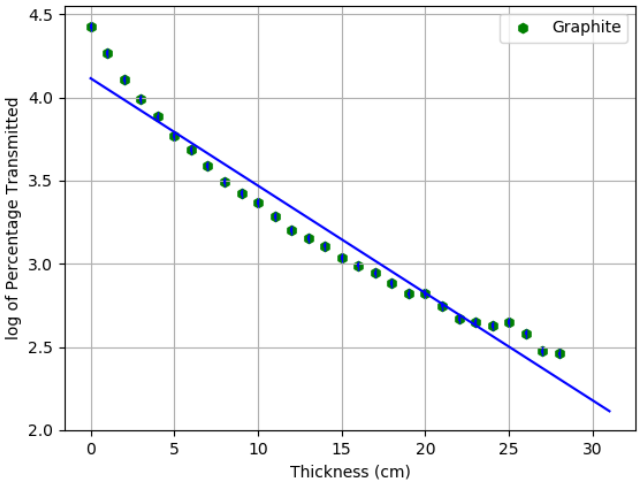
Figure 5c shows that refection saturation occurs at much smaller thickness for water at around 5cm. This saturation can also be seen for lead at a thickness of around 20 cm. Saturation for graphite must occur at a thickness greater than 30 cm, as at around 30 cm the percentage of neutrons reflected is still increasing. From Figure 5d absorption is much more likely to occur for water than it is for graphite and lead, as there is a great difference in percentage absorbed for the same thicknesses. However, water seems to have a percentage absorbed saturation point of around 10 cm, whereas lead continues to increase at a slower initial rate and eventually has a greater percentage of absorption past 25cm.



(a)



(b)



(c)

Figure 7: *Linear fit of attenuation lengths for 8000 neutrons over 10 repeats (a), (b) and (c).*

Figures 7a, 7b and 7c are plots of the linear fit to calculate the attenuation length. The error bars in Figure 7a, 7b and 7c are very small as 8000 neutrons were utilized which gave percentage errors of about 0.2, 0.02 and 0.01 % for water, lead and graphite respectively. These values differ from those in Figure 5 as the error on the logged values is the standard deviation of the logged values and not the log of the error. It can be seen in Figure 6b that the max thickness for water is around 15 cm compared to the other slabs which go up to around 28 cm.

	Water	Lead	Graphite
Attenuation length (cm)	2.0991 ± 0.1495	10.9768 ± 0.0456	15.6083 ± 0.1258
R^2	0.961	0.991	0.958

Table 3: *Attenuation lengths of water, lead and graphite*

Table 3 shows the calculated values for the attenuation length for each slab, comparing these values to the mean free paths in Table 1 it is seen that the value for water is out from its true value by an order of magnitude of about one. Whereas lead and graphite are about six times larger than their expected value. The error on the attenuation length was calculated by repeating the simulation ten times and then computing the standard deviation, as a result the attenuation length did not differ by much and has a low error. The R^2 value is highest for lead and lowest for graphite however, all R^2 are relatively high (above 95 %) this implies that the models are a good fit for the data. Looking at Figure 7c it seems that a curve would fit the model better rather than a straight line as it appears to show weak exponential decay, this suggests that the R^2 value does not entirely suggest a good fit for the data.

5. Discussion and Conclusion

From Table 2 it can be concluded that most neutrons for all three slabs were reflected as the percentage reflected are all above 60%. Water experienced the most reflection (79.86 ± 0.21%), which was to be expected as it has the lowest mean free path (see Table 1), indicating that it will go through the most collisions compared to the other materials and therefore more likely to be scattered resulting in reflection.

Water has the lowest mean free path due to it having the greatest scattering cross-section. Arising from water being a molecule comprised of oxygen and hydrogen atoms and thus the large number of hydrogen atoms in water creates a greater scattering cross-section.

Although lead has a greater absorbing limit implying theoretically it should have the greatest proportion of neutrons absorbed due to the probability of absorption being greater, water has the largest amount of absorption (19.805 ± 0.22%). This is again due to water having the greatest absorption cross section and thus having a greater initial percentage absorbed. This is supported by Figure 6d as water has a greater initial gradient. However, if this simulation was to be repeated over a greater range of thicknesses, lead could be seen to have the largest amount of absorption. This is because water will reach a saturation point for percentage absorbed first.

Graphite has the greatest percentage of transmission (30.325 ± 0.40%), this is to be expected as graphite has the second largest mean free path of 2.52 cm and a very low absorption cross section compared to the other slabs and a relatively large scattering cross section. This indicates that neutrons are likely to take around four steps within a 10 cm slab and every interaction is over 3000 times more likely to be scattering compared to absorption. This is coherent with the results as graphite has the lowest percentage of absorption of around 0.8%, this is further supported by Figure 6a and 6b which shows that graphite has the greatest transmission over the same thicknesses.

However, it should be noted that this preliminary test was done with 100 neutrons. This was since the simulation would be time-consuming to do all the simulations with a vast number of neutrons. Similarly, when calculating the attenuation length 8000 neutrons were used instead of the previous 20,000. This was again due to the simulation taking too long.

The attenuation lengths for all three slabs are greater than their respective mean free paths, this is reasonable as we have made several assumptions in the simulation, such as once a neutron is reflected out of the slab it does not re-enter. This in-turn reduces the effective number of neutrons in the finite slab. It was also assumed that the slab was infinite in the x and y plane, which subsequently reduced the number of neutrons which were able to be transmitted. It should also be noted that thicknesses which gave a transmission of zero were discarded as logging these values would return a value of infinity. As a result, this would also affect the linear fit by partially varying the gradient and hence the attenuation length.

This simulation could be improved by creating a finite slab to allow transmission in all directions, and track neutrons which leave the slab to see if they re-enter. In this simulation only thermal neutrons were considered, if repeated other neutrons could be included. Finally, there could be a greater variation in neutron number and thickness and more materials could be tested.

References

[1] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/lecture-slides-and-files/MIT6_0002F16_lec6.pdf, *Monte Carlo Simulation*, MIT open course, (07/0519)

[2] Downey, Allen B., *Think Stats*, O'Reilly Media, Inc., First Edition, 2011, 23-33

[3] http://corysimon.github.io/articles/uniformdistn-on-sphere/, *Generating uniformly distributed numbers on a sphere*, Cory Simon, (29/04/19)

[4] Riley, K. F., Hobson, M. P., Bence, S. J., *Mathematical Methods for Physics and Engineering*, Cambridge University Press, Third Edition, 2006, 357-367

[5] http://galileo.phys.virginia.edu/classes/304/h2o.pdf, *The Properties of Water*, Virginia education, (30/04/19)

[6] Cottingham, W. N., Greenwood, D. A., *An Introduction to Nuclear Physics*, Cambridge University Press, Second Edition, 2001, 110-130

Appendix

```
"""
Parampreet Singh - 25/04/19
Computational Physics: Project 3
"""

#Import packages
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import linregress

#Functions

#Draga Function for generating random numbers
def randssp(p,q):

    #global m, a, c, x

    try: x
    except NameError:
        m = pow(2, 31)
        a = pow(2, 16) + 3
        c = 0
        x = 123456789

    try: p
    except NameError:
        p = 1
    try: q
    except NameError:
        q = p

    r = np.zeros([p,q])

    for l in range (0, q):
        for k in range (0, p):
            x = np.mod(a*x + c, m)
            r[k, l] = x/m

    return r

#Function to create random uniform numbers in a matrix of (x, y ,z)
def random_uniform_matrix(n):
    value = np.random.uniform(0, 1, n*3) #Random uniform numbers
    value_2 = np.reshape(value, (3, n)) #Turns array into matrix with 3 rows
    x = value_2[0, :] #Takes first row from matrix (x)
    y = value_2[1, :]
    z = value_2[2, :]

    return x, y, z

#Function to create random uniform numbers between 0 and 1
def random_uniform(n):
    value = np.random.uniform(0, 1, n) #Random uniform numbers

    return value

#Function to create random uniform numbers distributed according to exp(-x/lambda) #Step length
def random_exp(n, lambda_): #number of itterations, mean free path
    k_i = np.array(random_uniform(n)) # U(0, 1)
    x_i = -lambda_*np.log(k_i) #Inverse CDF
```

```
    return x_i

#Function to find the number of absorbing molecules
def absorbing_molecules(density, molar_mass):
    n = ((density*avogadro_constant)/molar_mass) #Units cm^-3

    return n

#Function to find mean free path
def mean_free_path(absorbing_molecules, absorption_cross_section, scattering_cross_section):
    lambda_ = 1/((absorbing_molecules*absorption_cross_section) + (absorbing_molecules*scatterin

    return lambda_

#Function to create random uniform numbers between any range
def random_uniform_any(min_, max_, n):
    value = np.random.uniform(min_, max_, n) #Random uniform numbers

    return value

#Function to find absorbing limit
def absorbing_limit(absorption_cross_section, scattering_cross_section):
    u = absorption_cross_section / (absorption_cross_section + scattering_cross_section)

    return u

#Function to generate isotropic unit vectors
def uniform_sphere(n):
    u = random_uniform(n)
    theta = random_uniform_any(-np.pi, np.pi, n) #Randomly distributed numbers between -pi and p
    phi = np.arccos(1 - 2*u) # F−1(u)=arccos(1−2u) #inverse CDF
    x = np.sin(phi) * np.cos(theta) # r = 1 # cartesian to spherical coordinates
    y = np.sin(phi) * np.sin(theta)
    z = np.cos(phi)
    return x, y, z

#Generating isotropic steps with lengths distributed as exp(-x/lambda)
def isotropic_steps(N, lambda_):
    A = random_exp(N, lambda_)
    n = uniform_sphere(N)
    value = A*n
    x = value[0, :] #Takes first row from matrix (x)
    y = value[1, :]
    z = value[2, :]
    return x, y, z

#Function to find number of neutrons absorbed, transmitted and reflected in slab and particle hi
def neutron_path(n, thickness, mean_free_path, absorbing_limit): # n = neutron number
    is_absorbed, is_transmitted, is_reflected= 0, 0, 0 #Tally
    for i in range(0, n):
        first_step = random_exp(1, mean_free_path) #To allow guranteed penetration
        step_counter = 1 #count steps
        absorption = 0
        position_x, position_y, position_z = 0, 0, float(first_step) #Initial position
        particle_history_x, particle_history_y, particle_history_z = [0, 0], [0, 0], [0, positio
        while position_z > 0 and position_z < thickness and absorption == 0: #While in slab
            absorption_probability = random_uniform(1) #Find number from U(0, 1) #Is particle ab
            if absorption_probability < absorbing_limit:
                is_absorbed += 1
                absorption = 1 #Particle is absorbed
```

```
            else:
                step_x, step_y, step_z = isotropic_steps(1, mean_free_path)
                position_x = position_x + step_x
                position_y = position_y + step_y
                position_z = position_z + step_z
                step_counter += 1
                particle_history_x.append(float(position_x))
                particle_history_y.append(float(position_y))
                particle_history_z.append(float(position_z))
                #particle_z.append(position_z)
        if position_z < 0:
            is_reflected += 1 #Particle reflected
        elif position_z > thickness:
            is_transmitted += 1 #Particle transmitted
        percentage_transmitted = round((is_transmitted/n)*100, 3)
        r = round((is_reflected/n)*100, 3) #% reflected
        a = round((is_absorbed/n)*100, 3) #% absorbed
    return is_absorbed, is_transmitted, is_reflected, particle_history_x, particle_history_y, pa

#Function to unpack data from file
def Unpack(Name, column):
    value = np.loadtxt(Name, delimiter = " ", usecols = (column), unpack=True) #Separates data w

    return value


#Main Body

#Constants
absorption_cross_section_water = 0.6652 #Units = barn
absorption_cross_section_lead = 0.158 #Units = barn
absorption_cross_section_graphite = 0.0045 #Units = barn

scattering_cross_section_water = 103.0 #Units = barn
scattering_cross_section_lead = 11.221 #Units = barn
scattering_cross_section_graphite = 4.74 #Units = barn

density_water = 1.00 #Units g/cm^-3
density_lead = 11.35 #Units g/cm^-3
density_graphite = 1.67 #Units g/cm^-3

molar_mass_water = 18.015 #Units g/mol
molar_mass_lead = 207.2 #Units g/mol
molar_mass_graphite = 12.0107 #Units g/mol

barn = 10**(-24) #cm^2
avogadro_constant = 6.022*10**23 #Units mol^-1

#Finding absorbing molecules #Units 1/cm^-3
absorbing_molecules_water = absorbing_molecules(density_water, molar_mass_water)
absorbing_molecules_lead = absorbing_molecules(density_lead, molar_mass_lead)
absorbing_molecules_graphite = absorbing_molecules(density_graphite, molar_mass_graphite)

#Finding mean free path #Units cm
mean_free_path_water = mean_free_path(absorbing_molecules_water, absorption_cross_section_water,
mean_free_path_lead = mean_free_path(absorbing_molecules_lead, absorption_cross_section_lead, sc
mean_free_path_graphite = mean_free_path(absorbing_molecules_graphite, absorption_cross_section_
mean_free_path_water_45 = 45 #cm

#Finding absorbing limit # if u < limit ; then molecule is absorbed
absorbing_limit_water = absorbing_limit(absorption_cross_section_water, scattering_cross_section
```

```
absorbing_limit_lead = absorbing_limit(absorption_cross_section_lead, scattering_cross_section_l
absorbing_limit_graphite = absorbing_limit(absorption_cross_section_graphite, scattering_cross_s

#Error on Neutron number = 20000
water_error = np.std(Unpack("Water transmission.txt", 19))
lead_error = np.std(Unpack("Lead transmission.txt", 19))
graphite_error = np.std(Unpack("Graphite transmission.txt", 19))

#Calculating error for log values
water_l_std = np.std(np.log(Unpack("Water transmission.txt", 7)))
lead_l_std = np.std(np.log(Unpack("Lead transmission.txt", 7)))
graphite_l_std = np.std(np.log(Unpack("Graphite transmission.txt", 7)))

#Variables
neutron_number = 20000
thickness = 10 #Units cm
#Calculating number of neutrons absorbed, transmitted and reflected at constant neutron number a
water_absorbed1, water_transmitted1, water_reflected1, water_particle_history_x, water_particle_
lead_absorbed1, lead_transmitted1, lead_reflected1, lead_particle_history_x, lead_particle_histo
graphite_absorbed1, graphite_transmitted1, graphite_reflected1, graphite_particle_history_x, gra


#DATA FOR CAHNGING NEUTRON NUMBER
################################################################################

repeats = 20 #Number of rows
lower_limit_neutrons = 1000
upper_limit_neutrons = 20000
neutron_interval = 20 #Number of columns
neutron_numbers = np.linspace(lower_limit_neutrons, upper_limit_neutrons, neutron_interval)
water_transmitted = []
lead_transmitted = []
graphite_transmitted = []
water_reflected = []
lead_reflected = []
graphite_reflected = []
water_absorbed = []
lead_absorbed = []
graphite_absorbed = []

#File to store transmission, refledction and absoprtion data

water_filename = "Water transmission.txt"
file_1 = open(water_filename, "w")
lead_filename = "Lead transmission.txt"
file_2 = open(lead_filename, "w")
graphite_filename = "Graphite transmission.txt"
file_3 = open(graphite_filename, "w")

water_filenamer = "Water reflection.txt"
file_4 = open(water_filename, "w")
lead_filenamer = "Lead reflection.txt"
file_5 = open(lead_filename, "w")
graphite_filenamer = "Graphite reflection.txt"
file_6 = open(graphite_filename, "w")

water_filenamea = "Water absorption.txt"
file_7 = open(water_filename, "w")
lead_filenamea = "Lead absorption.txt"
file_8 = open(lead_filename, "w")
graphite_filenamea = "Graphite absorption.txt"
```

```
file_9 = open(graphite_filename, "w")

for i in range(0, repeats): #Iterates over all repeats

    for i in range(0, neutron_interval): #Iterates over all neutron numbers

        #Calculating number of neutrons absorbed, transmitted and reflected with
        #varying neutron number and calculate particle history and percentage transmitted

        water_absorbed1, water_transmitted1, water_reflected1, water_particle_history_x, water_p
        water_transmitted.append(water_percentage_transmitted1)
        water_reflected.append(water_percentage_reflected)
        water_absorbed.append(water_percentage_absorbed)

        lead_absorbed1, lead_transmitted1, lead_reflected1, lead_particle_history_x, lead_partic
        lead_transmitted.append(lead_percentage_transmitted1)
        lead_reflected.append(lead_percentage_reflected)
        lead_absorbed.append(lead_percentage_absorbed)

        graphite_absorbed1, graphite_transmitted1, graphite_reflected1, graphite_particle_histor
        graphite_transmitted.append(graphite_percentage_transmitted1)
        graphite_reflected.append(graphite_percentage_reflected)
        graphite_absorbed.append(graphite_percentage_absorbed)

    #Storing percentage transmitted, reflected and absorbed

    water_transmission_column = np.column_stack(water_transmitted)
    water_reflected_column = np.column_stack(water_reflected)
    water_absorbed_column = np.column_stack(water_absorbed)

    lead_transmission_column = np.column_stack(lead_transmitted)
    lead_reflected_column = np.column_stack(lead_reflected)
    lead_absorbed_column = np.column_stack(lead_absorbed)

    graphite_transmission_column = np.column_stack(graphite_transmitted)
    graphite_reflected_column = np.column_stack(graphite_reflected)
    graphite_absorbed_column = np.column_stack(graphite_absorbed)

#Writing to file

water_transmission_output = np.reshape(water_transmission_column, (repeats, neutron_interval))
np.savetxt(water_filename, water_transmission_output)
file_1.close
lead_transmission_output = np.reshape(lead_transmission_column, (repeats, neutron_interval))
np.savetxt(lead_filename, lead_transmission_output)
file_2.close
graphite_transmission_output = np.reshape(graphite_transmission_column, (repeats, neutron_interv
np.savetxt(graphite_filename, graphite_transmission_output)
file_3.close

water_reflected_output = np.reshape(water_reflected_column, (repeats, neutron_interval))
np.savetxt(water_filenamer, water_reflected_output)
file_4.close
lead_reflected_output = np.reshape(lead_reflected_column, (repeats, neutron_interval))
np.savetxt(lead_filenamer, lead_reflected_output)
file_5.close
graphite_reflected_output = np.reshape(graphite_reflected_column, (repeats, neutron_interval))
np.savetxt(graphite_filenamer, graphite_reflected_output)
file_6.close

water_absorbed_output = np.reshape(water_absorbed_column, (repeats, neutron_interval))
```

```
np.savetxt(water_filenamea, water_absorbed_output)
file_7.close
lead_absorbed_output = np.reshape(lead_absorbed_column, (repeats, neutron_interval))
np.savetxt(lead_filenamea, lead_absorbed_output)
file_8.close
graphite_absorbed_output = np.reshape(graphite_absorbed_column, (repeats, neutron_interval))
np.savetxt(graphite_filenamea, graphite_absorbed_output)
file_9.close


#Plotting std against neutron number
water_std = [] #Transmission
lead_std = []
graphite_std = []
water_stdr = [] #Reflection
lead_stdr = []
graphite_stdr = []
water_stda = [] #Absoroption
lead_stda = []
graphite_stda = []
std_x_value = np.linspace(lower_limit_neutrons, upper_limit_neutrons, neutron_interval)

for i in range(0, neutron_interval):

    #Finding std

    water_1 = np.std(Unpack("Water transmission.txt", i))
    water_std.append(water_1)
    lead_1 = np.std(Unpack("Lead transmission.txt", i))
    lead_std.append(lead_1)
    graphite_1 = np.std(Unpack("Graphite transmission.txt", i))
    graphite_std.append(graphite_1)

    water_2 = np.std(Unpack("Water reflection.txt", i))
    water_stdr.append(water_2)
    lead_2 = np.std(Unpack("Lead reflection.txt", i))
    lead_stdr.append(lead_2)
    graphite_2 = np.std(Unpack("Graphite reflection.txt", i))
    graphite_stdr.append(graphite_2)

    water_3 = np.std(Unpack("Water absorption.txt", i))
    water_stda.append(water_3)
    lead_3 = np.std(Unpack("Lead absorption.txt", i))
    lead_stda.append(lead_3)
    graphite_3 = np.std(Unpack("Graphite absorption.txt", i))
    graphite_stda.append(graphite_3)

#Plotting std against percentage

#Transmission
#All slabs
plt.figure(1)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage transmitted") #20 repeats
plt.scatter(std_x_value, water_std, c="b", marker = "o", label = "Water")
plt.scatter(std_x_value, lead_std, c="r", marker = "x", label = "Lead")
plt.scatter(std_x_value, graphite_std, c="g", marker = "h", label = "Graphite")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron all T")
```

```
#Water
plt.figure(2)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage transmitted") #20 repeats
plt.scatter(std_x_value, water_std, c="b", marker = "o", label = "Water")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron wat T")

#Reflectoin
plt.figure(3)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage reflected") #20 repeats
plt.scatter(std_x_value, water_stdr, c="b", marker = "o", label = "Water")
plt.scatter(std_x_value, lead_stdr, c="r", marker = "x", label = "Lead")
plt.scatter(std_x_value, graphite_stdr, c="g", marker = "h", label = "Graphite")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron all R")
#Water
plt.figure(4)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage reflected") #20 repeats
plt.scatter(std_x_value, water_stdr, c="b", marker = "o", label = "Water")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron wat R")

#Absorption
plt.figure(5)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage absorbed") #20 repeats
plt.scatter(std_x_value, water_stda, c="b", marker = "o", label = "Water")
plt.scatter(std_x_value, lead_stda, c="r", marker = "x", label = "Lead")
plt.scatter(std_x_value, graphite_stda, c="g", marker = "h", label = "Graphite")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron all A")
#Water
plt.figure(6)
plt.xlabel("Neutron Number")
plt.ylabel("Standard deviation of percentage absorbed") #20 repeats
plt.scatter(std_x_value, water_stda, c="b", marker = "o", label = "Water")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("STD vs changing neutron wat A")

##########################################################################################

#The atentuation lengths of the slabs and show how transmission, reflection and absorption vaire
#DATA FOR CHANGING THICKNESS AND ATTENUATION LENGTH

##########################################################################################

max_length = 30
lengths = np.linspace(1, max_length, max_length) #Lengths being tested from 1 to 30 cm
```

```
neutron_number = 8000
water_transmitted_l = [] #Lists to append percentage transmitted
lead_transmitted_l = []
graphite_transmitted_l = []
water_reflected = []
lead_reflected = []
graphite_reflected = []
water_absorbed = []
lead_absorbed = []
graphite_absorbed = []


#10 Repeats were done to find std of attentuation length i.e. the error
for i in range(0, max_length): #Iterate over all lenghts
#Calculating number of neutrons absorbed, transmitted and reflected at constant neutron number
#and particle history and percentage transmitted at varying thickness
    water_absorbed1, water_transmitted1, water_reflected1, water_particle_history_x, water_parti
    water_transmitted_l.append(water_percentage_transmitted1)
    water_reflected.append(water_percentage_reflected)
    water_absorbed.append(water_percentage_absorbed)

    lead_absorbed1, lead_transmitted1, lead_reflected1, lead_particle_history_x, lead_particle_h
    lead_transmitted_l.append(lead_percentage_transmitted1)
    lead_reflected.append(lead_percentage_reflected)
    lead_absorbed.append(lead_percentage_absorbed)

    graphite_absorbed1, graphite_transmitted1, graphite_reflected1, graphite_particle_history_x,
    graphite_transmitted_l.append(graphite_percentage_transmitted1)
    graphite_reflected.append(graphite_percentage_reflected)
    graphite_absorbed.append(graphite_percentage_absorbed)

#Exponential Decay scatter Graphs

#Water
plt.figure(7)
plt.grid()
plt.tight_layout
plt.xlabel("Thickness (cm)")
plt.ylabel("Percentage Transmitted")
plt.scatter(lengths, water_transmitted_l, marker="o", c="b", label="Water")
plt.scatter(lengths, lead_transmitted_l, marker="x", c="r", label="Lead")
plt.scatter(lengths, graphite_transmitted_l, marker="h", c="g", label="Graphite")
#plt.errorbar(lengths, water_transmitted_l, yerr=water_error, c="r", fmt="|")
plt.legend()
plt.savefig("percent transmitted vs length ")

#Lead
plt.figure(8)
plt.grid()
plt.tight_layout
plt.xlabel("Thickness (cm)")
plt.ylabel("Percentage Reflected")
plt.scatter(lengths, water_reflected, marker="o", c="b", label="Water")
plt.scatter(lengths, lead_reflected, marker="x", c="r", label="Lead")
plt.scatter(lengths, graphite_reflected, marker="h", c="g", label="Graphite")
#plt.errorbar(lengths, lead_transmitted_l, yerr=lead_error, c="b", fmt="|")
plt.legend()
plt.savefig("percent redlected vs length ")

#Graphite
plt.figure(9)
```

```
plt.grid()
plt.tight_layout
plt.xlabel("Thickness (cm)")
plt.ylabel("Percentage Absorbed")
plt.scatter(lengths, water_absorbed, marker="o", c="b", label="Water")
plt.scatter(lengths, lead_absorbed, marker="x", c="r", label="Lead")
plt.scatter(lengths, graphite_absorbed, marker="h", c="g", label="Graphite")
#plt.errorbar(lengths, graphite_transmitted_l, yerr=graphite_error, c="r", fmt="|")
plt.legend()
plt.savefig("percent absorbed vs length ")


# Plotting to find attentuation length

#Water
length_w = [] #List to append lengths which have transmission
water_transmitted_l_real = [] #List to append non-zero transmission
for i in range(0, max_length - 1):
    if water_transmitted_l[i] != 0: #Only appends non zero transmission values
        water_transmitted_l_real.append(water_transmitted_l[i])
        length_w.append(i) # coresponding x values
log_water_transmitted_l_real = np.log(water_transmitted_l_real)
#finds gradient and y intercept
coeff_w, cov = np.polyfit(length_w, log_water_transmitted_l_real, 1, cov=True)
aL_w = -1/coeff_w[0] #Atentuation length
x_w = np.array(np.linspace(0, length_w[-1]+3, 30))
y_w = coeff_w[0]*x_w + coeff_w[1] #Line of best fit
#PLotting straight line to find atentuation length
plt.figure(10)
plt.grid()
plt.tight_layout
plt.ylabel("log of Percentage Transmitted")
plt.xlabel("Thickness (cm)")
plt.scatter(length_w, log_water_transmitted_l_real, c="b", marker="o", label="Water")
plt.errorbar(length_w, log_water_transmitted_l_real, yerr = water_l_std,  c="r", fmt="|")
plt.plot(x_w, y_w) #plotting line of best fit
plt.legend()
plt.savefig("transmission vs thickness - water")
r_value_w = linregress(length_w, log_water_transmitted_l_real) #[2]
r_value_w_r = np.square(r_value_w[2]) #r^2 value

#Lead
length_l = []
lead_transmitted_l_real = []
for i in range(0, max_length - 1):
    if lead_transmitted_l[i] != 0:
        lead_transmitted_l_real.append(lead_transmitted_l[i])
        length_l.append(i)
log_lead_transmitted_l_real = np.log(lead_transmitted_l_real)
coeff_l, cov = np.polyfit(length_l, log_lead_transmitted_l_real, 1, cov=True)
aL_l = -1/coeff_l[0]
x_l = np.array(np.linspace(0, length_l[-1]+3, 30))
y_l = coeff_l[0]*x_l + coeff_l[1]
plt.figure(11)
plt.grid()
plt.tight_layout
plt.ylabel("log of Percentage Transmitted")
plt.xlabel("Thickness (cm)")
plt.scatter(length_l, log_lead_transmitted_l_real, c="r", marker="x", label="Lead")
plt.errorbar(length_l, log_lead_transmitted_l_real, yerr = lead_l_std,  c="b", fmt="|")
plt.plot(x_l, y_l, c="r")
```

```
plt.legend()
plt.savefig("transmission vs thickness - lead")
r_value_l = linregress(length_l, log_lead_transmitted_l_real) #[2]
r_value_l_r = np.square(r_value_l[2])

#Graphite
length_g = []
graphite_transmitted_l_real = []
for i in range(0, max_length - 1):
    if graphite_transmitted_l[i] != 0:
        graphite_transmitted_l_real.append(graphite_transmitted_l[i])
        length_g.append(i)
log_graphite_transmitted_l_real = np.log(graphite_transmitted_l_real)
coeff_g, cov = np.polyfit(length_g, log_graphite_transmitted_l_real, 1, cov=True)
aL_g = -1/coeff_g[0]
x_g = np.array(np.linspace(0, length_g[-1]+3, 30))
y_g = coeff_g[0]*x_g + coeff_g[1]
plt.figure(12)
plt.grid()
plt.tight_layout
plt.ylabel("log of Percentage Transmitted")
plt.xlabel("Thickness (cm)")
plt.scatter(length_g, log_graphite_transmitted_l_real, c="g", marker="h", label="Graphite")
plt.errorbar(length_g, log_graphite_transmitted_l_real, yerr = graphite_l_std,  c="b", fmt="|")
plt.plot(x_g, y_g, c="b")
plt.legend()
plt.savefig("transmission vs thickness - graphite")
r_value_g = linregress(length_g, log_graphite_transmitted_l_real) #[2]
r_value_g_r = np.square(r_value_g[2])

##########################################################################################################

#Variables
neutron_number = 20000
thickness = 10 #Units cm

#Calculating number of neutrons absorbed, transmitted and reflected at constant neutron number a
water_absorbed1, water_transmitted1, water_reflected1, water_particle_history_x, water_particle_
lead_absorbed1, lead_transmitted1, lead_reflected1, lead_particle_history_x, lead_particle_histo
graphite_absorbed1, graphite_transmitted1, graphite_reflected1, graphite_particle_history_x, gra

#Printing Results
#Reults will print last simulation that took place
print ("\n------------------------------------------------------------")
print("Transmission Through A Fixed Thickness - Water")
print ("------------------------------------------------------------")
print("|Thickness (cm)             |"+str(thickness))
print("|Total Neutrons             |"+str(neutron_number))
print("|Neutrons Reflected         |"+str(water_reflected1))
print("|Neutrons Absorbed          |"+str(water_absorbed1))
print("|Neutrons Transmitted       |"+str(water_transmitted1))
print("|Percentage Transmitted (%) |"+str(water_percentage_transmitted1))
print("|Percentage Reflected (%)   |"+str(water_percentage_reflected))
print("|Percentage Absorbed (%)    |"+str(water_percentage_absorbed))

print ("\n------------------------------------------------------------")
print("Transmission Through A Fixed Thickness - Lead")
print ("------------------------------------------------------------")
print("|Thickness (cm)             |"+str(thickness))
print("|Total Neutrons             |"+str(neutron_number))
print("|Neutrons Reflected         |"+str(lead_reflected1))
```

```
print("|Neutrons Absorbed          |"+str(lead_absorbed1))
print("|Neutrons Transmitted       |"+str(lead_transmitted1))
print("|Percentage Transmitted (%) |"+str(lead_percentage_transmitted1))
print("|Percentage Reflected (%)   |"+str(lead_percentage_reflected))
print("|Percentage Absorbed (%)    |"+str(lead_percentage_absorbed))

print ("\n-------------------------------------------------------------")
print("Transmission Through A Fixed Thickness - Graphite")
print ("-------------------------------------------------------------")
print("|Thickness (cm)             |"+str(thickness))
print("|Total Neutrons             |"+str(neutron_number))
print("|Neutrons Reflected         |"+str(graphite_reflected1))
print("|Neutrons Absorbed          |"+str(graphite_absorbed1))
print("|Neutrons Transmitted       |"+str(graphite_transmitted1))
print("|Percentage Transmitted (%) |"+str(graphite_percentage_transmitted1))
print("|Percentage Reflected (%)   |"+str(graphite_percentage_reflected))
print("|Percentage Absorbed (%)    |"+str(graphite_percentage_absorbed))


#Plotting graphs for report

"""
#Plotting using Draga random number generator to see if spectral problem present (it should be)#

draga_value = randssp(3, 5000) #points in 3 dimensions (x,y,z)
x1 = draga_value[0, :] #Takes first column from matrix (x)
y1 = draga_value[1, :]
z1 = draga_value[2, :]
fig_1 = plt.figure(13)
ax = fig_1.add_subplot(111, projection = "3d")
ax.scatter(x1, y1, z1, c="r", marker="o")
plt.title("Spectral problem on Draga code")


#Plotting using built in random number generator to see if spectral problem present (it shouldn'

x2, y2, z2 = random_uniform_matrix(5000)
fig_2 = plt.figure(14)
ax_2 = fig_2.add_subplot(111, projection = "3d")
ax_2.scatter(x2, y2, z2, c="b", marker="o")
plt.title("Uniform points with no spectral problem")


#Samples distributed according to exp(-x/lambda)

x3 =random_exp(1000, mean_free_path_water_45)
plt.figure(15)
bins = 10 #Number of bins
bin_frequency, bin_edges, bin_temp = plt.hist(x3, bins)  #BinFreq = y
plt.xlabel("x (cm)")
plt.ylabel("N")
plt.savefig("Exponentialy distributed samples hist")
bin_x = []
for i in range(0, bins):
    bin_x.append((bin_edges[i] + bin_edges[i+1])/2) #Finding mid point of bins
coeff, covarience = np.polyfit(bin_x, np.log(bin_frequency), 1, cov = True)
coeff_m = coeff[0] # gradient
aL = (-1/coeff_m) #attentuation length
coeff_c = coeff[1] # y intercept
x4 = np.array(np.linspace(0,bin_x[bins-1], bins)) # x values
y4 = coeff_m*x4 + coeff_c #ln(y) = x * -1/lambda # coresponding y values
```

```
plt.figure(16)
plt.plot(x4, y4)
plt.scatter(bin_x, np.log(bin_frequency))
plt.title("Exponentialy distributed samples")
plt.xlabel("x (cm)")
plt.ylabel("log(N)")
plt.savefig("Exponentialy distributed samples lin")


#Generating isotropic unit vectors

x6, y6, z6 = uniform_sphere(1000)
fig_4 = plt.figure(17)
ax_4 = fig_4.add_subplot(111, projection = "3d")
ax_4.scatter(x6, y6, z6, c="r", marker="o")
plt.title("Uniform points over a sphere")#
plt.savefig("SPhere")


#Generating isotropic steps with lengths distributed as exp(-x/lambda)

x7, y7, z7 = isotropic_steps(10, mean_free_path_water)
fig_5 = plt.figure(18)
ax_5 = fig_5.add_subplot(111, projection = "3d")
ax_5.plot(x7, y7, z7, c="b", marker="o")
plt.title("Isotropic steps with lengths distributed as exp(-x/lambda)")
ax_5.set_xlabel('X Label')
ax_5.set_ylabel('Y Label')
ax_5.set_zlabel('Z Label')
plt.savefig("Isotropic steps ")


#Plotting random walks

#Water
fig_6 = plt.figure(19)
ax_6 = fig_6.add_subplot(111, projection = "3d")
ax_6.plot(water_particle_history_x, water_particle_history_y, water_particle_history_z, c="b", m
ax_6.set_xlabel('X ')
ax_6.set_ylabel('Y ')
ax_6.set_zlabel('Z ')
for i in range(0, water_steps+1):
    ax_6.text(water_particle_history_x[i], water_particle_history_y[i], water_particle_history_z
plt.legend()
plt.savefig("Simulated random walk-water")

#Lead
fig_7 = plt.figure(20)
ax_7 = fig_7.add_subplot(111, projection = "3d")
ax_7.plot(lead_particle_history_x, lead_particle_history_y, lead_particle_history_z, c="r", mark
ax_7.set_xlabel('X ')
ax_7.set_ylabel('Y ')
ax_7.set_zlabel('Z ')
for i in range(0, lead_steps+1):
    ax_7.text(lead_particle_history_x[i], lead_particle_history_y[i], lead_particle_history_z[i]
plt.legend()
plt.savefig("Simulated random walk-lead")

#Graphite
fig_8 = plt.figure(21)
ax_8 = fig_8.add_subplot(111, projection = "3d")
```

```
ax_8.plot(graphite_particle_history_x, graphite_particle_history_y, graphite_particle_history_z,
ax_8.set_xlabel('X ')
ax_8.set_ylabel('Y ')
ax_8.set_zlabel('Z ')
for i in range(0, graphite_steps+1):
    ax_8.text(graphite_particle_history_x[i], graphite_particle_history_y[i], graphite_particle_
plt.legend()
plt.savefig("Simulated random walk-graphite")
"""
```