Neutron Irradiation

S. Rager (z5558988)[1, 2]
[1] *Cohort B - Thu 14-18 class*
[2] *Word count: 2850 words*
(Dated: 08:40 Tuesday 9th April, 2024)

This series of experiments measured the half-lives of radio-nuclides produced in the irradiation of indium, silver, and copper and studied the growth of induced activity as a function of irradiation time. It was concluded that In-116m$_1$, Ag-99, Ag-116, Cu-66, and Cu-99 were produced and the importance of neutron capture cross sections was understood as they pertain to choosing materials for neutron moderators and absorbers. Finally, the percent composition of copper in a couple of coins was determined.

AIM

To measure the half-lives of some radio-nuclides produced in the thermal neutron irradiation of indium, silver, and copper metals, study the growth of the induced activity of indium as a function of irradiation time, understand the effects of thermal neutron shielding, and perform a simplified version of neutron activation analysis.

INTRODUCTION

Neutrons can only be liberated in nuclear reactions. The main interactions neutrons have with matter occur through nuclear processes because neutrons possess no charge. Because they do not experience a Coulomb barrier, very low energy neutrons can react with nuclei. Experimentally, neutrons with energies between 0.01 eV and 0.1 eV have high probabilities of reacting with nuclei, but most neutrons produced in nuclear reactions have energies in the MeV range and therefore must be slowed before they can be used to react with nuclei. Fast neutrons may be slowed by a series of elastic collisions with the nuclei in a substance called a moderator. In this case, the moderator was paraffin. A moderator is meant to thermalize neutrons in as few collisions as possible with a small probability of neutron absorption by the moderator.

As in FIG. 1., the irradiation apparatus consisted of a well with a neutron source at its center surrounded by six buckets placed at equidistant intervals around the neutron source, screwed onto the ends of Perspex rods as to be removable, with paraffin between the source and the rods. The well was enclosed on all sides by concrete and there was a removable steel plate on the top to allow the Perspex rods to be removed along with their buckets' contents. The neutron source was 20 Gigabecquerel americium-241 mixed with beryllium powder sealed inside a stainless steel cylinder 22 mm in diameter and 31 mm long. One becquerel corresponds to one disintegration per second. The Am-241 has a long half-life of 433 years and emits α-particles that react with the Be according to

$$^{9}_{4}Be +^{4}_{2} He \rightarrow^{1} 2_{6}C +^{1}_{0} n + 5.7MeV \qquad (1)$$

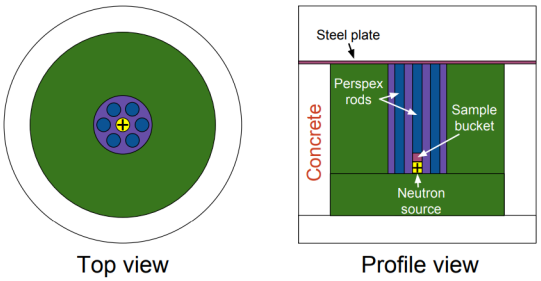

FIG. 1. Neutron Source Diagram

Table 2: Neutron flux ratios in the six moderator wells

Well position	Flux ratio
1	1.1
2	1.01
3	1.03
4	1.00
5	1.08
6	1.18

10^6 neutrons per second are emitted by the source. 5.7 MeV is liberated due to mass losses. It has been experimentally shown that the neutron energy spectrum ranges from 5.5 MeV to 11 MeV and peaks around 3.5 MeV. The 40 mm of paraffin enclosing the neutron source is used to slow the neutrons to thermal energies. 40 mm is a compromise between the 200 mm necessary to completely thermalize the neutrons and the smaller distance needed due to the decrease in neutron flux with increased separation.

The In samples were 30mm discs with a glued cardboard backing. The Ag samples were rolled hollow cylinders 20mm in diameter and 40 mm long. Samples were placed in Perspex buckets screwed onto the ends of the Perspex rods and lowered into the irradiation apparatus to be irradiated for their designated amounts of time. Two types of Geiger-Muller (GM) tubes were used for radiation detection. An end-window GM tube was used for all the experiments besides determining the half-life of Ag. To find the half-life of Ag, a thin-walled GM tube was used. The end-window GM tube was operated at 950 V and the thin-walled GM tube was operated at 900 V. Regard-

less of the type of GM tube used, they were both connected to a GM-Radiation counter to supply their operating voltage and take computer measurements of the count rates. Both apparatuses also had an uncertainty of one over the square root of the number of counts they recorded, making them accurate to better than plus or minus one count.

The count rate observed by the GM tubes, c, is proportional to the decay rate of the radioactive species

$$c = K(-\frac{dn}{dt}) = k\lambda n \tag{2}$$

For n_T target nuclei exposed to a flux ϕ of neutrons with σ neutron capture cross-section and λ decay constant, the time rate of change of the number of product nuclei n is

$$\frac{dn}{dt} = \phi\sigma n_T - \lambda n \Rightarrow n = \frac{\phi\sigma n_T}{\lambda}(1 - e^{-\lambda t}) \tag{3}$$

So that for long irradiation times t >> $\frac{1}{\lambda}$, n $\rightarrow \frac{\phi\sigma n_T}{\lambda}$. If n_T is assumed constant as it is for the growth of In experiment, the number of product nuclei saturates because eventually product nuclei decay as fast as they are produced by neutrons.

METHOD

Samples were placed in one of six buckets screwed onto the bottom of Perspex rods surrounding a 20 Gbq americium-241 neutron source. When samples were removed after being irradiated for a given time, it was noted which well they came from as to adjust the counts measured from the GM counter for neutron flux. Background measurements were also taken at different times throughout the lab on every day the lab was performed then averaged and subtracted from the neutron counts for isotopes. Background measurements were taken at different times throughout all lab sessions that were conducted using different preset times to increase the accuracy of the background measurements at any given point in time for each experiment conducted. The counts were corrected for neutron flux by dividing by the flux ratio provided by Table 2: Neutron flux ratios in the six moderator wells FIG.1.

To find the half-life of the In isotope, the sample was irradiated overnight then removed. The sample was allowed to rest for 2 minutes before being placed in the GM counter to be counted for 300 second intervals for a total of 3600 seconds.

In the growth of In activity experiment, In discs were irradiated for 10, 25, 40, 60, 90, 120, 150, and 180 minutes. Split between two afternoons, the measurements were taken for the discs irradiated for 90, 120, 150, and 180 minutes on the first afternoon and the other half of the experiment was conducted on the second. The discs were allowed to rest for 2 minutes before being counted for 600 seconds.

In the half-life of Ag isotope experiment, a silver cylinder was irradiated for 15 minutes then placed on the counter as soon as possible and counted for 15 sets of 30 seconds. Because most Ag isotopes have short half-lives, this experiment was repeated with the same methodology on the third and final day of the lab but an even greater attempt was made to transfer the silver cylinder from the neutron source to the GM tube quickly.

For the effect of shielding experiment, an inactive In disc was irradiated for 10 minutes as a control, allowed to cool for 2 minutes, and counted for 300 seconds. This process was repeated for 3 more rounds with the inactive In disc replaced by an inactive In disc sandwiched between lead, then cadmium, and finally indium. On the final day of lab, the same methodology was also applied to an inactive In disc sandwiched between silver.

To find the half-life of Cu isotopes two copper discs were irradiated for one hour and 1.25 hours respectively, each then allowed to cool for 2 minutes, and counted for 3 sets of 300 seconds. On the final day of the lab another copper disc was irradiated for one hour, let rest for 2 minutes, and then counted for 6 sets of 300 seconds. This disc was then allowed to sit for 2 hours before being irradiated for 15 minutes and counted for 15 sets of 30 seconds, following the same methodology of the half-life of Ag isotope experiment out of curiosity.

In order to find the Cu content of brass and 20c coins, the counts from the half-life of Cu isotope experiment were averaged to give an estimate of the number of neutrons that would be radiated by a pure copper coin of roughly the same dimensions and mass as the brass and 20c coins. Two brass coins were irradiated for one hour and 1.25 hours respectively then left to rest for two minutes before being counted for 3 sets of 300 seconds. On the final day, another brass coin was irradiated for one hour before being left to rest for 2 minutes and counted for 6 sets of 300 seconds. The exact same methodology was repeated for the 20c coins. Lastly, the data from the brass and 20c coins were averaged to provided the average number of neutrons radiated by each respective coin in 300 seconds. This was then divided by the average number of neutrons radiated by a pure copper coin to find the percent composition of copper of each coin.

RESULTS & ANALYSIS

Half-Life of In Isotope

To within 0.3% the measured Indium isotope half-life of 54.3 minutes agrees with the half-life stated in the referenced Table of Nuclides for In-116m_1 of 54.1 minutes. This half-life is derived from the equation of FIG. 2. It is interesting to note that this is a metastable state of In with one more neutron than the most common nuclide of In, In-115 which is itself weakly radioactive. In-116m_1 will continue to decay, but at a much slower rate than previously. FIG. 3. is a plot using log counts and linear time scales, pro-

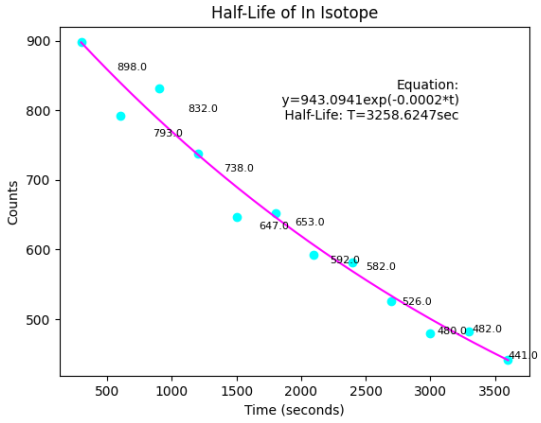

FIG. 2. Count Uncertainties +/- for Points Left to Right: [0.03, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.05, 0.05, 0.05]

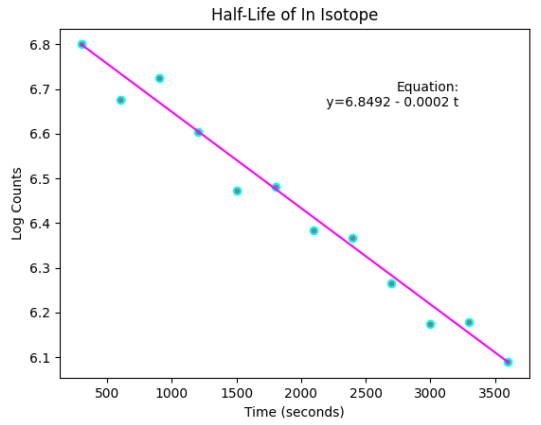

FIG. 3. Log Counts for Points Left to Right: [6.8 6.68 6.72 6.6 6.47 6.48 6.38 6.37 6.27 6.17 6.18 6.09]
Log Count Uncertainties +/- for Points Left to Right: [3.71e-05, 4.48e-05, 4.17e-05, 4.99e-05, 6.08e-05, 5.99e-05, 6.94e-05, 7.12e-05, 8.29e-05, 9.51e-05, 9.45e-05, 1.08e-04]

vided to make the exponential decay equation easier to visualize. The plot includes error bars, but the errors are so small, as can be seen in the caption, that the bars appear as grey points.

Growth of In Activity

After removing the In sample from the moderator one should wait for at least 1.5 minutes before starting the count to allow any contributions from short-lived isotopes to bleed off. The saturation count found as the asymptote to the equation of FIG. 4. is 1186. This is 505 counts less than the sum of the first two counts from FIG. 2. The data gathered in FIG. 4. is relatively noisy which contributes to this discrepancy.

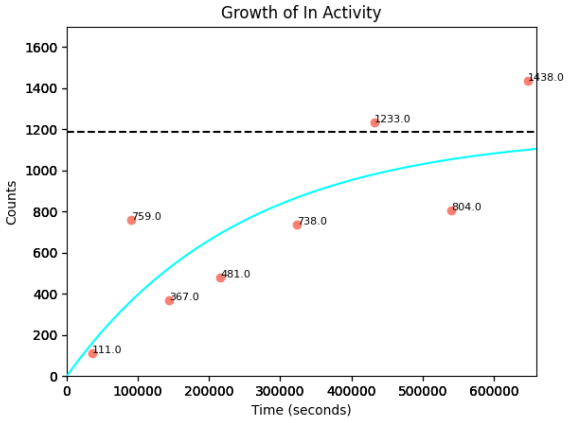

FIG. 4. Equation: y=1186(1-exp(-0.000004*t))
Saturation Count:1186 Count Uncertainties +/- for Points Left to Right: [0.10, 0.04, 0.05, 0.05, 0.04, 0.03, 0.04, 0.03]

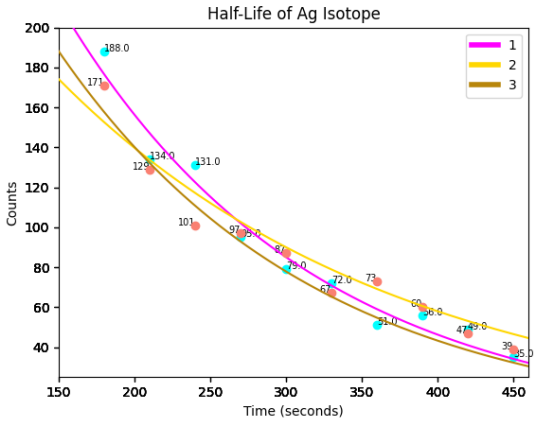

FIG. 5. Equation 1 (All Data Points): y=528.2186exp(-0.0061*t) Half-Life: T=113.8514sec
Equation 2: y=336.8359exp(-0.0044*t) Half-Life: T=157.5549sec
Equation 3: y=454.1474exp(-0.0059*t) Half-Life: T=117.9128sec
Cyan Count Uncertainties +/- for Points Left to Right: [0.07, 0.09, 0.09, 0.10, 0.11, 0.12, 0.14, 0.13, 0.14, 0.17]
Salmon Count Uncertainties +/- for Points Left to Right: [0.08, 0.09, 0.10, 0.10, 0.11, 0.12, 0.12, 0.13, 0.15, 0.16]

Half-Life of Ag Isotope

The half-life of Ag isotope 99 was measured to be 1.90 minutes, and this agrees with the value cited in the Table of Nuclides of 1.8 minutes to within 5.6%. The half-life of Ag-99 corresponds to Equation 1 in FIG. 5. The first few points of FIG. 6. are well off the curve of best fit because they account for the numerous contributions to the counts from short-lived isotopes that are radiated off the disk before a distinct isotope remains to be counted. Since the majority of Ag isotopes are long-lived on the scale of days or short-lived on the scale of under three minutes, a more accurate measurement of the half-life of silver isotopes could be achieved by letting the GM counter run for multiple days or moving the GM counter closer to the

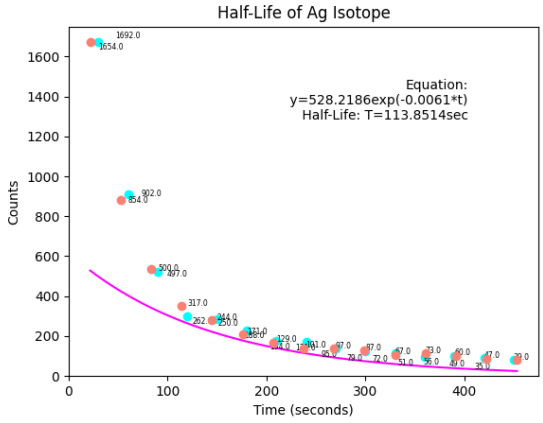

FIG. 6. Cyan Count Uncertainties +/- for Points Left to Right: [0.02, 0.03, 0.05, 0.06, 0.06, 0.07, 0.09, 0.09, 0.10, 0.11, 0.12, 0.14, 0.13, 0.14, 0.17]
Salmon Count Uncertainties +/- for Points Left to Right: [0.02, 0.03, 0.05, 0.06, 0.06, 0.08, 0.09, 0.10, 0.10, 0.11, 0.12, 0.12, 0.13, 0.15, 0.16]

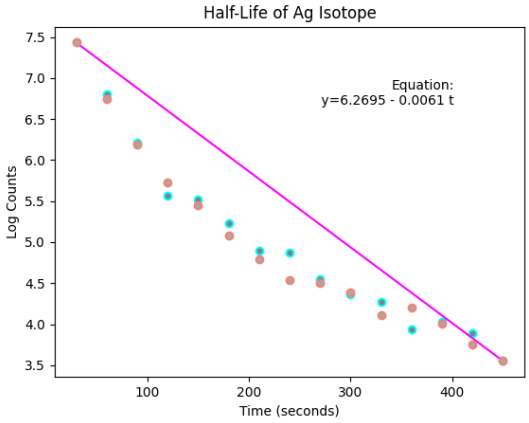

FIG. 7. Cyan Log Counts for Points Left to Right: [7.43 6.8 6.21 5.57 5.52 5.24 4.9 4.88 4.55 4.37 4.28 3.93 4.03 3.89 3.56]
Cyan Log Count Uncertainties +/- for Points Left to Right: [1.44e-05, 3.69e-05, 9.03e-05, 2.36e-04, 2.53e-04, 3.88e-04, 6.45e-04, 6.67e-04, 1.08e-03, 1.42e-03, 1.64e-03, 2.75e-03, 2.39e-03, 2.92e-03, 4.83e-03]
Salmon Log Counts for Points Left to Right: [7.41 6.75 6.21 5.76 5.5 5.14 4.86 4.62 4.57 4.47 4.2 4.29 4.09 3.85 3.66]
Salmon Log Count Uncertainties +/- for Points Left to Right: [1.49e-05, 4.01e-05, 8.94e-05, 1.77e-04, 2.62e-04, 4.47e-04, 6.83e-04, 9.85e-04, 1.05e-03, 1.23e-03, 1.82e-03, 1.60e-03, 2.15e-03, 3.10e-03, 4.11e-03]

irradiation apparatus to further minimize the time between removing the Ag cylinder from the apparatus and beginning counting. These limitations are why this experiment was repeated on the final day of the lab and multiple exponential decay plots were fitted to the data to try to capture some of the contributions from other short-lived isotopes. The half-life found from Equation 2 of FIG. 5. of 2.6 minutes corresponds best with that of Ag-116 of 2.68 minutes. These agree to within 3.1%. The half-life found from Equation 3

	Unshielded Indium	Lead Shield	Cadmium Shield	Indium Shield	Silver Shield
Counts in 300 seconds	359.405941	313.861386	159.405941	273.267327	257.281553
Count Uncertainties +/-	0.052748	0.056446	0.079204	0.060493	0.062344

FIG. 8.

of FIG. 5. of 1.96 minutes corresponds best with that of Ag-99 of 1.8 minutes again to within 8.4%. FIG. 7. is a plot of log counts versus linear times scales to better understand the exponential decay equation again. It has grey error bars that appear as dots on the plot because the error is so small.

Effect of Shielding

Cadmium is the best thermal neutron shield, or thermal neutron absorber as seen by FIG. 8. This is because cadmium has a high capture cross section, or probability, of absorbing thermal neutrons. In the growth of In activity experiment all of the In disks are not irradiated together in the same bucket because In acts as a neutron shield. Behind cadmium and silver it was the third best neutron shield of the four metals tested. One may assume lead would be an effective neutron shield because it is an effective gamma shield, however, the feature that makes it and other heavy nuclei materials an effective gamma shield is a detriment to its ability to absorb thermal neutrons because neutron collisions with heavy nuclei are very elastic. Therefore, heavy nuclei do very little to slow down neutrons. The second best neutron shield, slightly better than indium in this experiment, was silver. Interestingly, there is an expired patent, last cited 2023-05-02 for a neutron absorber to be used in pressurized water nuclear reactors, combining these three most effective tested neutron shields in the proportions 9-10% In, 4.35-5.35% Cd, and the rest Ag.

Half-Life of Cu Isotope

For the first copper disc, a half-life of 5.43 minutes indicates that Cu-66 was formed since Cu-66 has a half-life quoted from the referenced Table of Nuclides of 5.1 minutes. These values agree to within 6.1%. The previous analysis used the Blue Equation from FIG. 9. For the second copper disc, a half-life of 5.02 minutes agrees again with the quoted half-life of Cu-66 of 5.10 minutes to within 1.6%. This was using the Orange Equation from FIG. 9. Any other isotopes formed were either too short-lived or long-lived to have their half-lives measured in the time frame that was used. Most copper isotopes listed in the Table of Nuclides had half-lives too short-lived to expect to measure them with the experimental set up at hand. This is why longer irradiation times were used in hopes of measuring the half-lives of isotopes with half-lives on the scales of minutes. When the final copper irradiation was undertaken, a half-life of 3.3 minutes was found from the Gold Equation of FIG.

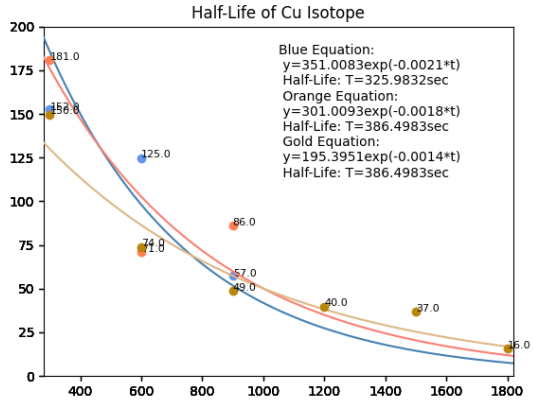

FIG. 9. Count Uncertainties +/- for Points Left to Right Alternating Cyan and Orange: [0.08, 0.07, 0.09, 0.12, 0.13, 0.11]
Gold Count Uncertainties +/- for Points Left to Right: [0.08, 0.12, 0.14, 0.16, 0.16, 0.25]

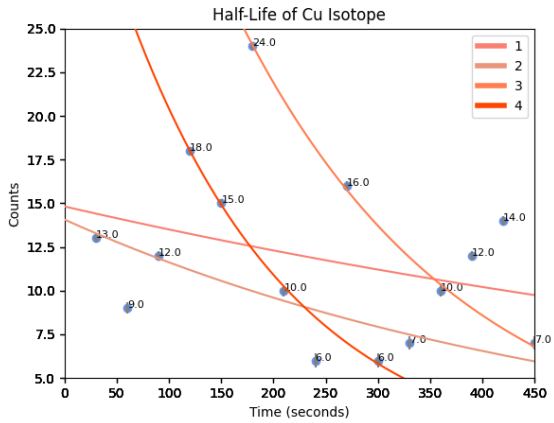

FIG. 10. Equation 1 (All Data Points): y=14.8152exp(-0.0009*t) Half-Life: T=745.0791sec
Equation 2: y=14.061exp(-0.0019*t) Half-Life: T=363.1432sec
Equation 3: y=55.7999exp(-0.0047*t) Half-Life: T=148.1979sec
Equation 4: y=38.1565exp(-0.0063*t) Half-Life: T=110.6589sec
Count Uncertainties +/- for Points Left to Right: [0.28, 0.33, 0.29, 0.24, 0.26, 0.20, 0.32, 0.41, 0.25, 0.41, 0.38, 0.32, 0.29, 0.27, 0.38]

9. This corresponds best with the half-life of Cu-69 of 3.0 minutes to within 9.1%.

In an attempt to find the half-lives of some of the short-lived isotopes of Cu, the methodology of the half-life of Ag isotope experiment was followed with the extra time on the final day of the lab. From FIG. 10., Equation 1 gives a half-life of 12.42 minutes, Equation 2 gives a half-life of 6.05 minutes, Equation 3 gives a half-life of 2.47 minutes, and Equation 4 gives a half-life of 1.83 minutes. These correspond to Cu-62, quoted half-life of 9.73 minutes, to within 21.66%; Cu-66, quoted half-life of 5.10 minutes, to within 15.7%; Cu-69, quoted half-life of 3.0 minutes,

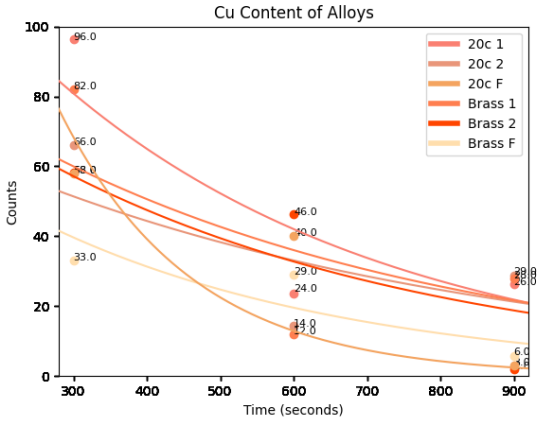

FIG. 11. 20c 1 - Equation:y=155.4408exp(-0.0022*t) Half-Life: T=318.0699sec Count Uncertainties +/- for Points Left to Right: [0.10, 0.20, 0.20]
20c 2 - Equation:y=79.8694exp(-0.0015*t) Half-Life: T=471.6902sec Count Uncertainties +/- for Points Left to Right: [0.12, 0.27, 0.19]
20c F - Equation:y=361.4731exp(-0.0056*t) Half-Life: T=124.8464sec Count Uncertainties +/- for Points Left to Right: [0.13, 0.16, 0.58]
Brass 1 - Equation:y=99.8636exp(-0.0017*t) Half-Life: T=407.9597sec Count Uncertainties +/- for Points Left to Right: [0.12, 0.27, 0.19]
Brass 2 - Equation:y=99.7755exp(-0.0019*t) Half-Life: T=373.4636sec Count Uncertainties +/- for Points Left to Right: [0.13, 0.15, 0.71]
Brass F - Equation:y=80.3592exp(-0.0024*t) Half-Life: T=294.3679sec Count Uncertainties +/- for Points Left to Right: [0.17, 0.19, 0.41]

to within 17.7%; and Cu-69, quoted half-life of 3.0 minutes, to within 38.9% respectively. The data does not agree particularly well with the values from the Table of Nuclides because the data is noisy and the experimental set up made it somewhat difficult to measure half-lives of short-lived isotopes.

Cu Content of Coins

During the final lab session, the copper disk and coins were counted for 6 sets of 300 seconds to get more data points to be able to obtain better curve fits. FIG. 9. shows these extra data points and how quickly they become negligible. While these extra data points were also collected for the coins in FIG. 11., they are omitted from the figure because they were all background. Once corrected for background and flux these points had zero or negative value and were thus omitted. FIG. 11. shows exponential decay curves fitted to each coin examined. For type of coin, brass or 20c, these counts were then averaged to give the average number of counts over a 300 second period and plotted in the bar graph of FIG. 12. with the averaged counts of pure copper over a 300 second period taken from FIG. 9. While the counts themselves had negligible uncertainties derived from the Poisson distribution, taking averages and finding percent composition in-

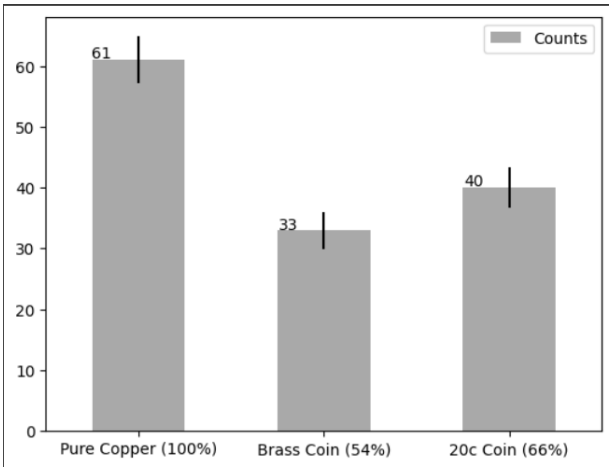

FIG. 12. Average Count Uncertainties +/- for Bars Left to Right: [3.87, 3.05, 3.34] Percent Composition Uncertainties +/- for Brass and 20c Coin Respectively: [8.52, 10.40]

troduced uncertainties as listed in the caption of FIG. 12.

Using this rudimentary methodology, the brass coin had a percent composition of copper of (54±8.52)% and the 20c coin had a percent composition of copper of (66±10.4)%. The copper, brass, and 20c coins had thicknesses of 2.1, 2.2, and 2.3 mm respectively, all ±0.01 mm. Their diameters were 28.3, 23.9, and 28.3 mm respectively with the same uncertainty. The presented methodology to find the Cu content of the coins is reasonable, since all the coins were relatively the same size, but the brass coin and copper disk differed in diameter by nearly 15.5%. A better method is presented in a paper by A. A. Gordus from the Department of Chemistry at the University of Michigan. With similar intentions, they irradiated a silver coin placed against a silver backing and were able to establish an empirical relationship for the percent composition of silver in the coin independent of coin thickness. They found that the ratio of the activity per gm of coin per percent silver divided by the activity in the silver backer disk was a constant E. From this they derived,

$$P = \frac{C_o}{C_b} mE \tag{4}$$

where C_o and C_b are the activities in the coin and backing respectively, m the mass of the coin, and P the percent silver in the coin. Were the experiment in this report to be continued or followed up on, it is recommended that a similar approach to Gordus's be followed where the coin whose composition of copper is to be determined is irradiated with a copper backing and an attempt is made at finding an empirical relationship.

CONCLUSIONS

The half-lives of In-116m$_1$, Ag-99, Ag-116, Cu-66, and Cu-69 were measured and the cause of count saturation was understood. Knowledge was gained of the importance of neutron capture cross sections in choosing a material to be used as a neutron absorber or moderator and basic neutron activation analysis was understood by determining the percent composition of copper in coins.

ACKNOWLEDGEMENTS

Davide Percopo (z5425666) for his contributions as my lab partner

I. Friedlander G. et al, "Nuclear and Radiochemistry" (1981) - 3rd edition
II. Krane K.S., "Introductory Nuclear Physics" (1988)
III. Lederer C.M., Hollander J.M. and Perlman I., "Table of Isotopes" (1968) - 6th edition.
IV. Gordus, A. A. (1967). "QUANTITATIVE NONDESTRUCTIVE NEUTRON ACTIVATION ANALYSIS OF SILVER IN COINS." Archaeometry 10(1): 78-86. <http://hdl.handle.net/2027.42/65909>
V. Google Patents, Silver alloy containing indium and cadmium for making neutron-absorbing elements, and use thereof, accessed 9 April 2024, <https://patents.google.com/patent/EP0737357A1/en>.

1. Half-Life of In Isotope

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df1=pd.read_csv(path+'In_exp1.tsv', sep='\t', skiprows=10)
df2=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df3=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df4=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df4.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df2.loc[:, 'Voltage']+df3.loc[:, 'Voltage']+coin)/8)
cdf=[]
i=1
while i <= 12:
  j=df1.loc[i, 'Voltage']-correction
  cdf.append(j)
  i+=1
cdf2=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf2.append(p3)
del cdf2[1::2]
cdf2 = np.array(cdf2, dtype=np.float32)
cdf3 = list(itertools.accumulate(cdf2))
cdf3 = np.array(cdf3, dtype=np.float32)
cdf3 = cdf3/1.1
t=[300,600,900,1200,1500,1800,2100,2400,2700,3000,3300,3600]
t = np.array(t, dtype=np.float32)
#[plt.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(t,
cdf2)]
#plt.title("Half-Life of In Isotope")
```

```
#plt.xlabel("Time (seconds)")
#plt.yscale("log")
#plt.ylabel("Counts")
#ax = plt.axes()
#ax.set_facecolor("black")
#plt.plot(t, cdf2, 'o', color='cyan')

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data = t
y_data = cdf2

# Convert data to PyTorch tensors
x_tensor = torch.from_numpy(x_data).float().view(-1, 1)
y_tensor = torch.from_numpy(y_data).float().view(-1, 1)
# Define the exponential decay model
#init = [940.69, 0.0002]
init = [940, 0.0001]
a_init = init[0]
b_init = init[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
```

```
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred = model(x_tensor)
    # Compute the loss
    loss = loss_function(y_pred, y_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit = model.a.item()
b_fit = model.b.item()

# Generate the fitted curve
t2=np.linspace(0,3600,3600)
y_fit = a_fit * np.exp(-b_fit * t2)

fig=plt.figure()
ax=fig.add_subplot(111, label="1")
ax2=fig.add_subplot(111, label="2", frame_on=False)
[plt.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(t, cdf2)]
plt.title("Half-Life of In Isotope")
ax.scatter(x_data, y_data, color="cyan")
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Counts")
sa_fit=str(round(a_fit,4))
T=np.log(2)/b_fit
sb_fit=str(round(b_fit,4))
T=str(round(T,4))
ax.text(0.85, 0.85, 'Equation:\n y='+sa_fit+'exp(-'+sb_fit+'*t)\n Half-
Life: T='+T+'sec', horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(t2, y_fit, color="magenta")
ax2.set_xticks([])
ax2.set_yticks([])
plt.show()
fig.savefig('Half-Life of In Isotope.png')
fig=plt.figure()
```

```
ax=fig.add_subplot(111, label="1")
ax2=fig.add_subplot(111, label="2", frame_on=False)
plt.title("Half-Life of In Isotope")
ax.scatter(x_data, np.log(y_data), color="cyan")
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Log Counts")
sa_fit2=np.log(a_fit)
sa_fit2=str(round(sa_fit2,4))
y_data2=np.log(y_data)
y_data2=np.array(y_data2)
y_data2=np.round(y_data2, 2)
#[plt.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(t,
y_data2)]
ax.text(0.85, 0.85, 'Equation:\n y='+sa_fit2+' - '+sb_fit+' t',
horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(t2, np.log(y_fit), color="magenta")
ax2.set_xticks([])
ax2.set_yticks([])
yerrs=1/np.sqrt(y_data)*(1/(y_data))
ax.errorbar(x_data, np.log(y_data), xerr=0, yerr=yerrs, fmt='.',
color='gray')
plt.show()
fig.savefig('Half-Life of In Isotope2.png')
print('Log Counts for Points Left to Right:',y_data2)
yerr=1/np.sqrt(y_data)
print('Count Uncertainties +/- for Points Left to Right:',yerr)
print('Log Count Uncertainties +/- for Points Left to Right:',yerrs)
```

2. Growth of In Activity

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
Indf1=pd.read_csv(path+'exp2_10.tsv', sep='\t', skiprows=10)
Indf2=pd.read_csv(path+'exp2_25.tsv', sep='\t',skiprows=10)
Indf3=pd.read_csv(path+'exp2_40.tsv', sep='\t',skiprows=10)
Indf4=pd.read_csv(path+'exp2_60.tsv', sep='\t',skiprows=10)
```

```
Indf5=pd.read_csv(path+'exp2_90.tsv', sep='\t',skiprows=10)
Indf6=pd.read_csv(path+'exp2_120.tsv', sep='\t',skiprows=10)
Indf7=pd.read_csv(path+'exp2_150.tsv', sep='\t',skiprows=10)
Indf8=pd.read_csv(path+'exp2_180.tsv', sep='\t',skiprows=10)
Indf9=pd.read_csv(path+'background_1.tsv', sep='\t',skiprows=10)
Indf10=pd.read_csv(path+'background_4.tsv', sep='\t',skiprows=10)
#Flux Ratio Indexed on Well Position
FR=[1,1.1,1.01,1.03,1.00,1.08,1.18]
correction = np.ceil((Indf9.loc[:, 'Voltage']+Indf10.loc[:, 'Voltage'])/2)
Indcdf1=[]
i=1
while i <= 1:
  j=Indf1.loc[i, 'Voltage']-correction
  Indcdf1.append(j)
  i+=1
Indcdf12=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf1:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf12.append(p3)
del Indcdf12[1::2]
Indcdf12 = np.array(Indcdf12, dtype=np.float32)
Indcdf13 = list(itertools.accumulate(Indcdf12))
Indcdf13 = np.array(Indcdf13, dtype=np.float32)
Indcdf13 = Indcdf13/FR[2]

Indcdf2=[]
i=1
while i <= 1:
  j=Indf2.loc[i, 'Voltage']-correction
  Indcdf2.append(j)
  i+=1
Indcdf22=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf2:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf22.append(p3)
del Indcdf22[1::2]
Indcdf22 = np.array(Indcdf22, dtype=np.float32)
```

```
Indcdf23 = list(itertools.accumulate(Indcdf22))
Indcdf23 = np.array(Indcdf23, dtype=np.float32)
Indcdf23 = Indcdf23/FR[3]

Indcdf3=[]
i=1
while i <= 1:
  j=Indf3.loc[i, 'Voltage']-correction
  Indcdf3.append(j)
  i+=1
Indcdf32=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf3:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf32.append(p3)
del Indcdf32[1::2]
Indcdf32 = np.array(Indcdf32, dtype=np.float32)
Indcdf33 = list(itertools.accumulate(Indcdf32))
Indcdf33 = np.array(Indcdf33, dtype=np.float32)
Indcdf33 = Indcdf33/FR[4]

Indcdf4=[]
i=1
while i <= 1:
  j=Indf4.loc[i, 'Voltage']-correction
  Indcdf4.append(j)
  i+=1
Indcdf42=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf4:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf42.append(p3)
del Indcdf42[1::2]
Indcdf42 = np.array(Indcdf42, dtype=np.float32)
Indcdf43 = list(itertools.accumulate(Indcdf42))
Indcdf43 = np.array(Indcdf43, dtype=np.float32)
Indcdf43 = Indcdf43/FR[5]

Indcdf5=[]
```

```
i=1
while i <= 1:
  j=Indf5.loc[i, 'Voltage']-correction
  Indcdf5.append(j)
  i+=1
Indcdf52=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf5:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf52.append(p3)
del Indcdf52[1::2]
Indcdf52 = np.array(Indcdf52, dtype=np.float32)
Indcdf53 = list(itertools.accumulate(Indcdf52))
Indcdf53 = np.array(Indcdf53, dtype=np.float32)
Indcdf53 = Indcdf53/FR[3]

Indcdf6=[]
i=1
while i <= 1:
  j=Indf6.loc[i, 'Voltage']-correction
  Indcdf6.append(j)
  i+=1
Indcdf62=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf6:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:5])
        Indcdf62.append(p3)
del Indcdf62[1::2]
Indcdf62 = np.array(Indcdf62, dtype=np.float32)
Indcdf63 = list(itertools.accumulate(Indcdf62))
Indcdf63 = np.array(Indcdf63, dtype=np.float32)
Indcdf63 = Indcdf63/FR[4]

Indcdf7=[]
i=2
while i <= 2:
  j=Indf7.loc[i, 'Voltage']-correction
  Indcdf7.append(j)
  i+=1
```

```
Indcdf72=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf7:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Indcdf72.append(p3)
del Indcdf72[1::2]
Indcdf72 = np.array(Indcdf72, dtype=np.float32)
Indcdf73 = list(itertools.accumulate(Indcdf72))
Indcdf73 = np.array(Indcdf73, dtype=np.float32)
Indcdf73 = Indcdf73/FR[5]

Indcdf8=[]
i=1
while i <= 1:
  j=Indf8.loc[i, 'Voltage']-correction
  Indcdf8.append(j)
  i+=1
Indcdf82=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Indcdf8:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:5])
        Indcdf82.append(p3)
del Indcdf82[1::2]
Indcdf82 = np.array(Indcdf82, dtype=np.float32)
Indcdf83 = list(itertools.accumulate(Indcdf82))
Indcdf83 = np.array(Indcdf83, dtype=np.float32)
Indcdf83 = Indcdf83/FR[6]

fig=plt.figure()
t=[10*3600, 25*3600, 40*3600, 60*3600, 90*3600, 120*3600, 150*3600,
180*3600]
scat=[]
scat.append(Indcdf13)
scat.append(Indcdf23)
scat.append(Indcdf33)
scat.append(Indcdf43)
scat.append(Indcdf53)
scat.append(Indcdf63)
scat.append(Indcdf73)
```

```
scat.append(Indcdf83)

# Generate example data
x_data2 = np.array(t)
y_data2 = np.array(scat)

# Convert data to PyTorch tensors
x_tensor2 = torch.from_numpy(x_data2).float().view(-1, 1)
y_tensor2 = torch.from_numpy(y_data2).float().view(-1, 1)
# Define the exponential decay model
init2 = [1185.8, 0.000001]
a_init2 = init2[0]
b_init2 = init2[1]
class AsymptoteModel(nn.Module):
    def __init__(self):
        super(AsymptoteModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init2, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init2, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a *(1- torch.exp(-self.b * x))

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = AsymptoteModel()
optimizer = optim.Adam(model.parameters(), lr=0.000001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred = model(x_tensor2)
    # Compute the loss
    loss = loss_function(y_pred, y_tensor2)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
b_fit2 = model.b.item()
a_fit2 = model.a.item()

# Generate the fitted curve
t3=np.linspace(0,660000,660000)
Y = a_fit2 * (1-np.exp(-b_fit2 * t3))
ax31=fig.add_subplot(111, label="301")
ax32=fig.add_subplot(111, label="302", frame_on=False)
ax33=fig.add_subplot(111, label="302", frame_on=False)
plt.title("Growth of In Activity")
ax31.scatter(t, scat, color="salmon")
ax32.plot(t3, Y, color="cyan")
ax33.axhline(y = a_fit2, color = 'k', linestyle = '--')
ax31.set_xlim([0,660000])
ax31.set_ylim([0,1700])
ax32.set_xlim([0,660000])
ax32.set_ylim([0,1700])
ax33.set_xlim([0,660000])
ax33.set_ylim([0,1700])
ax31.set_xlabel("Time (seconds)")
ax31.set_ylabel("Counts")
yerr=1/np.sqrt(scat)
scat = np.round(scat)
scat2=[]
for row in scat:
      p=str(row)
      p2 = p.replace('[', '')
      p3 = p2.replace(']', '')
      scat2.append(p3)
scat2 = np.array(scat2, dtype=np.float32)
[ax31.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(t,
scat2)]
print('Count Uncertainties +/- for Points Left to Right:',yerr)
a_fit2=str(round(a_fit2))
b_fit2=str(round(b_fit2,6))
print('Equation:\n y='+a_fit2+'(1-exp(-'+b_fit2+'*t))\n Saturation
Count:'+a_fit2+'')

fig.savefig('Growth of In Activity.png')
```

3. Half-Life of Ag Isotope

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
Agdf1=pd.read_csv(path+'exp3_15.tsv', sep='\t', skiprows=10) #15 entries
Agdf7=pd.read_csv(path+'exp3_ag1f.tsv', sep='\t', skiprows=10)
Agdf2=pd.read_csv(path+'background_3a.tsv', sep='\t',skiprows=10)
Agdf3=pd.read_csv(path+'background_3b.tsv', sep='\t',skiprows=10)
Agdf4=pd.read_csv(path+'background_3c.tsv', sep='\t',skiprows=10)
Agdf5=pd.read_csv(path+'background_agf1.tsv', sep='\t',skiprows=10)
Agdf6=pd.read_csv(path+'background_agf2.tsv', sep='\t',skiprows=10)
ag1=Agdf2.loc[:, 'Voltage']
ag1=np.sum(ag1)
ag2=Agdf3.loc[:, 'Voltage']
ag2=np.sum(ag2)
ag3=Agdf4.loc[:, 'Voltage']
ag3=np.sum(ag3)
ag4=Agdf5.loc[:, 'Voltage']
ag4=np.sum(ag4)
ag5=Agdf6.loc[:, 'Voltage']
ag5=np.sum(ag5)
correction = np.ceil((ag1+ag2+ag3+ag4+ag5)/120)
Agcdf=[]
i=1
while i <= 15:
  j=Agdf1.loc[i, 'Voltage']-correction
  Agcdf.append(j)
  i+=1
hlAg=np.array(Agcdf).tolist()
hlAg = hlAg[5:]
Agcdf2=[]
i=1
while i <= 15:
  j=Agdf7.loc[i, 'Voltage']-correction
  j=j/1.03
  j=round(j)
  Agcdf2.append(j)
  i+=1
hlAg2=np.array(Agcdf2)
```

```
hlAgf = hlAg2[5:]
Agt = [30,60,90,120,150,180,210,240,270,300,330,360,390,420,450]
Agt=np.array(Agt).tolist()
Agt2 = Agt[5:]
hlAg = np.array(hlAg, dtype=np.float32)
hlAg2 = np.array(hlAg2, dtype=np.float32)
Agt2 = np.array(Agt2, dtype=np.float32)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data3a = [210,270,300,360,390,420]
y_data3a = [134,100,90,75,62,48]
x_data3a=np.array(x_data3a)
y_data3a=np.array(y_data3a)
# Convert data to PyTorch tensors
x_tensor3a = torch.from_numpy(x_data3a).float().view(-1, 1)
y_tensor3a = torch.from_numpy(y_data3a).float().view(-1, 1)
# Define the exponential decay model
#init3 = [476.94, 0.006]
init3a = [350, 0.0001]
a_init3a = init3a[0]
b_init3a = init3a[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init3a, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init3a, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred3a = model(x_tensor3a)
    # Compute the loss
    loss = loss_function(y_pred3a, y_tensor3a)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit3a = model.a.item()
b_fit3a = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit3a = a_fit3a * np.exp(-b_fit3a * t23)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data3b = [210,270,300,360,450]
y_data3b = [133,95,79,51,35]
x_data3b=np.array(x_data3b)
y_data3b=np.array(y_data3b)
# Convert data to PyTorch tensors
x_tensor3b = torch.from_numpy(x_data3b).float().view(-1, 1)
y_tensor3b = torch.from_numpy(y_data3b).float().view(-1, 1)
# Define the exponential decay model
#init3 = [476.94, 0.006]
init3b = [435, 0.0001]
a_init3b = init3b[0]
b_init3b = init3b[1]
class ExponentialDecayModel(nn.Module):
```

```
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init3b, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init3b, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred3b = model(x_tensor3b)
    # Compute the loss
    loss = loss_function(y_pred3b, y_tensor3b)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit3b = model.a.item()
b_fit3b = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit3b = a_fit3b * np.exp(-b_fit3b * t23)

import torch
import torch.nn as nn
import torch.optim as optim
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

# Generate example data
x_data3 = Agt2
y_data3 = hlAg

# Convert data to PyTorch tensors
x_tensor3 = torch.from_numpy(x_data3).float().view(-1, 1)
y_tensor3 = torch.from_numpy(y_data3).float().view(-1, 1)
# Define the exponential decay model
#init3 = [476.94, 0.006]
init3 = [475, 0.0001]
a_init3 = init3[0]
b_init3 = init3[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init3, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init3, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred3 = model(x_tensor3)
    # Compute the loss
    loss = loss_function(y_pred3, y_tensor3)

    # Backward pass and optimization
    optimizer.zero_grad()
```

```
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit3 = model.a.item()
b_fit3 = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit3 = a_fit3 * np.exp(-b_fit3 * t23)

fig=plt.figure()
ax=fig.add_subplot(111, label="Ag1")
ax2=fig.add_subplot(111, label="Ag2", frame_on=False)
ax3=fig.add_subplot(111, label="Ag3", frame_on=False)
ax4=fig.add_subplot(111, label="Ag4", frame_on=False)
ax5=fig.add_subplot(111, label="Ag5", frame_on=False)
plt.title("Half-Life of Ag Isotope")
ax.scatter(x_data3, y_data3, color="cyan")
ax3.scatter(x_data3, hlAgf, color="salmon")
[ax.text(i, j, f'{j}', fontsize=7, ha='right') for (i, j) in zip(x_data3,
hlAgf)]
ax.set_xlim([150,460])
ax.set_ylim([25,200])
ax3.set_xlim([150,460])
ax3.set_ylim([25,200])
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Counts")
[ax.text(i, j, f'{j}', fontsize=7, ha='left') for (i, j) in zip(x_data3,
y_data3)]
a_fit32=str(round(a_fit3,4))
T3=np.log(2)/b_fit3
b_fit32=str(round(b_fit3,4))
T3=str(round(T3,4))
a_fit32a=str(round(a_fit3a,4))
T3a=np.log(2)/b_fit3a
b_fit32a=str(round(b_fit3a,4))
T3a=str(round(T3a,4))
a_fit32b=str(round(a_fit3b,4))
T3b=np.log(2)/b_fit3b
b_fit32b=str(round(b_fit3b,4))
T3b=str(round(T3b,4))
```

```
#ax.text(0.85, 0.85, 'Equation:\n y='+a_fit32+'exp(-'+b_fit32+'*t)\n Half-
Life: T='+T3+'sec', horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(t23, y_fit3, color="magenta")
ax2.set_xlim([150,460])
ax2.set_ylim([25,200])
ax4.plot(t23, y_fit3a, color="gold")
ax4.set_xlim([150,460])
ax4.set_ylim([25,200])
ax5.plot(t23, y_fit3b, color="darkgoldenrod")
ax5.set_xlim([150,460])
ax5.set_ylim([25,200])
custom_lines = [Line2D([0], [0], color='magenta', lw=4),
                Line2D([0], [0], color='gold', lw=4),
                Line2D([0], [0], color='darkgoldenrod', lw=4)]
ax.legend(custom_lines, ['1', '2', '3'])
plt.show()
fig.savefig('Half-Life of Ag Isotope.png')
print('Equation 1: y='+a_fit32+'exp(-'+b_fit32+'*t)\n Half-Life:
T='+T3+'sec\n'+
      'Equation 2: y='+a_fit32a+'exp(-'+b_fit32a+'*t)\n Half-Life:
T='+T3a+'sec\n'+
      'Equation 3: y='+a_fit32b+'exp(-'+b_fit32b+'*t)\n Half-Life:
T='+T3b+'sec')
yerr31=1/np.sqrt(y_data3)
print('Cyan Count Uncertainties +/- for Points Left to Right:',yerr31)
yerr31s=1/np.sqrt(hlAgf)
print('Salmon Count Uncertainties +/- for Points Left to Right:',yerr31s)

fig2=plt.figure()
ax=fig2.add_subplot(111, label="Ag12")
ax2=fig2.add_subplot(111, label="Ag22", frame_on=False)
ax3=fig2.add_subplot(111, label="Ag32", frame_on=False)
#[plt.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(Agt,
hlAg)]
plt.title("Half-Life of Ag Isotope")
ax.scatter(Agt, Agcdf, color="cyan")
ax.set_xlabel("Time (seconds)")
ax3.scatter(Agt, hlAg2, color="salmon")
[ax.text(i, j, f'{j}', fontsize=5.5, ha='left') for (i, j) in zip(Agt,
hlAg2)]
ax3.set_xticks([])
ax3.set_yticks([])
ax.set_xlim([0, 475])
ax.set_yticks([])
```

```
ax.text(0.85, 0.85, 'Equation:\n y='+a_fit32+'exp(-'+b_fit32+'*t)\n Half-
Life: T='+T3+'sec', horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(t23, y_fit3, color="magenta")
ax2.set_xticks([])
ax2.set_ylabel("Counts")
ax2.set_ylim([0, 1750])
[ax2.text(i, j, f'{j}', fontsize=5.5, ha='left') for (i, j) in zip(Agt,
Agcdf)]
plt.show()
fig2.savefig('Half-Life of Ag Isotope 2.png')
yerr32=1/np.sqrt(Agcdf)
print('Count Uncertainies +/- for Points Left to Right:',yerr32)
yerr32s=1/np.sqrt(hlAg2)
print('Salmon Count Uncertainties +/- for Points Left to Right:',yerr32s)

fig3=plt.figure()
ax=fig3.add_subplot(111, label="Ag13")
ax2=fig3.add_subplot(111, label="Ag23", frame_on=False)
ax3=fig3.add_subplot(111, label="Ag33", frame_on=False)
plt.title("Half-Life of Ag Isotope")
ax.scatter(Agt, np.log(Agcdf), color="cyan")
ax.set_xlabel("Time (seconds)")
ax3.scatter(Agt, np.log(hlAg2), color="salmon")
ax3.set_xticks([])
ax3.set_yticks([])
sa_fit2=np.log(a_fit3)
sa_fit2=str(round(sa_fit2,4))
sb_fit=str(round(b_fit3,4))
ax.text(0.85, 0.85, 'Equation:\n y='+sa_fit2+' - '+sb_fit+' t',
horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(t23, np.log(y_fit3), color="magenta")
ax2.set_xticks([])
ax.set_ylabel("Log Counts")
ax2.set_yticks([])
Agcdf = np.array(Agcdf, dtype=np.float32)
yerr33=1/np.sqrt(Agcdf)*(1/(Agcdf))
ax.errorbar(Agt, np.log(Agcdf), xerr=0, yerr=yerr33, fmt='.',
color='gray')
yerr33s=1/np.sqrt(hlAg2)*(1/(hlAg2))
ax3.errorbar(Agt, np.log(hlAg2), xerr=0, yerr=yerr33s, fmt='.',
color='darkgray')
plt.show()
fig3.savefig('Half-Life of Ag Isotope 3.png')
```

```
r=np.round(np.log(Agcdf),2)
print('Cyan Log Counts for Points Left to Right:', r)
print('Cyan Log Count Uncertainties +/- for Points Left to Right:',yerr33)
rs=np.round(np.log(hlAg2),2)
print('Salmon Log Counts for Points Left to Right:', rs)
print('Salmon Log Count Uncertainties +/- for Points Left to
Right:',yerr33s)
```

4. Effect of Shielding

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
In4df1=pd.read_csv(path+'exp4_1.tsv', sep='\t', skiprows=10) #15 entries
Cad4df1=pd.read_csv(path+'exp4_cad.tsv', sep='\t', skiprows=10) #15
entries
Ind4df1=pd.read_csv(path+'exp4_ind.tsv', sep='\t', skiprows=10) #15
entries
Lead4df1=[330]
Ag4df1=pd.read_csv(path+'exp4_Ag.tsv', sep='\t', skiprows=10)
In4df2=pd.read_csv(path+'background_3a.tsv', sep='\t',skiprows=10) #20
entries
In4df3=pd.read_csv(path+'background_3b.tsv', sep='\t',skiprows=10) #10
entries
In4df4=pd.read_csv(path+'background_3c.tsv', sep='\t',skiprows=10) #11
entries
In41=In4df2.loc[:, 'Voltage']
In41=np.sum(In41)/len(In41)
In42=In4df3.loc[:, 'Voltage']
In42=np.sum(In42)/len(In42)
In43=In4df4.loc[:, 'Voltage']
In43=np.sum(In43)/len(In43)
correction = np.ceil((In41+In42+In43)/3)
In4cdf=(In4df1.loc[1, 'Voltage']-correction)/1.01
Cad4cdf=(Cad4df1.loc[1, 'Voltage']-correction)/1.01
Ind4cdf=(Ind4df1.loc[1, 'Voltage']-correction)/1.01
Ag4cdf=(Ag4df1.loc[1, 'Voltage']-correction)/1.03
Lead4cdf=(Lead4df1-correction)/1.01
```

```
df4 = pd.DataFrame({'Unshielded Indium': [In4cdf, 1/np.sqrt(In4cdf)],
'Lead Shield': [Lead4cdf[0], 1/np.sqrt(Lead4cdf[0])],'Cadmium Shield':
[Cad4cdf, 1/np.sqrt(Cad4cdf)],
                    'Indium Shield': [Ind4cdf, 1/np.sqrt(Ind4cdf)],'Silver
Shield': [Ag4cdf, 1/np.sqrt(Ag4cdf)]}
                   ,index=['Counts in 300 seconds', 'Count Uncertainties
+/-'])
df4
```

5. Neutron Irradiation Copper

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
Cudf1=pd.read_csv(path+'exp5_1.tsv', sep='\t', skiprows=10)
Cudf2=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
Cudf3=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
Cudf4=pd.read_csv(path+'exp5_2.tsv', sep='\t', skiprows=10)
Cudf5=pd.read_csv(path+'exp5_Cuf1.tsv', sep='\t', skiprows=10)
Cudf6=pd.read_csv(path+'background_2fexp5.tsv', sep='\t', skiprows=10)
sum=Cudf6.loc[:, 'Voltage']
sum=np.sum(sum)
correction = np.ceil((Cudf2.loc[:, 'Voltage']+Cudf3.loc[:,
'Voltage']+sum)/10)
Cucdf=[]
i=1
while i <= 3:
  j=Cudf1.loc[i, 'Voltage']-correction
  Cucdf.append(j)
  i+=1
Cucdf2=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in Cucdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
```

```
        p3=(p2[1:4])
        Cucdf2.append(p3)
del Cucdf2[1::2]
Cu2cdf=[]
Cucdf2 = np.array(Cucdf2, dtype=np.float32)
Cucdf3 = Cucdf2/1.01
i=1
while i <= 3:
  j=Cudf4.loc[i, 'Voltage']-correction
  Cu2cdf.append(j)
  i+=1
Cu2cdf2=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in Cu2cdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Cu2cdf2.append(p3)
del Cu2cdf2[1::2]
Cu2cdf2 = np.array(Cu2cdf2, dtype=np.float32)
Cu2cdf3 = Cu2cdf2/1.03
Cu3cdf=[]
i=1
while i <= 6:
  j=Cudf5.loc[i, 'Voltage']-correction
  Cu3cdf.append(j)
  i+=1
Cu3cdf2=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in Cu3cdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Cu3cdf2.append(p3)
del Cu3cdf2[1::2]
Cu3cdf2 = np.array(Cu3cdf2, dtype=np.float32)
Cu3cdf3 = Cu3cdf2/1.03
Cut=[300,600,900]
Cut = np.array(Cut, dtype=np.float32)
Cutf=[300,600,900,1200,1500,1800]
Cutf = np.array(Cutf, dtype=np.float32)

import torch
```

```
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
Cux_data = Cut
Cux_data2 = Cutf
Cuy_data = np.copy(Cucdf3)
Cu2y_data = np.copy(Cu2cdf3)
Cu3y_data = np.copy(Cu3cdf3)

# Convert data to PyTorch tensors
Cux_tensor = torch.from_numpy(Cux_data).float().view(-1, 1)
Cux_tensor2 = torch.from_numpy(Cux_data2).float().view(-1, 1)
Cuy_tensor = torch.from_numpy(Cuy_data).float().view(-1, 1)
Cu2y_tensor = torch.from_numpy(Cu2y_data).float().view(-1, 1)
Cu3y_tensor = torch.from_numpy(Cu3y_data).float().view(-1, 1)
# Define the exponential decay model
#Cuinit = [357.44, 0.001]
Cuinit = [360, 0.0001]
Cua_init = Cuinit[0]
Cub_init = Cuinit[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(Cua_init, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(Cub_init, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
```

```
    # Forward pass
    Cuy_pred = model(Cux_tensor)
    # Compute the loss
    loss = loss_function(Cuy_pred, Cuy_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
Cua_fit = model.a.item()
Cub_fit = model.b.item()

#Cuinit = [311.67, 0.0008]
Cuinit = [310, 0.0001]
Cua_init = Cuinit[0]
Cub_init = Cuinit[1]

model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    Cuy_pred = model(Cux_tensor)
    # Compute the loss
    loss = loss_function(Cuy_pred, Cuy_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

Cu2a_fit = model.a.item()
Cu2b_fit = model.b.item()
###########
Cuinit = [185, 0.0001]
Cua_init = Cuinit[0]
Cub_init = Cuinit[1]
```

```
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    Cuy_pred = model(Cux_tensor2)
    # Compute the loss
    loss = loss_function(Cuy_pred, Cu3y_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

Cu3a_fit = model.a.item()
Cu3b_fit = model.b.item()

Cut2=np.linspace(0,2120,2120)
Cuy_fit = Cua_fit * np.exp(-Cub_fit * Cut2)
Cu2y_fit = Cu2a_fit * np.exp(-Cu2b_fit * Cut2)
Cu3y_fit = Cu3a_fit * np.exp(-Cu3b_fit * Cut2)

fig=plt.figure()
ax=fig.add_subplot(111, label="51")
ax2=fig.add_subplot(111, label="52", frame_on=False)
ax3=fig.add_subplot(111, label="53", frame_on=False)
ax4=fig.add_subplot(111, label="54", frame_on=False)
ax5=fig.add_subplot(111, label="55", frame_on=False)
ax6=fig.add_subplot(111, label="56", frame_on=False)
Cucdf3 = np.round(Cucdf3)
[ax.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(Cut,
Cucdf3)]
plt.title("Half-Life of Cu Isotope")
Culs = np.linspace(0,900,4)
Culs = np.array(Culs).tolist()
del Culs[0]
Culs = np.array(Culs)
ax.scatter(Cux_data, Cuy_data, color="cornflowerblue")
Cua_fitw=str(round(Cua_fit,4))
CuT=np.log(2)/Cub_fit
Cub_fitw=str(round(Cub_fit,4))
CuTw=str(round(CuT,4))
```

```
Cu2a_fitw=str(round(Cu2a_fit,4))
Cu2T=np.log(2)/Cu2b_fit
Cu2b_fitw=str(round(Cu2b_fit,4))
Cu2Tw=str(round(Cu2T,4))
Cu3a_fitw=str(round(Cu3a_fit,4))
Cu3T=np.log(2)/Cu3b_fit
Cu3b_fitw=str(round(Cu3b_fit,4))
Cu3Tw=str(round(Cu2T,4))
ax.text(0.5, 0.95, 'Blue Equation:\n y='+Cua_fitw+'exp(-'+Cub_fitw+'*t)\n
Half-Life: T='+CuTw+'sec\n Orange Equation:\n y='+Cu2a_fitw+
        'exp(-'+Cu2b_fitw+'*t)\n Half-Life: T='+Cu2Tw+'sec\n Gold
Equation:\n y='+Cu3a_fitw+
        'exp(-'+Cu3b_fitw+'*t)\n Half-Life: T='+Cu3Tw+'sec',
horizontalalignment='left', verticalalignment='top',
transform=ax.transAxes)
ax2.plot(Cut2, Cuy_fit, color="steelblue")
Cu2cdf3 = np.round(Cu2cdf3)
[ax3.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(Cut,
Cu2cdf3)]
ax3.scatter(Cut, Cu2y_data, color="coral")
ax6.scatter(Cutf, Cu3y_data, color="darkgoldenrod")
b=np.round(Cu3cdf3)
[ax6.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(Cutf, b)]
ax4.plot(Cut2, Cu2y_fit, color="salmon")
ax5.plot(Cut2, Cu3y_fit, color="burlywood")
ax.set_ylim([0, 200])
ax.set_xlim([280, 1820])
ax2.set_ylim([0, 200])
ax2.set_xlim([280, 1820])
ax3.set_ylim([0, 200])
ax3.set_xlim([280, 1820])
ax4.set_ylim([0, 200])
ax4.set_xlim([280, 1820])
ax5.set_ylim([0, 200])
ax5.set_xlim([280, 1820])
ax6.set_ylim([0, 200])
ax6.set_xlim([280, 1820])
plt.show()
yerr=1/np.sqrt([152, 181, 125, 71, 57, 86])
print('Count Uncertainties +/- for Points Left to Right Alternating Cyan
and Orange:',yerr)
tyerr=1/np.sqrt([150, 74, 49, 40, 37, 16])
print('Gold Count Uncertainties +/- for Points Left to Right:',tyerr)
fig.savefig('Half-Life of Copper Isotope.png')
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
a=pd.read_csv(path+'exp5_Cuf2.tsv', sep='\t', skiprows=10)
b=pd.read_csv(path+'background_3a.tsv', sep='\t',skiprows=10)
c=pd.read_csv(path+'background_3b.tsv', sep='\t',skiprows=10)
d=pd.read_csv(path+'background_3c.tsv', sep='\t',skiprows=10)
e=pd.read_csv(path+'background_agf1.tsv', sep='\t',skiprows=10)
f=pd.read_csv(path+'background_agf2.tsv', sep='\t',skiprows=10)
cu1=b.loc[:, 'Voltage']
cu1=np.sum(cu1)
cu2=c.loc[:, 'Voltage']
cu2=np.sum(cu2)
cu3=d.loc[:, 'Voltage']
cu3=np.sum(cu3)
cu4=e.loc[:, 'Voltage']
cu4=np.sum(cu4)
cu5=f.loc[:, 'Voltage']
cu5=np.sum(cu5)
correction = np.ceil((cu1+cu2+cu3+cu4+cu5)/120)
Cucdf=[]
i=1
while i <= 15:
  j=a.loc[i, 'Voltage']-correction
  Cucdf.append(j)
  i+=1
Cus=np.array(Cucdf).tolist()
Cust = [30,60,90,120,150,180,210,240,270,300,330,360,390,420,450]
Cust=np.array(Cust).tolist()
Cus = np.array(Cus, dtype=np.float32)
Cust = np.array(Cust, dtype=np.float32)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data3 = Cust
y_data3 = Cus
```

```
# Convert data to PyTorch tensors
x_tensor3 = torch.from_numpy(x_data3).float().view(-1, 1)
y_tensor3 = torch.from_numpy(y_data3).float().view(-1, 1)
# Define the exponential decay model
init3 = [15, 0.0001]
a_init3 = init3[0]
b_init3 = init3[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init3, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init3, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred3 = model(x_tensor3)
    # Compute the loss
    loss = loss_function(y_pred3, y_tensor3)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
```

```
a_fit3 = model.a.item()
b_fit3 = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit3 = a_fit3 * np.exp(-b_fit3 * t23)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data31 = [30,90,210,330]
x_data31 = np.array(x_data31)
y_data31 = [13,12,10,7]
y_data31 = np.array(y_data31)

# Convert data to PyTorch tensors
x_tensor31 = torch.from_numpy(x_data31).float().view(-1, 1)
y_tensor31 = torch.from_numpy(y_data31).float().view(-1, 1)
# Define the exponential decay model
init31 = [15, 0.0001]
a_init31 = init31[0]
b_init31 = init31[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init31, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init31, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
```

```
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred31 = model(x_tensor31)
    # Compute the loss
    loss = loss_function(y_pred31, y_tensor31)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit31 = model.a.item()
b_fit31 = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit31 = a_fit31 * np.exp(-b_fit31 * t23)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
x_data32 = [180,270,360,450]
y_data32 = [24,16,10,7]
x_data32=np.array(x_data32)
y_data32=np.array(y_data32)

# Convert data to PyTorch tensors
x_tensor32 = torch.from_numpy(x_data32).float().view(-1, 1)
y_tensor32 = torch.from_numpy(y_data32).float().view(-1, 1)
# Define the exponential decay model
init32 = [55, 0.0001]
a_init32 = init32[0]
b_init32 = init32[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
```

```
        self.a = nn.Parameter(torch.tensor(a_init32, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init32, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred32 = model(x_tensor32)
    # Compute the loss
    loss = loss_function(y_pred32, y_tensor32)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit32 = model.a.item()
b_fit32 = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit32 = a_fit32 * np.exp(-b_fit32 * t23)

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
```

```
# Generate example data
x_data33 = [120,150,210,300]
y_data33 = [18,15,10,6]
x_data33=np.array(x_data33)
y_data33=np.array(y_data33)

# Convert data to PyTorch tensors
x_tensor33 = torch.from_numpy(x_data33).float().view(-1, 1)
y_tensor33 = torch.from_numpy(y_data33).float().view(-1, 1)
# Define the exponential decay model
init33 = [35, 0.0001]
a_init33 = init33[0]
b_init33 = init33[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init33, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init33, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred33 = model(x_tensor33)
    # Compute the loss
    loss = loss_function(y_pred33, y_tensor33)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit33 = model.a.item()
b_fit33 = model.b.item()
# Generate the fitted curve
t23=np.linspace(0,500,500)
y_fit33 = a_fit33 * np.exp(-b_fit33 * t23)

from matplotlib.lines import Line2D
fig=plt.figure()
ax=fig.add_subplot(111, label="1")
ax2=fig.add_subplot(111, label="2", frame_on=False)
ax3=fig.add_subplot(111, label="3", frame_on=False)
ax4=fig.add_subplot(111, label="4", frame_on=False)
ax5=fig.add_subplot(111, label="5", frame_on=False)
plt.title("Half-Life of Cu Isotope")
ax.scatter(Cust, Cus, color="cornflowerblue")
[ax.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(Cust,
Cus)]
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Counts")
sa_fit=str(round(a_fit3,4))
T=np.log(2)/b_fit3
sb_fit=str(round(b_fit3,4))
T=str(round(T,4))
sa_fit1=str(round(a_fit31,4))
T1=np.log(2)/b_fit31
sb_fit1=str(round(b_fit31,4))
T1=str(round(T1,4))
sa_fit2=str(round(a_fit32,4))
T2=np.log(2)/b_fit32
sb_fit2=str(round(b_fit32,4))
T2=str(round(T2,4))
sa_fit3=str(round(a_fit33,4))
T3=np.log(2)/b_fit33
sb_fit3=str(round(b_fit33,4))
T3=str(round(T3,4))
#ax.text(0.95, 0.95, 'Equation:\n y='+sa_fit+'exp(-'+sb_fit+'*t)\n Half-
Life: T='+T+'sec', horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
yerrc2=1/np.sqrt(Cus)
```

```
ax.errorbar(Cust, Cus, xerr=0, yerr=yerrc2, fmt='.', color='gray')
ax2.plot(t23, y_fit3, color="salmon")
ax3.plot(t23, y_fit31, color="darksalmon")
ax4.plot(t23, y_fit32, color="coral")
ax5.plot(t23, y_fit33, color="orangered")
ax.set_xlim([0,450])
ax.set_ylim([5,25])
ax2.set_xlim([0,450])
ax2.set_ylim([5,25])
ax3.set_xlim([0,450])
ax3.set_ylim([5,25])
ax4.set_xlim([0,450])
ax4.set_ylim([5,25])
ax5.set_xlim([0,450])
ax5.set_ylim([5,25])
custom_lines = [Line2D([0], [0], color='salmon', lw=4),
                Line2D([0], [0], color='darksalmon', lw=4),
                Line2D([0], [0], color='coral', lw=4),
                Line2D([0], [0], color='orangered', lw=4)]
ax.legend(custom_lines, ['1', '2', '3', '4'])
plt.show()
fig.savefig('Half-Life of Cu Isotope 2.png')
print('Count Uncertainties +/- for Points Left to Right:',yerrc2)
print('Equation 1 (All Data Points): y='+sa_fit+'exp(-'+sb_fit+'*t)\n
Half-Life: T='+T+'sec\n'+
      'Equation 2: y='+sa_fit1+'exp(-'+sb_fit1+'*t)\n Half-Life:
T='+T1+'sec\n'+
      'Equation 3: y='+sa_fit2+'exp(-'+sb_fit2+'*t)\n Half-Life:
T='+T2+'sec\n'+
      'Equation 4: y='+sa_fit3+'exp(-'+sb_fit3+'*t)\n Half-Life:
T='+T3+'sec')
```

5. Neutron Irradiation Coins

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
```

```
import torch.nn as nn
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df120c1=pd.read_csv(path+'exp5_20c1.tsv', sep='\t', skiprows=10)
df220c1=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df320c1=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdf20c1=[]
i=1
while i <= 3:
  j=df120c1.loc[i, 'Voltage']-correction
  cdf20c1.append(j)
  i+=1
cdf220c1=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in cdf20c1:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf220c1.append(p3)
del cdf220c1[1::2]
cdf320c1 = np.array(cdf220c1, dtype=np.float32)
cdf320c1 = cdf320c1/1.1
texp5=[300,600,900]
texp5 = np.array(texp5, dtype=np.float32)

x_data20c1 = texp5
y_data20c1 = np.copy(cdf320c1)

# Convert data to PyTorch tensors
x_tensor20c1 = torch.from_numpy(x_data20c1).float().view(-1, 1)
y_tensor20c1 = torch.from_numpy(y_data20c1).float().view(-1, 1)
# Define the exponential decay model
#init20c1 = [185.08, 0.0006]
init20c1 = [145, 0.0001]
a_init20c1 = init20c1[0]
b_init20c1 = init20c1[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
```

```
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init20c1,
dtype=torch.float32, requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init20c1,
dtype=torch.float32, requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred20c1 = model(x_tensor20c1)
    # Compute the loss
    loss = loss_function(y_pred20c1, y_tensor20c1)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit20c1 = model.a.item()
b_fit20c1 = model.b.item()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
import torch.nn as nn
```

```
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df120c2=pd.read_csv(path+'exp5_20c2.tsv', sep='\t', skiprows=10)
df220c2=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df320c2=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdf20c2=[]
i=1
while i <= 3:
  j=df120c2.loc[i, 'Voltage']-correction
  cdf20c2.append(j)
  i+=1
cdf220c2=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdf20c2:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf220c2.append(p3)
del cdf220c2[1::2]
cdf320c2 = np.array(cdf220c2, dtype=np.float32)
cdf320c2 = cdf320c2/1.18

x_data20c2 = texp5
y_data20c2 = np.copy(cdf320c2)

# Convert data to PyTorch tensors
x_tensor20c2 = torch.from_numpy(x_data20c2).float().view(-1, 1)
y_tensor20c2 = torch.from_numpy(y_data20c2).float().view(-1, 1)
# Define the exponential decay model
#init20c2 = [148.59, 0.0008]
init20c2 = [70, 0.0001]
a_init20c2 = init20c2[0]
b_init20c2 = init20c2[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init20c2,
dtype=torch.float32, requires_grad=True))
```

```
        self.b = nn.Parameter(torch.tensor(b_init20c2,
dtype=torch.float32, requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred20c2 = model(x_tensor20c2)
    # Compute the loss
    loss = loss_function(y_pred20c2, y_tensor20c2)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit20c2 = model.a.item()
b_fit20c2 = model.b.item()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
import torch.nn as nn
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
```

```
df1b1=pd.read_csv(path+'exp5_b1.tsv', sep='\t', skiprows=10)
df2b1=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df3b1=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdfb1=[]
i=1
while i <= 3:
  j=df1b1.loc[i, 'Voltage']-correction
  cdfb1.append(j)
  i+=1
cdf2b1=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdfb1:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf2b1.append(p3)
del cdf2b1[1::2]
cdf3b1 = np.array(cdf2b1, dtype=np.float32)

x_datab1 = texp5
y_datab1 = np.copy(cdf3b1)
# Convert data to PyTorch tensors
x_tensorb1 = torch.from_numpy(x_datab1).float().view(-1, 1)
y_tensorb1 = torch.from_numpy(y_datab1).float().view(-1, 1)
# Define the exponential decay model
#initb1 = [136.63, 0.0007]
initb1 = [90, 0.0001]
a_initb1 = initb1[0]
b_initb1 = initb1[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_initb1, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_initb1, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)
```

```
def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_predb1 = model(x_tensorb1)
    # Compute the loss
    loss = loss_function(y_predb1, y_tensorb1)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fitb1 = model.a.item()
b_fitb1 = model.b.item()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
import torch.nn as nn
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df1b2=pd.read_csv(path+'exp5_b2.tsv', sep='\t', skiprows=10)
df2b2=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df3b2=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
```

```
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdfb2=[]
i=1
while i <= 3:
  j=df1b2.loc[i, 'Voltage']-correction
  cdfb2.append(j)
  i+=1
cdf2b2=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdfb2:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf2b2.append(p3)
del cdf2b2[1::2]
cdf3b2 = np.array(cdf2b2, dtype=np.float32)
cdf3b2 = cdf3b2/1.08

x_datab2 = texp5
y_datab2 = np.copy(cdf3b2)
# Convert data to PyTorch tensors
x_tensorb2 = torch.from_numpy(x_datab2).float().view(-1, 1)
y_tensorb2 = torch.from_numpy(y_datab2).float().view(-1, 1)
# Define the exponential decay model
#initb2 = [167.23, 0.001]
initb2 = [90, 0.0001]
a_initb2 = initb2[0]
b_initb2 = initb2[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_initb2, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_initb2, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)
```

```
# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_predb2 = model(x_tensorb2)
    # Compute the loss
    loss = loss_function(y_predb2, y_tensorb2)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fitb2 = model.a.item()
b_fitb2 = model.b.item()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
import torch.nn as nn
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df1bf=pd.read_csv(path+'exp5_brassf.tsv', sep='\t', skiprows=10)
df2bf=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df3bf=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdfbf=[]
```

```
i=1
while i <= 6:
  j=df1bf.loc[i, 'Voltage']-correction
  cdfbf.append(j)
  i+=1
cdf2b2f=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdfbf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf2b2f.append(p3)
del cdf2b2f[1::2]
del cdf2b2f[3:]
cdf3bf = np.array(cdf2b2f, dtype=np.float32)
cdf3bf = cdf3bf/1.03
t=[300,600,900,1200,1500,1800]
t = np.array(t, dtype=np.float32)
x_databf = texp5
y_databf = np.copy(cdf3bf)
# Convert data to PyTorch tensors
x_tensorbf = torch.from_numpy(x_databf).float().view(-1, 1)
y_tensorbf = torch.from_numpy(y_databf).float().view(-1, 1)
# Define the exponential decay model
#initb2 = [167.23, 0.001]
initbf = [100, 0.0001]
a_initbf = initbf[0]
b_initbf = initbf[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_initb2, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_initb2, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
```

```
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_predbf = model(x_tensorbf)
    # Compute the loss
    loss = loss_function(y_predbf, y_tensorbf)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fitbf = model.a.item()
b_fitbf = model.b.item()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
import torch
import torch.nn as nn
import torch.optim as optim
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
df120f=pd.read_csv(path+'exp5_20cf.tsv', sep='\t', skiprows=10)
df220f=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
df320f=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
df420=pd.read_csv(path+'background_2fexp5.tsv', sep='\t',skiprows=10)
coin=df420.loc[:, 'Voltage']
coin=np.sum(coin)
correction = np.ceil((df220c1.loc[:, 'Voltage']+df320c1.loc[:,
'Voltage']+coin)/8)
cdf20f=[]
i=1
while i <= 6:
```

```
  j=df120f.loc[i, 'Voltage']-correction
  cdf20f.append(j)
  i+=1
cdf2202f=[]
delete_list = ['1    ','Name: Voltage, dtype: float64']
for row in cdf20f:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        cdf2202f.append(p3)
del cdf2202f[1::2]
del cdf2202f[3:]
cdf320f = np.array(cdf2202f, dtype=np.float32)

x_data20f = texp5
y_data20f = np.copy(cdf320f)
# Convert data to PyTorch tensors
x_tensor20f = torch.from_numpy(x_data20f).float().view(-1, 1)
y_tensor20f = torch.from_numpy(y_data20f).float().view(-1, 1)
# Define the exponential decay model
#initb2 = [167.23, 0.001]
init20f = [370, 0.0001]
a_init20f = init20f[0]
b_init20f = init20f[1]
class ExponentialDecayModel(nn.Module):
    def __init__(self):
        super(ExponentialDecayModel, self).__init__()
        self.a = nn.Parameter(torch.tensor(a_init20f, dtype=torch.float32,
requires_grad=True))
        self.b = nn.Parameter(torch.tensor(b_init20f, dtype=torch.float32,
requires_grad=True))

    def forward(self, x):
        return self.a * torch.exp(-self.b * x)

def loss_function(y_pred, y_true):
    x = (y_pred - y_true)**2
    return torch.mean(x)

# Instantiate the model and define the loss function and optimizer
model = ExponentialDecayModel()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
```

```
num_epochs = 100000
for epoch in range(num_epochs):
    # Forward pass
    y_pred20f = model(x_tensor20f)
    # Compute the loss
    loss = loss_function(y_pred20f, y_tensor20f)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print the loss every 100 epochs
    #if (epoch + 1) % 100 == 0:
        #print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Extract the fitted parameters
a_fit20f = model.a.item()
b_fit20f = model.b.item()

from matplotlib.lines import Line2D
# Generate the fitted curve
exp5t2=np.linspace(0,920,920)
y_fit20c1 = a_fit20c1 * np.exp(-b_fit20c1 * exp5t2)
y_fit20c2 = a_fit20c2 * np.exp(-b_fit20c2 * exp5t2)
y_fit20f = a_fit20f * np.exp(-b_fit20f * exp5t2)
y_fitb1 = a_fitb1 * np.exp(-b_fitb1 * exp5t2)
y_fitb2 = a_fitb2 * np.exp(-b_fitb2 * exp5t2)
y_fitbf = a_fitbf * np.exp(-b_fitbf * exp5t2)

na_fit20c1=str(round(a_fit20c1,4))
T20c1=np.log(2)/b_fit20c1
nb_fit20c1=str(round(b_fit20c1,4))
nT20c1=str(round(T20c1,4))
na_fit20c2=str(round(a_fit20c2,4))
T20c2=np.log(2)/b_fit20c2
nb_fit20c2=str(round(b_fit20c2,4))
nT20c2=str(round(T20c2,4))
na_fitb1=str(round(a_fitb1,4))
Tb1=np.log(2)/b_fitb1
nb_fitb1=str(round(b_fitb1,4))
nTb1=str(round(Tb1,4))
na_fitb2=str(round(a_fitb2,4))
Tb2=np.log(2)/b_fitb2
nb_fitb2=str(round(b_fitb2,4))
```

```
nTb2=str(round(Tb2,4))
na_fit20f=str(round(a_fit20f,4))
T20f=np.log(2)/b_fit20f
nb_fit20f=str(round(b_fit20f,4))
nT20f=str(round(T20f,4))
na_fitbf=str(round(a_fitbf,4))
Tbf=np.log(2)/b_fitbf
nb_fitbf=str(round(b_fitbf,4))
nTbf=str(round(Tbf,4))

fig=plt.figure()
ax=fig.add_subplot(111, label="501")
ax2=fig.add_subplot(111, label="502", frame_on=False)
ax3=fig.add_subplot(111, label="503", frame_on=False)
ax4=fig.add_subplot(111, label="504", frame_on=False)
ax5=fig.add_subplot(111, label="505", frame_on=False)
ax6=fig.add_subplot(111, label="506", frame_on=False)
ax7=fig.add_subplot(111, label="507", frame_on=False)
ax8=fig.add_subplot(111, label="508", frame_on=False)
ax9=fig.add_subplot(111, label="509", frame_on=False)
ax10=fig.add_subplot(111, label="510", frame_on=False)
ax11=fig.add_subplot(111, label="511", frame_on=False)
ax12=fig.add_subplot(111, label="512", frame_on=False)
plt.title("Cu Content of Alloys")
ax.scatter(x_data20c1, y_data20c1, color="salmon")
ax2.scatter(x_data20c2, y_data20c2, color="darksalmon")
ax3.scatter(x_datab1, y_datab1, color="coral")
ax4.scatter(x_datab1, y_datab2, color="orangered")
ax5.scatter(x_databf, y_databf, color="navajowhite")
ax6.scatter(x_data20f, y_data20f, color="sandybrown")
ax7.plot(exp5t2, y_fit20c1, color="salmon")
ax8.plot(exp5t2, y_fit20c2, color="darksalmon")
ax9.plot(exp5t2, y_fitb1, color="coral")
ax10.plot(exp5t2, y_fitb2, color="orangered")
ax11.plot(exp5t2, y_fitbf, color="navajowhite")
ax12.plot(exp5t2, y_fit20f, color="sandybrown")
#ax.text(0.95, 0.95, 'Equation:\n y='+Cua_fit+'exp(-'+Cub_fit+'*t)\n Half-
Life: T='+CuT+'\n Equation:\n y='+Cu2a_fit+'exp(-'+Cu2b_fit+'*t)\n Half-
Life: T='+Cu2T+'', horizontalalignment='right', verticalalignment='top',
transform=ax.transAxes)
y_data20c1=np.round(y_data20c1)
y_data20c2=np.round(y_data20c2)
y_data20f=np.round(y_data20f)
y_datab1=np.round(y_datab1)
y_datab2=np.round(y_datab2)
```

```
y_databf=np.round(y_databf)
[ax.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in
zip(x_data20c1, y_data20c1)]
[ax2.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in
zip(x_data20c2, y_data20c2)]
[ax3.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in
zip(x_data20f, y_data20f)]
[ax4.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(x_databl,
y_databl)]
[ax5.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(x_databr,
y_databr)]
[ax6.text(i, j, f'{j}', fontsize=8, ha='left') for (i, j) in zip(x_databf,
y_databf)]
ax.set_ylim([0, 100])
ax.set_xlim([280, 920])
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Counts")
ax2.set_ylim([0, 100])
ax2.set_xlim([280, 920])
ax3.set_ylim([0, 100])
ax3.set_xlim([280, 920])
ax4.set_ylim([0, 100])
ax4.set_xlim([280, 920])
ax5.set_ylim([0, 100])
ax5.set_xlim([280, 920])
ax6.set_ylim([0, 100])
ax6.set_xlim([280, 920])
ax7.set_ylim([0, 100])
ax7.set_xlim([280, 920])
ax8.set_ylim([0, 100])
ax8.set_xlim([280, 920])
ax9.set_ylim([0, 100])
ax9.set_xlim([280, 920])
ax10.set_ylim([0, 100])
ax10.set_xlim([280, 920])
ax11.set_ylim([0, 100])
ax11.set_xlim([280, 920])
ax12.set_ylim([0, 100])
ax12.set_xlim([280, 920])

yerr51=1/np.sqrt(y_data20c1)
yerr52=1/np.sqrt(y_data20c2)
yerr53=1/np.sqrt(y_databl)
yerr54=1/np.sqrt(y_databr)
yerr55=1/np.sqrt(y_databf)
```

```
yerr56=1/np.sqrt(y_data20f)
print('20c 1 -> Equation:y='+na_fit20c1+'exp(-'+nb_fit20c1+'*t) Half-Life:
T='+nT20c1+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr51 )
print('20c 2 -> Equation:y='+na_fit20c2+'exp(-'+nb_fit20c2+'*t) Half-Life:
T='+nT20c2+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr52)
print('20c F -> Equation:y='+na_fit20f+'exp(-'+nb_fit20f+'*t) Half-Life:
T='+nT20f+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr56)
print('Brass 1 -> Equation:y='+na_fitb1+'exp(-'+nb_fitb1+'*t) Half-Life:
T='+nTb1+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr52)
print('Brass 2 -> Equation:y='+na_fitb2+'exp(-'+nb_fitb2+'*t) Half-Life:
T='+nTb2+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr54)
print('Brass F -> Equation:y='+na_fitbf+'exp(-'+nb_fitbf+'*t) Half-Life:
T='+nTbf+'sec'+'\n   Count Uncertainties +/- for Points Left to
Right:',yerr55)
custom_lines = [Line2D([0], [0], color='salmon', lw=4),
                Line2D([0], [0], color='darksalmon', lw=4),
                Line2D([0], [0], color='sandybrown', lw=4),
                Line2D([0], [0], color='coral', lw=4),
                Line2D([0], [0], color='orangered', lw=4),
                Line2D([0], [0], color='navajowhite', lw=4)]
ax.legend(custom_lines, ['20c 1', '20c 2', '20c F','Brass 1', 'Brass
2','Brass F'])
plt.show()
fig.savefig('Cu Content of Coins.png')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import itertools
#%matplotlib inline
path='/content/drive/MyDrive/PHYS3112/Neutron_Irradiation/'
Cudf1=pd.read_csv(path+'exp5_1.tsv', sep='\t', skiprows=10)
Cudf2=pd.read_csv(path+'background_2.tsv', sep='\t',skiprows=10)
Cudf3=pd.read_csv(path+'background_5.tsv', sep='\t',skiprows=10)
Cudf4=pd.read_csv(path+'exp5_2.tsv', sep='\t', skiprows=10)
Cudf5=pd.read_csv(path+'exp5_Cuf1.tsv', sep='\t', skiprows=10)
Cudf6=pd.read_csv(path+'background_2fexp5.tsv', sep='\t', skiprows=10)
sum=Cudf6.loc[:, 'Voltage']
sum=np.sum(sum)
```

```
correction = np.ceil((Cudf2.loc[:, 'Voltage']+Cudf3.loc[:,
'Voltage']+sum)/10)
Cucdf=[]
i=1
while i <= 3:
  j=Cudf1.loc[i, 'Voltage']-correction
  Cucdf.append(j)
  i+=1
Cucdf2=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in Cucdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Cucdf2.append(p3)
del Cucdf2[1::2]
Cu2cdf=[]
Cucdf2 = np.array(Cucdf2, dtype=np.float32)
Cucdf3 = Cucdf2/1.01
i=1
while i <= 3:
  j=Cudf4.loc[i, 'Voltage']-correction
  Cu2cdf.append(j)
  i+=1
Cu2cdf2=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in Cu2cdf:
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Cu2cdf2.append(p3)
del Cu2cdf2[1::2]
Cu2cdf2 = np.array(Cu2cdf2, dtype=np.float32)
Cu2cdf3 = Cu2cdf2/1.03
Cu3cdf=[]
i=1
while i <= 6:
  j=Cudf5.loc[i, 'Voltage']-correction
  Cu3cdf.append(j)
  i+=1
Cu3cdf2=[]
delete_list = ['1   ','Name: Voltage, dtype: float64']
for row in Cu3cdf:
```

```
  for word in delete_list:
        p=str(row)
        p2 = p.replace(word, "")
        p3=(p2[1:4])
        Cu3cdf2.append(p3)
del Cu3cdf2[1::2]
Cu3cdf2 = np.array(Cu3cdf2, dtype=np.float32)
Cu3cdf3 = Cu3cdf2/1.03
Cu=np.sum(Cu3cdf3)/len(Cu3cdf3)
eCu=np.sum(np.log(Cu3cdf3))/len(Cu3cdf3)
Brass=(np.sum(y_datab1)+np.sum(y_datab2)+np.sum(y_databf))/(len(y_datab1)+
len(y_datab2)+len(y_databf))
eBrass=(np.sum(np.log(y_datab1))+np.sum(np.log(y_datab2))+np.sum(np.log(y_
databf)))/(len(y_datab1)+len(y_datab2)+len(y_databf))
c20=(np.sum(y_data20c1)+np.sum(y_data20c2)+np.sum(y_data20f))/(len(y_data2
0c1)+len(y_data20c2)+len(y_data20f))
ec20=(np.sum(np.log(y_data20c1))+np.sum(np.log(y_data20c2))+np.sum(np.log(
y_data20f)))/(len(y_data20c1)+len(y_data20c2)+len(y_data20f))
#df = pd.DataFrame({'Pure Copper': [round(Cu)], 'Brass Coin':
[round(Brass)],'20c Coin': [round(c20)]})
df = pd.DataFrame({'':['Pure Copper (100%)', 'Brass Coin (54%)', '20c Coin
(66%)'], 'Counts':[round(Cu), round(Brass), round(c20)], 'yerr':[eCu,
eBrass, ec20]})
ax = df.plot.bar(x='', y='Counts', yerr='yerr', rot=0, color='darkgrey')
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() *
1.005))
print('Count Uncertainties +/- for Bars Left to Right:',eCu, eBrass, ec20)
peb=(eBrass/Brass+Cu/eCu)*0.54
pe20c=(ec20/c20+Cu/eCu)*0.66
print('Percent Uncertainties +/- for Brass and 20c Coin
Respectively:',peb, pe20c)
```

2/29/24 and 3/7/24

Neutron Irradiation

Decay rate: $-\frac{dn}{dt} = \lambda n \Rightarrow n = n_0 e^{-\lambda t}$ n_0 number of nuclei present at $t=0$, λ decay constant

$\frac{n_0}{2} = n_0 e^{-\lambda t}$ $\ln(1/2) = -\lambda t = \ln 1 - \ln 2 = -\ln 2 \Rightarrow \ln 2 = \lambda t$ $t_{1/2} = \frac{\ln 2}{\lambda}$

Nuclear counting: $c = K\left(-\frac{dn}{dt}\right) = k\lambda n$ $\frac{dn}{dt} = \phi \sigma n_T - \lambda n \Rightarrow n = \frac{\phi \sigma n_T}{\lambda}(1 - e^{-\lambda t})$

K = constant c = count rate n_T = number of target nuclei ϕ = flux (neutrons/area/time)
σ = neutron capture cross-section (area/nucleus)
$(t \gg 1/\lambda) n \rightarrow \phi \sigma n_T / \lambda$ i.e. n_T assumed constant $\Rightarrow$ number of product nuclei saturates since eventually the product nuclei decay as fast as they are being formed by the neutrons

Learning Goals:
- Use Poisson statistics to analyse counter-type data
- Explain the process of neutron irradiation
- Measure the half-lives of some radionuclides

~~Neutron Exp. Software Package~~

Output:
- Briefly explain with a diagram how the neutron moderator apparatus irradiates the samples
- Experimentally determine the half-life of In and Ag and compare with accepted values
- Explain the effect of shielding and provide experimental evidence for your explanations
- Determine the saturation count for In and discuss experimental considerations that should be taken

↑ Fit exponential decay curve ⇒ Just use Pytorch

1 Bq = 1 Becquerel = 1 disintegration/sec $^{9}_{4}Be + ^{4}_{2}He \rightarrow ^{12}_{6}C + ^{1}_{0}n + 5.7\,MeV$

Am-241 ($t_{1/2}$ = 433 years) emits α-particles which react with Be by
α-particles emitted w/ energies ~5.5 MeV $\Rightarrow$ neutrons can have initial energies up to ~11 MeV but experiments show neutron energy spectrum peaks at ~3.5 MeV. About 10^6 neutrons/sec. are emitted by source.

Warnings!

- Make sure a demonstrator switches off security alarm to neutron moderator room.
- Limit time in irradiation facility.
- Don't drop rods and buckets
- In, Cd, Pb are all poisonous ⇒ wear gloves. use tweezers
- All four rods must always be placed in moderator
- Ensure GM-Radiation counter switched off when changing tubes
- Don't poke tweezers through thin mica window

1. Half-Life of an Isotope Taken from 1

Correct the counts for room background and calculate the error for each point.
Fit an exponential decay curve to find the half-life of the In isotope.

Get a hard copy of the data, together with the curve of best fit using log (counts) vs. linear (time) scales

To find log count errors use $\delta z = \sqrt{(\delta x \frac{\partial f}{\partial x})^2}$

To find percent errors use $a = bc$ $\frac{\delta a}{a} = \frac{\delta b}{b} + \frac{\delta c}{c}$

Comment on results

Which In isotope's half-life have you measured? How does your half-life compare with the normally accepted value? $t_{1/2}$ = 54.1 min.

In 116 m, $t_{1/2}$ = 54.1 min. Agree within 6%

Measured background throughout experiment
60 90 120 150 180
Taken from 3 Taken from 4 Taken from 5 Taken from 6

2. Growth of In Activity
Taken From: 1,2 3 4 5
Correct counts for room background and neutron flux and, using the half-life from 1.) fit your experimental values of counts (including errors) versus irradiation time to find the saturation count S

Get a hard copy of the data and the best fit curve with linear scales on both axes (make sure the saturation line at S is included) Compare the saturation count with the sum of the first two counts from 1.) corrected for well position (Table 2) and comment on results.

After removing the In sample from the moderator why should you wait at least 1.5 minutes before starting to count?
To let all short-lived isotopes radiate off

Note:
exp5_background2
fine. Copper contributions had faded to just background. The Data consistent with other counts

20c
d = 28.3 mm
th = 2.3 mm

Brass
d = 23.9 mm
th = 2.2 mm

Cu coin
dimensions
dim = 28.3 mm
thi = 2.1 mm

3. Half-Life of Ag Isotope Taken from 4 15 rounds of 30 sec.

Subtract room background and fit an exponential decay curve to the counts for the last 5 minutes (leave out first 5 readings) to find half-life and best fit curve

On a log-linear graph, plot all (including first five) experimental counts with error bars, as a function of time, together with the best-fit curve. Extrapolate the best-fit curve back to zero time and comment on your results.

Which Ag isotope's half-life have you measured and how does your value compare with that normally accepted? $T_{1/2} = 1.9$ min. Found better match Ag 99 $T_{1/2} = 1.8$ min.
Ag 117 $T_{1/2} = 1.21$ min. Agree poorly - within 36%
Why are the first few points well off the curve of best fit?
Many short-lived isotopes are radiating neutrons off the cylinder
How could you determine the half-life of the Ag isotopes more precisely?
Be quicker in transfer

Cof2 Irradiate 15 min.
Immediately placed on counter
Taken from 4

Ag sandwich Taken from 3
at 5:23
Cof2 @ 5:25

4/4/2
4. Effect
Cadm
produ
In 2.) wh
bucket, a
Inactive I
lead shield
Cadmium sh
Indium sh
Referenc
Appendix
A±B =>
A*/B =>
stati
n counts
σ(N) =
N
Short Versio
1. Irrad
2. Irra
3.
4. Irr
5. Irr
5. Opt
a)
b)
15 m
20c
Bra
F

4/4/24 - Light gray/faded black pen at bottom of page and boxes

4. Effect of Shielding — Which metal is the best thermal neutron shield and why?

Cadmium because an indium disk shield with cadmium produced the fewest counts of the materials tested

In 2.) why aren't all the In discs irradiated together in the same bucket, and pulled out one at a time for counting at appropriate times?

Inactive In Taken from 2
lead shield Taken from 2 exp 4-2a counts 330 in 5 min.
Cadmium shield Taken from 2
Indium shield Taken from 2 Silver shield

References:

Friedlander G. et al. Nuclear and Radiochemistry (1981)
Krane K.S. Introductory Nuclear Physics (1988)
Lederer, Hollander, Perlman. Table of Isotopes (1968)

Appendix: Two independent measured quantities, A and B, obeying Poisson statistics

$A \pm B \Rightarrow \sigma(A \pm B) = (\sigma^2(A) + \sigma^2(B))^{1/2}$ Sum/Difference

$A \cdot / B \Rightarrow \frac{\sigma(A \cdot / B)}{A \cdot / B} = \left(\left(\frac{\sigma(A)}{A}\right)^2 + \left(\frac{\sigma(B)}{B}\right)^2\right)^{1/2}$ Product/Quotient

statistical error

116m, measured isotopes
Ag 99
Cu 66
Cu 69

n counts in a time t; the statistical error in the count rate N is

$\frac{\sigma(N)}{N} = \left(\left(\frac{\sigma(n)}{n}\right)^2 + \left(\frac{\sigma(t)}{t}\right)^2\right)^{1/2}$ assuming $\frac{\sigma(t)}{t}$ negligible $\frac{\sigma(N)}{N} = \frac{\sigma(n)}{n} = \frac{1}{\sqrt{n}}$

Short Version of What Must Be Done

1. Irradiate Over Night Count for 1 hr.
2. Irradiate 10, 25, 40, 60, 90, 2 hr, 2.5 hr, 3 hr
   Count each for 12 min.
3. Count Background 5 min.
   Irradiate M5 Count 7.5
4. Irradiate 10, cool 2, Count 5 x 4
5. Irradiate? { background_4 may be off. left in In sample
5. Optional 3 runs of 5 min. (300 sec.) no wait

Part 1 -
Part 2
1 hr
27.5 min
17 x 4 min = 68
60 + 68 + 27.5 = 2 hr 35.5 min.

a) Half-Life of Cu isotope: Cu1 Taken from 2 ↑1 hr. Cu2 Taken from 3 ↑1 hr. 15 min.

b) 20g1 Taken from 1 ↑1 hr. 20g2 Taken from 6 ↑1 hr. 15 min.
Brass coin Taken from 4 Brass coin taken from 5

agf1 taken from 2 ↙ Brass coin same
1 hr long copper taken from 3 let rest 2 min.
15 min. short copper taken from 2 no rest count for 15 sets of 30 sec.

20c coin goes in well at 4:05
Brass comes out of well at 4:35

1 hr taken from 4 rest

First 2 measurements of background f2 are bad (see note) to is
exp 5 -