



UPPSALA UNIVERSITET

BACHELOR THESIS IN PHYSICS

1FA599

Germanium detector efficiency and
identification of radioactive nuclides in
the atmosphere

Author:
Lars Jones

Supervisor:
Mattias Lantz

Subject Reader:
Andreas Solders

September 2, 2025

Preface

I would like to thank my supervisor Mattias Lantz for his support throughout this project. His guidance, feedback and excitement was really important in shaping this project.

Under the course of this project, Gemini and ChatGPT was used to partly generate some Python code. The code generated was assessed thoroughly and criticized. The generated code mainly served to accelerate the code writing process. They have **not** been used to generate any writing of the report.

Abstract

Radioactive elements are found everywhere on planet Earth, even in the air. In this project, the aim was to get a deeper understanding of the radioactive nuclides in the air. This was done by detecting gamma-rays from a filter, that had air flowing through it for a week, using a HPGe detector. Before these measurements were made, a foil of ^{232}Th was placed in the detector to get a grasp of the decay chain and to better understand the efficiency of the detector. The efficiency of the detector was found to be lower than the efficiency that was simulated. In the air filter, nuclides from the decay chains of ^{232}Th and ^{238}U were observed. All the nuclides identified were below the step of radon in the decay chains. The activity was then calculated and a theoretical decay fit, founded on the differential equations governing radioactive decay, was put upon the calculated values to be able to say something about the abundances of radioactive nuclides in the atmosphere. The decay fit was accurate for ^{238}U 's decay chain but less so for ^{232}Th 's when compared with experimental values. Future studies can continue the work on these differential equations to get a better understanding about the radionuclides in the atmosphere.

Sammanfattning

Radioaktiva grundämnen finns överallt på jorden, även i luften. I detta projekt var syftet att få en djupare förståelse för de radioaktiva nukliderna i luften. Detta gjordes genom att detektera gammastrålning från ett filter, som hade luft strömmades genom det under en vecka, med hjälp av en HPGe detektor. Innan dessa mätningar gjordes, placerades en folie av ^{232}Th i detektorn för att få en förståelse av sönderfallskedjan och för att bättre förstå detektorns effektivitet. Detektorns effektivitet visade sig vara lägre än den effektivitet som simulerades. Nukliderna i sönderfallskedjorna från ^{232}Th och ^{238}U var de som observerades. Alla identifierade nuklider låg under radonsteget i sönderfallskedjorna. Aktiviteten beräknades sedan, och en teoretisk sönderfallskurva, grundad i sönderfallslagarnas differentialekvationer, anpassades till de beräknade värdena för att kunna säga något om förekomsten av radioaktiva nuklider i atmosfären. Anpassningen var korrekt för ^{238}U :s sönderfallskedja men mindre så för ^{232}Th :s vid jämförelse med experimentella värden. Framtida studier kan fortsätta arbetet med dessa differentialekvationer för att få en bättre uppfattning av radionukliderna i atmosfären.

Contents

1	Introduction	6
2	Theory	7
2.1	Radioactive decay	7
2.1.1	Decay from a parent nuclide and growth of activity . .	7
2.1.2	Growth of activity in the air filter	8
2.1.3	Secular Equilibrium	9
2.2	^{232}Th and ^{238}U	9
2.2.1	Decay chains of ^{232}Th and ^{232}U	10
2.3	Gamma ray spectroscopy	10
2.3.1	Multi Channel Analyser	10
2.3.2	High Purity Germanium detector	11
2.3.3	Gamma ray interactions	11
2.4	Grubbs' Test	12
2.5	Propagation of uncertainty	12
2.6	True coincidence	13
2.7	Data point placement	13
3	Setup	15
4	Method	16
4.1	Calibration	16
4.2	The chosen decay chains	16
4.3	Data collection and measurements	16
4.3.1	^{232}Th foil	16
4.3.2	Background radiation	17
4.3.3	Air filter	17
4.4	Spectra analysis	18
4.4.1	^{232}Th foil	18
4.4.2	Background radiation	18
4.4.3	Air filter	18
4.5	Data analysis	19
4.5.1	^{232}Th foil	19
4.5.2	Background radiation	20
4.5.3	Air filter	21

5	Results	22
5.1	Background radiation	22
5.2	^{232}Th foil results	23
5.2.1	Spectrum analysis	23
5.2.2	Nuclides activity	25
5.2.3	HPGe detector efficiency assuming secular equilibrium	27
5.2.4	Nuclides activity scaled	28
5.2.5	Mean values of the scaled nuclides activity	30
5.3	Air filter results	31
5.3.1	Spectra analysis	31
5.3.2	Nuclide activity	32
6	Discussion	38
6.1	Discussion of ^{232}Th foil results	38
6.1.1	Spectrum analysis for the ^{232}Th foil	38
6.1.2	Nuclide activity for the ^{232}Th foil	38
6.1.3	HPGe detector efficiency assuming secular equilibrium	39
6.1.4	Nuclides activity scaled and mean values	39
6.2	Discussion of the Air filter results	39
6.2.1	Spectra analysis for the air filter	39
6.2.2	Nuclide activity for the air filter	40
7	Conclusion	42
8	Future research	43
A	Efficiency Table for the HPGe detector	46
B	Table of values for $T_{critical}$	47
C	Spectra from the air filter	48
C.1	^{232}Th 's decay chain and ^7Be peaks	48
C.2	^{238}U peaks	53
D	Tables with all the data extracted from the .csv files	57
D.1	Background radiation	57
D.2	^{232}Th foil	58
D.3	Air filter	60
D.3.1	Data from peaks from the decay chain of ^{232}Th	60

D.3.2	Data from peaks from the decay chain of ^{238}U	63
E	Python Code	65
E.1	^{232}Th foil data analysis	65
E.2	Air filter data analysis	80

1 Introduction

Radioactive radiation is everywhere around us, in our body, in the ground and even in the air we breathe. It is a central topic of discussion in the fields of energy, safety, and even medicine. In catastrophic events, radioactive elements can spread through the air with devastating consequences. Radiation of this kind can lead to severe health problems, including cancer [1]. So, what is the abundance of radioactive elements in the air and which elements can be found?

The radioactive elements Uranium and Thorium are naturally occurring and can be brought up in the air by agriculture or open-pit mining [2]. Therefore, it is valuable to measure the amount of radioactive nuclides found in the decay chains of ^{232}Th and ^{238}U . This helps improve our understanding of their abundances and supports the development of equations to model how these nuclides are distributed in the air. Several factors influence the ability to extract information about the atmosphere, such as weather conditions, seasonal variation, and the probability that a nuclide gets trapped in the filter.

In this project, measurements were conducted using a High-Purity Germanium (HPGe) detector and air samples collected using an air pump with filters on the roof of Ångströmlaboratoriet in Uppsala. The main objectives of the project are:

- To gain a deeper understanding of the detection efficiency of HPGe detectors.
- To identify radioactive nuclides in the atmosphere.
- To solve and experimentally verify differential equations describing the behavior of nuclides in the decay chains of ^{232}Th and ^{238}U present in air samples.

2 Theory

The theory will provide information that is useful to understand the project. It consist of explanations for reasoning and equations used during the project.

2.1 Radioactive decay

A radioactive atom decays at a rate that is different for every nuclide depending on how unstable they are. The decay rate of the atoms (activity) is directly proportional to the number of atoms present in the source. That is described by:

$$A(t) = -\frac{dN(t)}{dt} = \lambda N(t) \quad (1)$$

where $A(t)$ is the activity, which will be written as A , $N(t)$ is the number of atoms, which will be written as N , and λ is the decay constant. λ is defined as $\lambda = \frac{\ln 2}{t_{1/2}}$ where $t_{1/2}$ is the half-life of the radionuclide [3]. The solution to Equation 1 will look like:

$$N(t) = N_0 e^{-\lambda t}, \quad A(t) = \lambda N(t) = A_0 e^{-\lambda t} \quad (2)$$

where the subscript 0 refers to the initial value of the variable and t is time.

Following a radioactive decay, the atom is often left in an excited state. For the atom to transition into the ground state it may emit one or several gamma rays. The gamma rays emitted are photons with energies equal to the energy difference between the nuclear states [3].

2.1.1 Decay from a parent nuclide and growth of activity

When a radionuclide undergoes decay and transforms into another radionuclide, the original is referred to as the parent nuclide, and the resulting product is called the daughter nuclide. The rate of change in the number of radionuclides for the daughter is determined by the difference between the production of the daughter nuclide from decay of the parent nuclide and the decay of the daughter nuclide:

$$\frac{dN_D}{dt} = -A_D + A_P = -N_D \lambda_D + N_P \lambda_P \quad (3)$$

where the subscript D refers to daughter and the subscript P refers to parent [3]. This is a linear differential equation. The solution of this differential

equation came to the following:

$$N_D(t) = N_{P0} \cdot \left(\frac{\lambda_P}{\lambda_D - \lambda_P} \right) \cdot (e^{-\lambda_P t} - e^{-\lambda_D t}) + N_{D0} \cdot e^{-\lambda_D t} \quad (4)$$

With equation 1, equation 4 can be rewritten as:

$$A_D(t) = A_{P0} \cdot \left(\frac{\lambda_P}{\lambda_D - \lambda_P} \right) \cdot (e^{-\lambda_P t} - e^{-\lambda_D t}) + A_{D0} \cdot e^{-\lambda_D t} \quad (5)$$

This can be taken further to calculate the activity for the granddaughter nuclide while including the growth from the daughter nuclide and the parent nuclide [4]. This will lead to an equation looking like this:

$$\begin{aligned} A_G(t) = A_{P0} \cdot \lambda_P \cdot \lambda_D \cdot & \left(\frac{e^{-\lambda_P t}}{(\lambda_D - \lambda_P)(\lambda_G - \lambda_P)} + \frac{e^{-\lambda_D t}}{(\lambda_P - \lambda_D)(\lambda_G - \lambda_D)} \right. \\ & \left. + \frac{e^{-\lambda_G t}}{(\lambda_P - \lambda_G)(\lambda_D - \lambda_G)} \right) + A_{D0} \cdot \frac{\lambda_D}{\lambda_G - \lambda_D} (e^{-\lambda_D t} - e^{-\lambda_G t}) \\ & + A_{G0} \cdot e^{-\lambda_G t} \end{aligned} \quad (6)$$

In this equation the G refers to the granddaughter, the D refers to the daughter and P, the parent.

2.1.2 Growth of activity in the air filter

To be able to understand the growth of activity in the air filter, Equation 3 was used. It was altered to:

$$\frac{dN_D}{dt} = -N_D \lambda_D + N_P \lambda_P + D \quad (7)$$

where D is the capture of nuclides from the air. In the case with three nuclides in a chain it will look like:

$$\frac{dN_P}{dt} = -N_P \lambda_P + P \quad (8)$$

$$\frac{dN_D}{dt} = -N_D \lambda_D + N_P \lambda_P + D \quad (9)$$

$$\frac{dN_G}{dt} = -N_G \lambda_G + N_D \lambda_D + G \quad (10)$$

where P, D and G represent the production of the nuclide in question from the air i.e. the amount of nuclides that got stuck in the filter.

2.1.3 Secular Equilibrium

In radioactive decay chains, various types of equilibrium can occur, each offering distinct conclusions about the dynamics of the decay process and the relative activities of parent and daughter nuclides. Secular equilibrium is the equilibrium state that refers to when the half-life of a parent nuclide is much longer than the half-life of the daughter nuclide, that is, $t_{1/2P} \gg t_{1/2D}$. This will lead to the activity being the same for both the parent and the daughter nuclides, i.e. $A_P = A_D$ [3]. When this is combined with equation 3 it will lead to:

$$\frac{dN_D}{dt} = 0 \quad (11)$$

If we combine eq 3 with eq 11 we get:

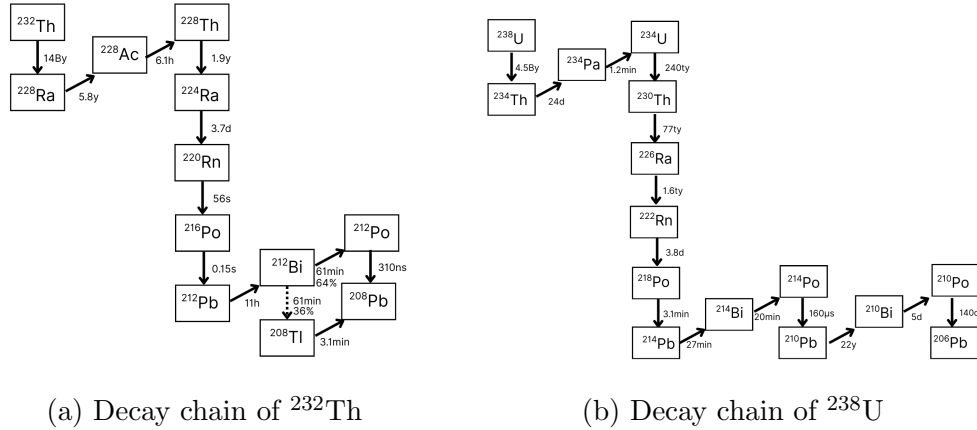
$$N_D \lambda_D = N_P \lambda_P \rightarrow N_D = \frac{N_P \lambda_P}{\lambda_D} \quad (12)$$

For a decay chain to reach secular equilibrium it can take different amount of time which can be calculated with equation 5.

2.2 ^{232}Th and ^{238}U

Both ^{232}Th and ^{238}U are naturally occurring radioactive elements and are embedded in the bedrock. They are leached out and follow streams to lakes and oceans where they reside in sediments and, in some extension, dissolved in the water. In both the decay chains of ^{232}Th and ^{238}U , there is a step of radon (^{220}Rn and ^{222}Rn respectively). At that stage of the decay chain, it can leave the ground and rise into the air. Radon can do this mainly because it is a noble gas that does not bind to soil particles or minerals. This leads to elements appearing in the air from both decay chains. There is a possibility that dust particles from uranium and thorium are carried into the air by agriculture or open-pit mining, which could lead to the detection of nuclides above radon in the decay chains.

2.2.1 Decay chains of ^{232}Th and ^{232}U



(a) Decay chain of ^{232}Th

(b) Decay chain of ^{238}U

Figure 1: Decay chains of ^{232}Th and ^{238}U

2.3 Gamma ray spectroscopy

Gamma ray spectroscopy is the study of the energy spectra of gamma ray sources and involves the analysis of emitted gamma radiation to identify and quantify radioactive materials.

2.3.1 Multi Channel Analyser

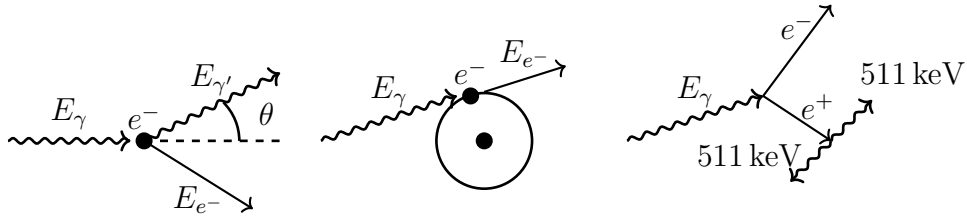
A Multi Channel Analyser (MCA) is an instrument that counts events in real time and sorts these events. The events are sorted into different channels based on some characteristic of the events. The characteristic of interest in this project is *pulse-height analysis* (PHA) which is proportional to the energy of the particle detected. An *analog-to-digital converter* (ADC) is used to convert the pulse into a channel number. The channel number corresponds to a narrow range of pulse heights, effectively grouping pulses with similar amplitudes. When pulses arrives, the MCA distributes all these, with respect to the height of the pulse, to the corresponding channel. Each peak distributed is counted as one event. The distribution arranged in increasing energies is called a *spectrum* [5]. MAESTRO [5] is a software which emulates an MCA.

2.3.2 High Purity Germanium detector

The detector that was used in the project was a High Purity Germanium (HPGe) detector which is a sort of semiconductor. The HPGe detector is kept cold so that no valence electrons can cross the band gap because of thermal energy, which leads to the electrons only being able to cross the band gap when they get excited by gamma radiation. When electrons crossed the band gap to the conduction band, they send a pulse to the MCA [3, 6].

2.3.3 Gamma ray interactions

The detected energy are essentially electrons that carry the accumulated energy of the gamma ray. When the energy detected cohere with the energy of the gamma ray, it is called the photoelectric effect. The energy detected in the detector may not cohere with the energy of the gamma ray because of two interactions that can happen in the detector. These are called Compton scattering and pair production [3].



Compton scattering Photoelectric effect Pair production

Figure 2: Gamma ray interactions

In the case of Compton scattering, the gamma ray will scatter off an electron which will result in the energy detected ($E_{detected}$) in the MCA to be different from the actual energy of the gamma ray (E_γ). The detected energy depends on the scattering angle (θ) as described by the following equation $E_{detected} = E_\gamma(1 - \frac{1}{1+E_\gamma(1-\cos\theta)/m_0c^2})$. This interaction will create a Compton edge and continuum [3].

The photoelectric effect arises when the gamma-ray photon interacts with one of the bound electrons in an atom. The energy detected equals the energy of the gamma-ray in this case as $E_{detected} = E_\gamma$ [3].

Pair production occurs when a gamma ray interacts with an entire atom. The process, in which the gamma ray converts into an electron-positron pair, takes place within the Coulomb field of the nucleus. For this process to take place, the gamma-ray must carry an energy of, at least, the same as the rest mass of the two particles it converts into which in this case is $511keV$ each. The positron will then annihilate with an electron which will result in two gamma-rays with $E = 511keV$ each, and they will propagate in opposite directions. This can result in a single escape detection event as $E_{detected} = E_{\gamma} - 511keV$ or a double escape detection event as $E_{detected} = E_{\gamma} - 1022keV$ [3].

2.4 Grubbs' Test

The Grubbs' Test is a statistical test used to detect outliers in a set of data points. The test was created by Frank Grubbs in 1950 to detect outliers. It is used frequently in statistical analyses [7]. The formula developed by Grubbs' looks like this:

$$T_i = \frac{|x_i - \bar{x}|}{\sigma} \quad (13)$$

where x_i is the value of one of the data points, \bar{x} is the mean value of all the data points, σ is the standard deviation and T_i is the Grubbs' value which will be compared to a another value called $T_{critical}$. The values of $T_{critical}$ can be found in a table (Figure 21) in Section B in the Appendix. It varies with the number of data points (n) in the set. If $T_i > T_{critical}$, the data point associated with that Grubbs' value, is an outlier to the set of data points [8]. To be able to perform the Grubbs' test on a set of data points, the set must be able to be considered as a normal population [7].

2.5 Propagation of uncertainty

To propagate the uncertainty the following equation is used from Physics Handbook [9]:

$$\Delta Z = \sqrt{\left(\frac{\delta Z}{\delta x}\right)^2 (\Delta x)^2 + \left(\frac{\delta Z}{\delta y}\right)^2 (\Delta y)^2 + \dots} \quad (14)$$

where Z is the product, x and y are some variables that Z is a function of. Δ is implying the error of the given variable.

2.6 True coincidence

True coincidence occurs when two or more photons are emitted from the same decay. The detector can register these as one event when they come in such a fast succession to each other. For the case of ^{208}Tl this occurs for the gamma rays with the energies at about 583 keV, 861 keV and 2615 keV. Both the lower energy gamma rays precede the one at 2615 keV as seen in Figure 3. This can result in a summation peak at their added energy. This leads to the peaks at these energies having fewer counts than expected. There are methods available to correct for this discrepancy but such corrections are not performed within this project [10, 11].

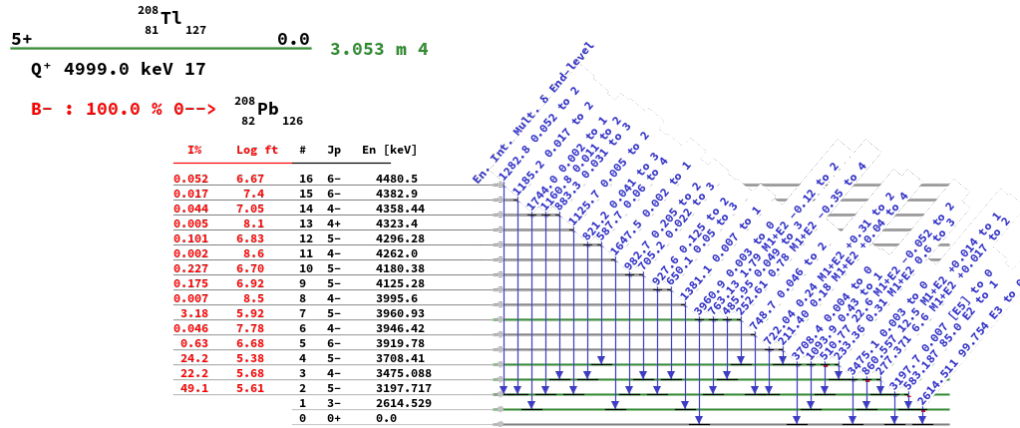


Figure 3: A decay scheme which shows the gamma decays between energy levels of ^{208}Tl [11]

2.7 Data point placement

All the measurements done in this project were done over different durations. Since a nuclide's activity changes over time due to radioactive decay, the activity at the beginning of a measurement period will not be the same as at the end. To be able to know where to put the data point in the time span of the measurement the following equation was used with respect to the time span and the decay constant of the nuclide in question:

$$\bar{t} = \frac{1}{\lambda} - \frac{t_2 - t_1}{e^{\lambda(t_2 - t_1)} - 1} + t_1 \quad (15)$$

where t_1 is the time the measurement started and t_2 is the time the measurement ended. This derives from the mean of a function over an interval equation [12]:

$$\bar{f} = \frac{1}{b-a} \int_a^b f(t) dt \quad (16)$$

where the function instead is the decay function 1. Then it will look like:

$$\bar{A} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} A(t) dt \quad (17)$$

3 Setup

The setup used to measure the radioactivity of the samples is called Uppsala Generic Gamma LABORatory (UGGLA). UGGLA consists of an HPGe detector on which the samples are placed. The detector is located inside a chamber that consists of lead, copper, and iron to minimize background radiation from other sources to hit the detector, that would interfere with the measurement of the actual sample. The detector is cooled with liquid nitrogen ($N_{2(l)}$) which is kept in a dewar tank. An amplifier is connected to the HPGe detector to amplify the signal which in turn will be registered at the computer using the software MAESTRO[5] which is an MCA (see section 2.3.1) emulator software.

To take samples from the air, an air pump was placed on the roof of Ångströmslaboratoriet. A filter called CHIEF from the company Senya, with dimensions 270mm x 270mm, was placed in front of the air pump to filter out the radionuclides from the air going through it. No consideration has been given to the filter's characteristics in this project.

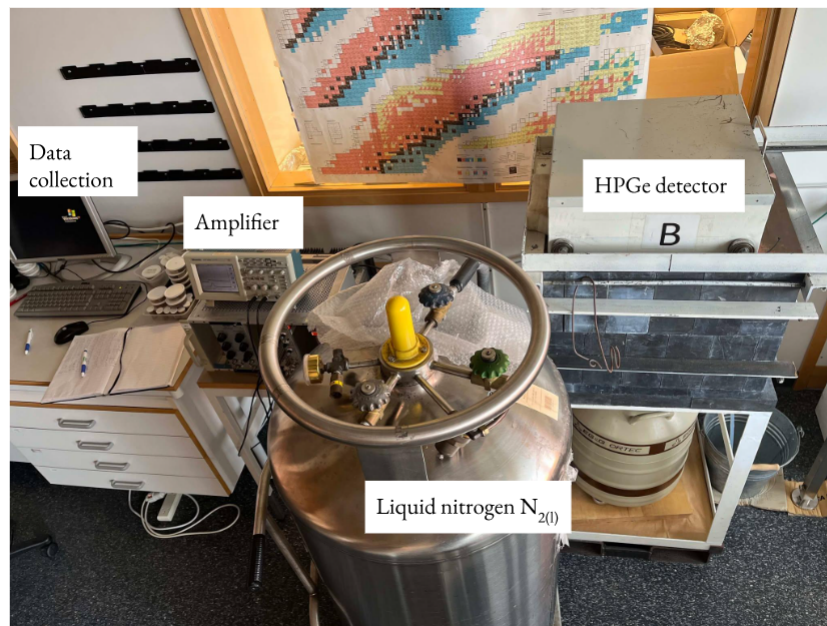


Figure 4: UGGLA setup

4 Method

The method used in this project will be presented in this section. Firstly, the decay chains investigated were chosen. After that the measurements at UGGLA took place with a ^{232}Th foil and a filter sample from the air. Data was obtained from the measurements, which was later analyzed.

4.1 Calibration

The detector was calibrated by placing a known sample in the detector. The peaks that were shown are at known energy levels and could therefore be used to calibrate which channels corresponds to which energy levels.

4.2 The chosen decay chains

The chosen decay chains to look at and analyze are the decay chains of ^{232}Th and ^{238}U . The reason for this is because ^{232}Th and ^{238}U are naturally occurring radioactive nuclides as mentioned in Section 2.2. Both nuclides are commonly found in nature, which makes them particularly relevant for investigations of atmospheric composition.

4.3 Data collection and measurements

4.3.1 ^{232}Th foil

A ^{232}Th foil was used to collect data on the mentioned nuclide. The foil's mass was measured. The foil was then placed in a plastic container before placed on the HPGe detector as centered as possible. When placed on the detector a measurement was taken with MAESTRO [5] for 102 500 s to achieve an energy spectra with energy (keV) on the x-axis and counts on the y-axis. The file got saved as a `.spe` and then imported to the software called InterSpec [13] for analysis, as seen in Figure 5.

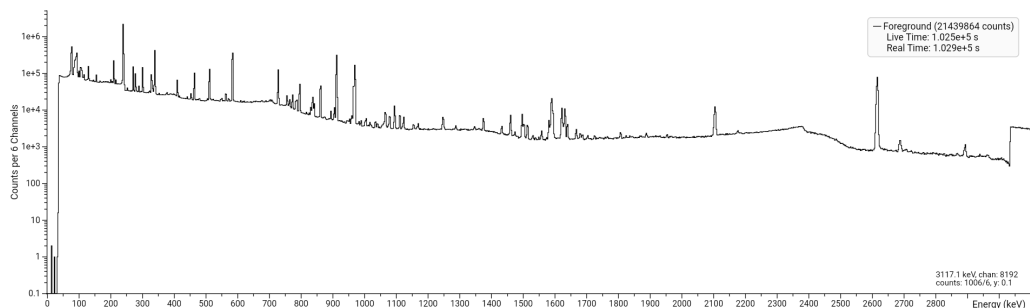


Figure 5: Example spectrum in InterSpec

4.3.2 Background radiation

A background measurement was performed within the same days as the measurement of the ^{232}Th foil. During this measurement, no sample was placed on the detector to ensure that only ambient background radiation was recorded. The background spectrum was collected over a total live time of 587 800 s. The background spectrum was saved in the same way and imported into InterSpec to be able to subtract the background from the measurement. How it was normalized and subtracted is further explained in Section 4.5.2.

4.3.3 Air filter

For the third measurement, an air pump was placed on the roof of the Ångström Laboratory. A filter was mounted in front of the air pump to collect airborne dust. The pump, with the filter in place, operated continuously for one week before being shut off. Immediately after the pump was turned off, the filter was transported to UGGLA, where it was cut into several pieces and placed in a plastic container. This container was then positioned on the HPGe detector. The time interval between shutting off the pump and placing the sample on the detector was recorded to correct for any time-related measurement offsets. A script was used to acquire several files with different lengths. The reason for this is that there are some nuclides that have a short half life and therefore can only be seen in early measurements. These files were one after the other imported to InterSpec for analysis.

4.4 Spectra analysis

4.4.1 ^{232}Th foil

When analyzing the spectrum in InterSpec for the ^{232}Th foil, the nuclides in ^{232}Th decay chain were in focus. The energy levels of the gamma-rays emitted from the different decays were found in the IAEA isotope browser [11]. All of the nuclides in the decay chain for ^{232}Th were analyzed and their corresponding peaks were identified and marked in InterSpec. When a peak is marked in InterSpec, some data is retrieved in the program which will be explained in Section 4.5.1. The data from the peaks was saved as `.csv` files directly from the software.

4.4.2 Background radiation

In the analysis of the background spectrum, all noticeable peaks were highlighted and the data of those peaks was saved as a `.csv` file. There was no identification of any nuclides because the subtraction was made only by looking at which energy the peak was located. If the peaks in the background overlapped with peaks in the spectra from the measurements, it was normalized and subtracted from the spectra.

4.4.3 Air filter

The air filter sample was analyzed in a similar method as the one above. The differences were the nuclides which were analyzed. In the case of the air filter, several nuclides were in focus. The nuclides in the decay chains of ^{232}Th and ^{238}U were looked at, and also ^7Be , ^{40}K and ^{137}Cs . There was only one spectrum in the spectra analysis in the case of the ^{232}Th foil, whereas for the air filter sample, there were several spectra saved. Fifteen spectra were saved, the live time and total time of each saved spectrum is shown in table 1. The difference between live time, i.e. the time that the data acquisition was active, and real time, was negligible. The rows that say " ^{232}Th " and " ^{238}U " are indicating which spectras that were used for the data analysis of the nuclides from the different decay chains.

Spectrum	1	2	3	4	5	6	7
Live time [s]	700	1800	3600	3600	3600	3600	1800
Total time [min]	11.7	41.7	101.7	161.7	221.7	281.7	311.7
²³² Th	X	X	X	X	X	X	X
²³⁸ U	X	X	X	X	X	X	

(a) First half of the data

8	9	10	11	12	13	14	15
1800	36000	1800	1800	86400	86400	1800	1800
341.7	941.7	971.7	1001.7	2441.7	3881.7	3911.7	3941.7
	X	X		X			

(b) Second half of the data

Table 1: The saved spectra with corresponding live time

4.5 Data analysis

4.5.1 ²³²Th foil

The analysis was made in a `python` environment and the full code can be found in section E.1 in the Appendix. The `.csv` files, which contained the information about the peaks, were imported into the software. The python library `pandas` was used to read the files and organize them. The three columns of information retrieved and used from the peak data were the columns that had information on which energy the peak was formed, the net area of the peak and the error of the net area from the software. The activity was calculated with the information from every peak for the different nuclides in the decay chain and with the efficiency of the detector. The efficiency of the detector for the different energy levels can be found in section A in the appendix. The efficiency of the detector was simulated using the Monte-Carlo code FLUKA [14, 15, 16]. The activity was calculated with equation:

$$A = \frac{N}{tI\epsilon} \quad (18)$$

where N is the number of events counted by the MCA in the peak region (net area), t is the live time of the measurement, I is the probability for the emission of gamma with a certain energy and ϵ is the efficiency of the detector

at the energy level. Some of the peaks were close to a background peak and the background peak were therefore subtracted from the peak as explained further down in Section 4.5.2. This is an alternative way to calculate the activity when there is a possibility to count the gamma-rays. Every event is one gamma-ray and therefore one decay. The total amount of events get divided by time to get decays per second. The I and ϵ works as scaling factors to scale the activity to be able to compare it to the activity deduced by the weight of the foil.

The activity of the ^{232}Th was calculated using equation 1 with $N = \frac{m_{\text{Th}}N_{\text{A}}}{M_{\text{Th}}}$ which made the equation look like:

$$A_{\text{Th}} = \lambda N = \lambda \frac{m_{\text{Th}}N_{\text{A}}}{M_{\text{Th}}} \quad (19)$$

where m_{Th} is the mass of the foil, N_{A} is the Avogadro constant and M_{Th} is the molar mass of ^{232}Th . Due to the very long half-life of ^{232}Th we can safely assume this equation is valid.

The error of the activity was also calculated for each peak using equation 14. When the activity with error was calculated, a Grubb's test was done to detect any outliers in the data of ^{228}Ac . This could be done because the data could be considered a normal population and had enough data points. To do this, equation 13 and the table in Figure 21, found in Section B in the Appendix, was used.

To be able to calculate the efficiency of the detector, secular equilibrium was assumed. Equation 18 was rearranged to look like $\epsilon = \frac{N}{tIA_{\text{Th}}}$ where A_{Th} is the activity of ^{232}Th deduced from its weight. The simulated efficiency curve for the detector was scaled to fit the data for ^{228}Ac by calculating a mean scaling factor from all the data points. This scaling factor was then applied to analyze the peaks of the other nuclides in the decay chain by multiplying it with their respective activities. A mean value for the activity for the different nuclides were then calculated and plotted comparing the means to the activity of ^{232}Th . This mean value was calculated using a function in numpy called `numpy.mean`.

4.5.2 Background radiation

All of the peaks, found in the background spectrum, were compared to all the peaks found in other spectra with respect to the energy. If the peaks

were on top of each other, with a tolerance of 1 keV, the activity would be calculated with Equation 20

$$A = \frac{N - CPS_{BG}t}{tI\epsilon} \quad (20)$$

where CPS_{BG} is the count rate for the peak in the background that is being subtracted.

4.5.3 Air filter

The analysis of the air filter measurement was also made in a `python` environment and the full code can be found in Section E.2 in the Appendix. The files were imported and read in the same way as done with the files from the ^{232}Th foil. The activity for each peak identified was calculated using Equation 18 or Equation 20, depending on if there was a background peak within the tolerance, with the alteration that the efficiency was multiplied with the scaling factor found from the ^{232}Th foil measurement. The activity for each nuclide was plotted over time and the equations from Section 2.1.1 were used to fit a theoretical curve upon the values. For the parent nuclide, Equation 2 was used. For the daughter nuclide, Equation 5 was used and Equation 6 was used for the granddaughter nuclide. They were defined as `parent_decay`, `daughter_decay` and `granddaughter_decay`. The equations in Section 2.1.2 was used to obtain relative values for the number of nuclides trapped in the filter at $t = 0$. The assumption of steady state was made i.e. the change in number of nuclides is zero ($\frac{dN}{dt} = 0$) and the assumption that it is the same production from the air into the filter for all nuclides ($P = D = G$).

The data points were placed in every time interval using equation 17. For the placing of the data points, $A(t) = A_0 \cdot e^{-\lambda t}$, was used. This was done for ^{212}Pb in ^{232}Th 's decay chain and for ^{214}Pb in ^{238}U 's decay chain. The data points for the nuclides after lead were placed at the same t as lead. A small correction should be added to t for the decay products after lead, but it is not included in this work.

5 Results

The results of this project are divided into three parts. Firstly, the results from the background radiation measurement will be presented. Secondly, the measurement and analysis of the ^{232}Th foil will be presented. Lastly, the results coming from the air filter measurement and analysis will be presented.

5.1 Background radiation

The spectrum from the background radiation is shown in Figure 6. The marked peaks are shown in Figure 7

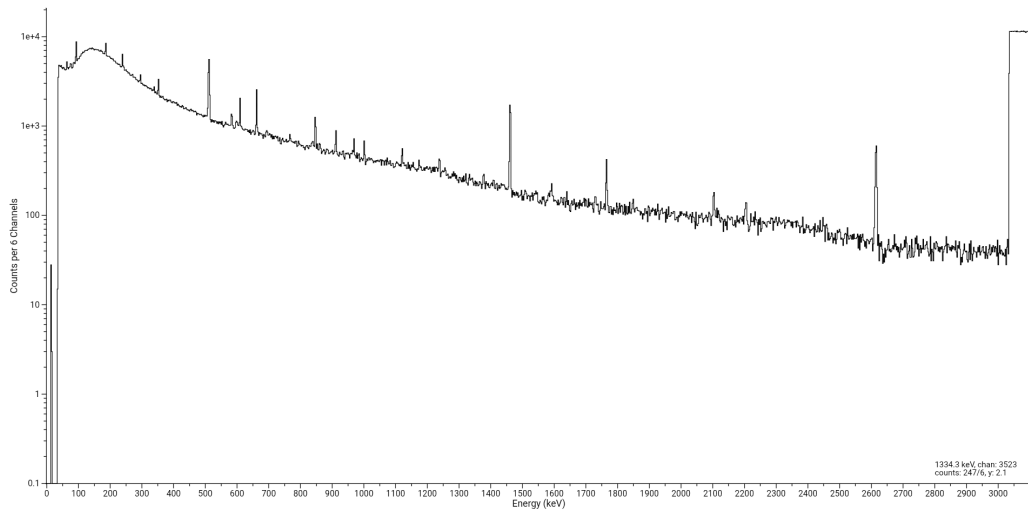


Figure 6: Gamma-ray spectrum of background radiation measured for 587800 s (around one week)



Figure 7: Gamma-ray spectrum of background radiation with marked peaks

The data from the marked peaks in the background spectrum can be found in Table 4 in the Appendix D.1.

5.2 ^{232}Th foil results

5.2.1 Spectrum analysis

The data collected from the HPGe detector came out to the spectrum in Figure 8.

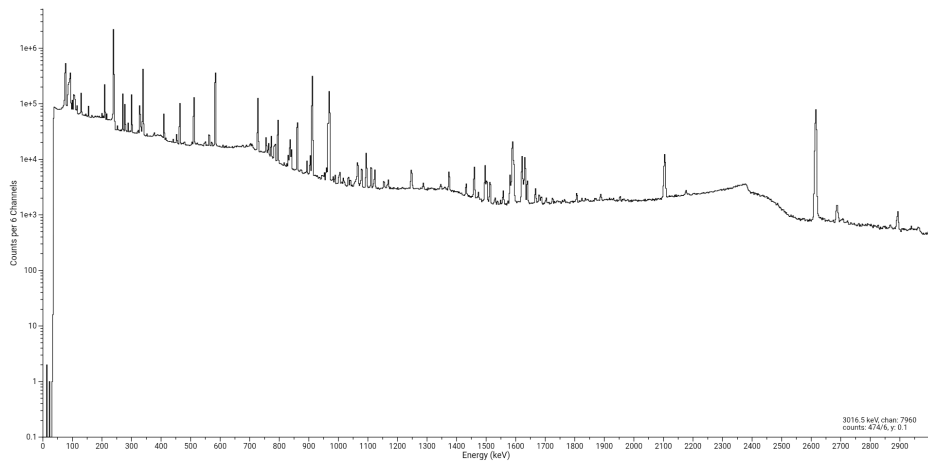


Figure 8: Gamma-ray spectrum from the ^{232}Th foil measured for 102500 s (around one day)

The spectrum presented in Figure 8 was analyzed looking at the nuclides in the decay chain of ^{232}Th . The nuclides identified from the decay chain were ^{228}Ac (blue), ^{224}Ra (green), ^{212}Pb (red), ^{212}Bi (orange) and ^{208}Tl (purple). The peaks from the different nuclides can be shown with their corresponding colors in Figure 9.

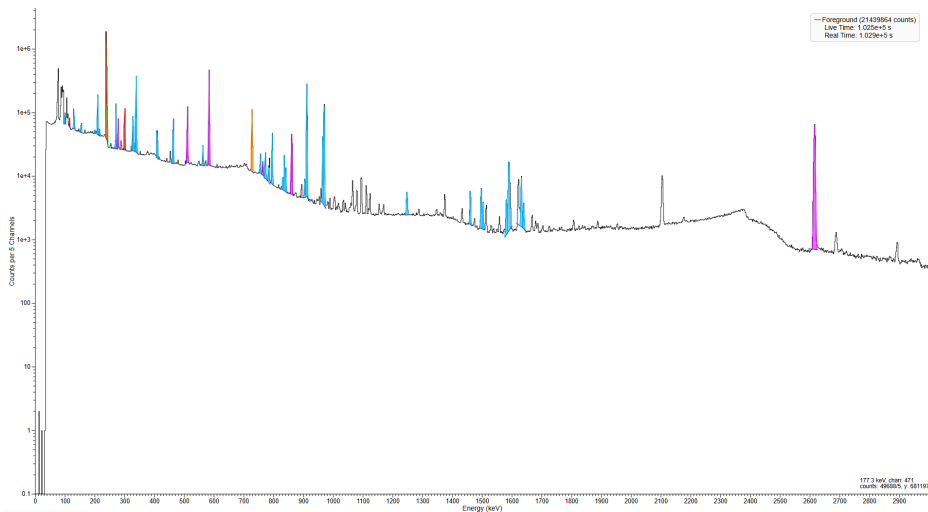
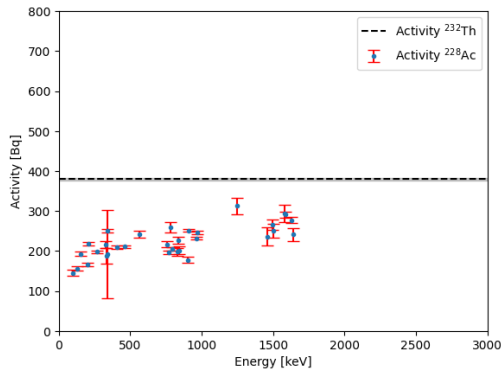


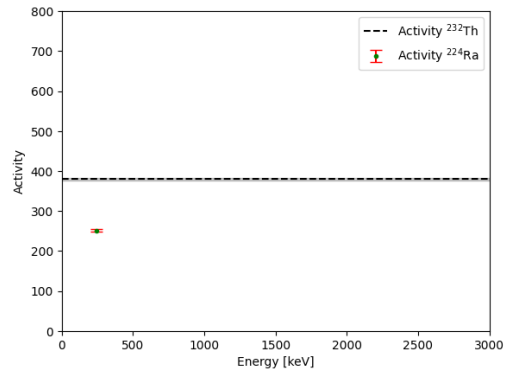
Figure 9: Gamma-ray spectrum from the ^{232}Th foil with the nuclides identified

5.2.2 Nuclides activity

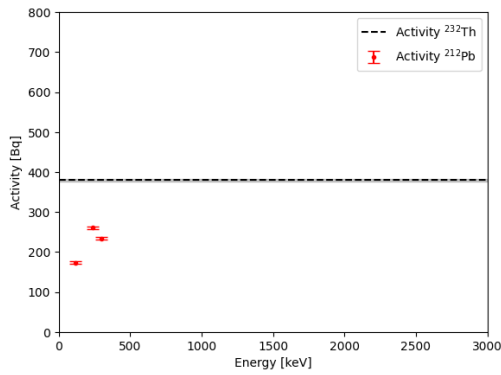
The foil's mass was measured to be $m_{Th} = 0.0932$ g and its activity was calculated to be around $A_{Th} = 379.51$ Bq. The activity of each peak for the different nuclides is presented in Figure 10. The Grubbs' test resulted in founding no outlier in the set of data points from ^{228}Ac . The activity of ^{232}Th is shown as a line in the Figure to compare the activity of nuclides to the activity of ^{232}Th . This is the result of using equation 18 with the data extracted from InterSpec [13].



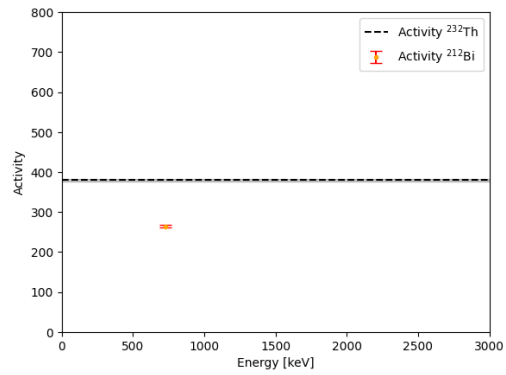
(a) Activity of ^{228}Ac against energy



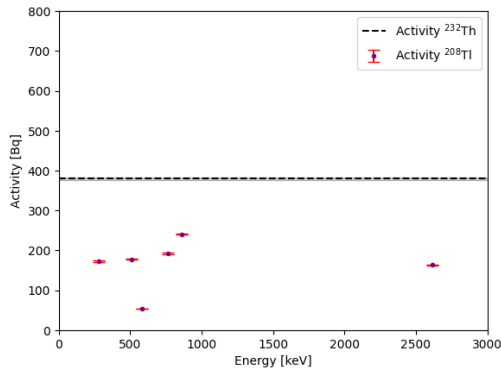
(b) Activity of ^{224}Ra against energy



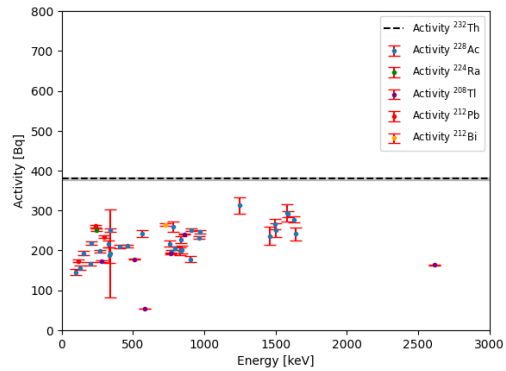
(c) Activity of ^{212}Pb against energy



(d) Activity of ^{212}Bi against energy



(e) Activity of ^{208}Tl against energy



(f) All the activities

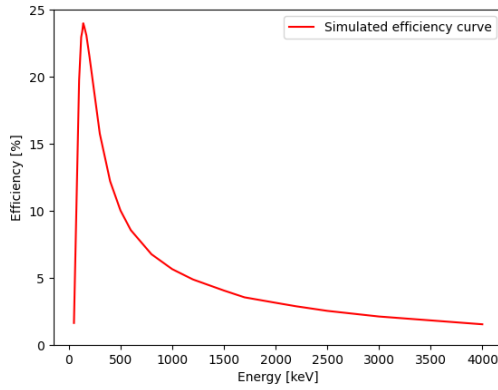
Figure 10: The activity of the identified nuclides compared to the calculated activity of ^{232}Th

5.2.3 HPGe detector efficiency assuming secular equilibrium

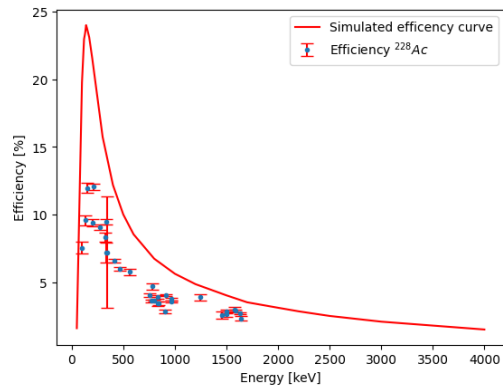
To be able to calculate how long it would take the foil to reach secular equilibrium, By combining Equation 5 and Equation 2, and applying the conditions $A_P = A_D$ and $A_{D0} = 0$, Equation 21 was derived.

$$A_{P0} \cdot e^{-\lambda_P t} = A_{P0} \cdot \left(\frac{\lambda_P}{\lambda_D - \lambda_P} \right) \cdot (e^{-\lambda_P t} - e^{-\lambda_D t}) \quad (21)$$

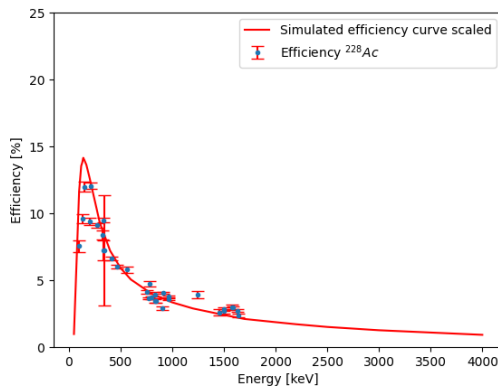
When solving for t , it came out to approximately 179 years, which is the time it would take to reach secular equilibrium from being only ^{232}Th . For a secular equilibrium of at least 99%, it was calculated to be about 38 years. The result of assuming secular equilibrium can be seen in subfigure 11b. The figure shows the efficiency calculated from the data of ^{228}Ac compared with the efficiency curve simulated with FLUKA plotted by itself in subfigure 11a. The scaling factor was calculated to be approximately 0.5867 ± 0.0054 . Subfigure 11c shows the resulting efficiency curve when multiplied with the calculated scaling factor. Subfigure 11d compares the original simulated efficiency curve with the scaled one.



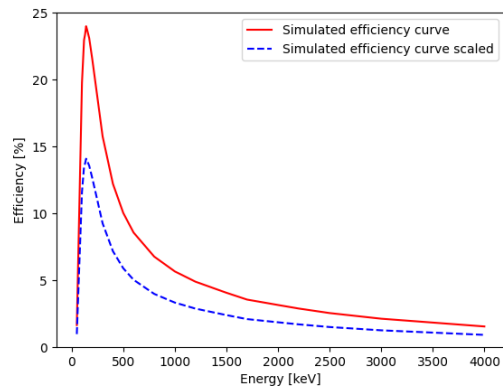
(a) Simulated efficiency curve of the HPGe detector



(b) Efficiency of ^{228}Ac compared to the efficiency curve



(c) The efficiency curve scaled to fit the ^{228}Ac data

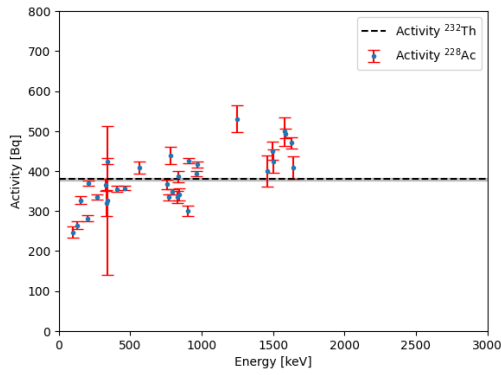


(d) The simulated efficiency curve and the scaled efficiency curve

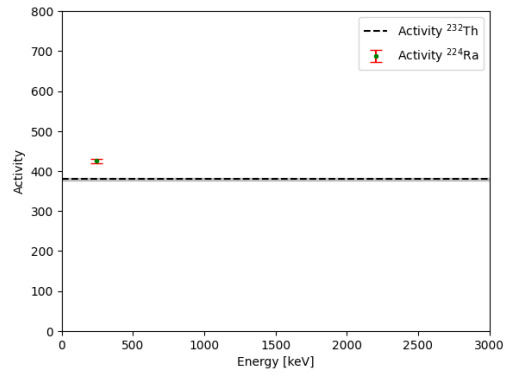
Figure 11: Plots of the simulated efficiency curve and the scaled efficiency curve with data from ^{228}Ac

5.2.4 Nuclides activity scaled

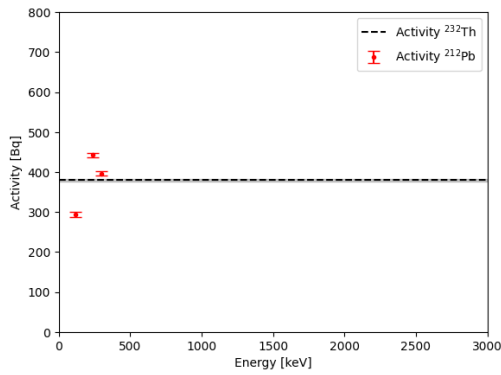
This section compiles the activity of the different nuclides when multiplied with the scaling factor derived in Section 5.2.3.



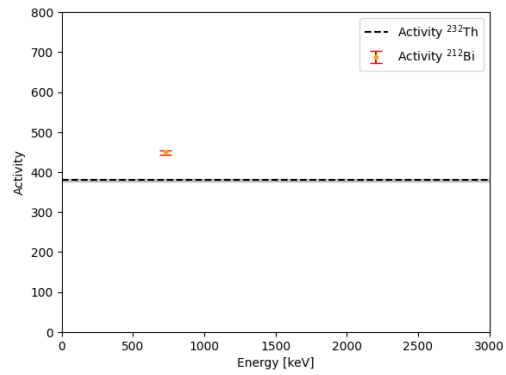
(a) Activity of ^{228}Ac



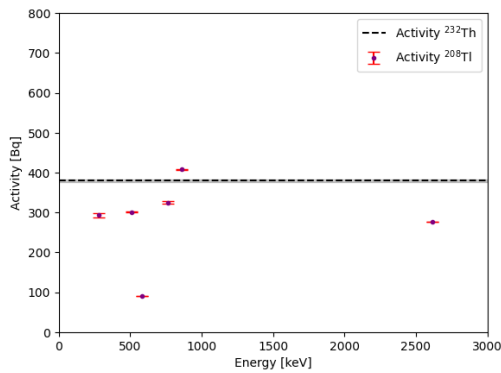
(b) Activity of ^{224}Ra



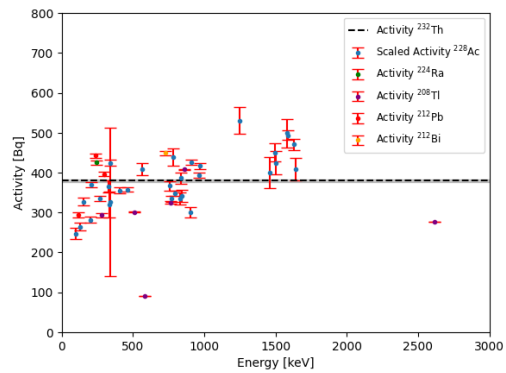
(c) Activity of ^{212}Pb



(d) Activity of ^{212}Bi



(e) Activity of ^{208}Tl

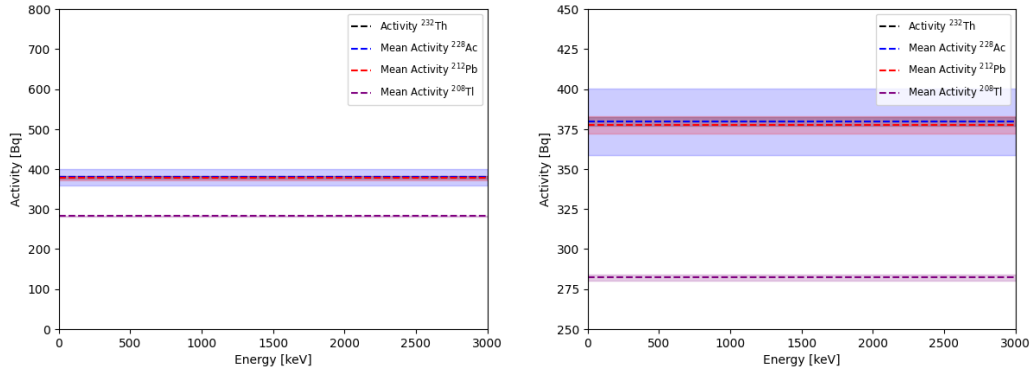


(f) Every nuclides' activity

Figure 12: Plots that showcase the activity of the different nuclides multiplied with the scaling factor

5.2.5 Mean values of the scaled nuclides activity

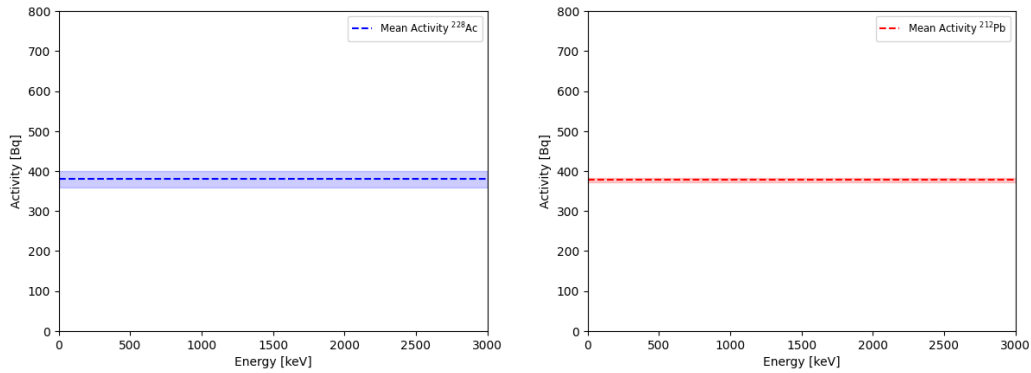
The means that were calculated from the data points for the different nuclides are shown in Figure 13. The means of ^{228}Ac and ^{212}Pb are very close to each other and to the activity of ^{232}Th , which makes them hard to distinguish and are therefore shown on their own in figure 14. The means were calculated to be around: $A_{Ac-228,Mean} = 379.51 \pm 20.752$ Bq, $A_{Pb-212,Mean} = 377.39 \pm 5.3312$ Bq and $A_{Tl-208,Mean} = 282.16 \pm 1.7931$ Bq. As mentioned before, the activity deduced from the weight of the foil was $A_{Th} = 379.51$ Bq.



(a) Means of the different nuclides' activities

(b) A zoomed in graph of the means of the different nuclides' activities

Figure 13: Two graphs of the calculated means of the nuclides's activities, one of the zoomed in on the same result



(a) Mean activity of ^{228}Ac

(b) Mean activity of ^{212}Pb

Figure 14: Mean activities of ^{228}Ac and ^{212}Pb in their own graphs

5.3 Air filter results

5.3.1 Spectra analysis

The spectra in Figures 15 and 16 displays the analyzed spectra that were first saved with a live time of 700 s. When looking at the nuclides in the decay chain of ^{232}Th , that is: ^{212}Pb (red), ^{212}Bi (orange) and ^{208}Tl (pink). They were identified as one peak, one peak and four peaks respectively. Those peaks are shown in Figure 15. One peak was identified to be from ^7Be (yellow) which also is shown in Figure 15. In figure 16 the nuclides from the decay chain of ^{238}U were analyzed. The nuclides identified were ^{214}Pb (blue) and ^{214}Bi (green) with four and sixteen peaks respectively.

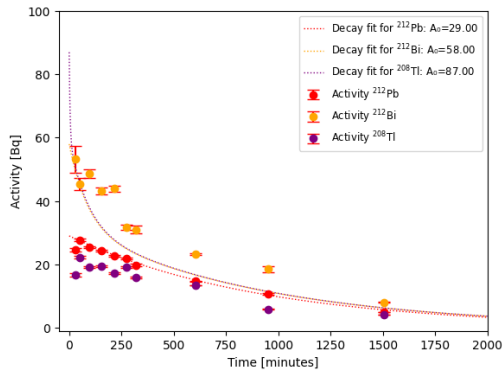
The rest of the spectra analyzed can be found in Section C in the Appendix.

dotted lines are the curve fits appointed for the activity decay following the equations found in Section 2.1.1. The relative activities of the nuclides at the time the air collection was stopped ($t=0$) were calculated to be 1 for ^{212}Pb , 2 for ^{212}Bi and 3 for ^{208}Tl after correcting for the branching ratio of the decay chain when calculating the activities. In Figure 18 the same activity is presented but with a logarithmic scale on the y-axis. The activity of ^7Be is presented in Figure 19.

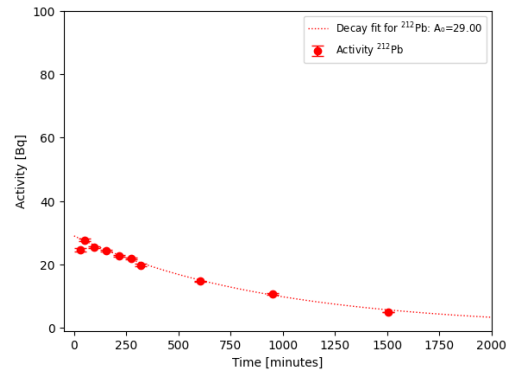
The data points for each nuclide are only from one of the peaks identified. The peaks used for the analysis of the activity decay over time can be seen in Table 2.

Nuclide	^7Be	^{212}Pb	^{212}Bi	^{208}Tl
Energy [keV]	478	239	728	2615

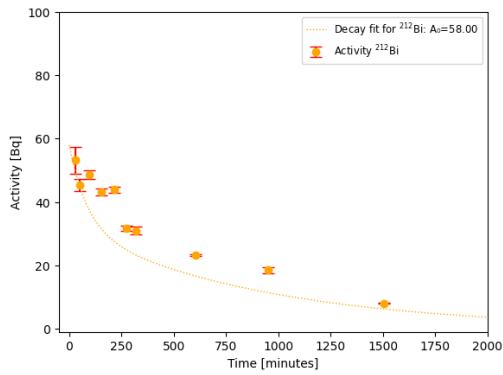
Table 2: The peaks that were followed over time



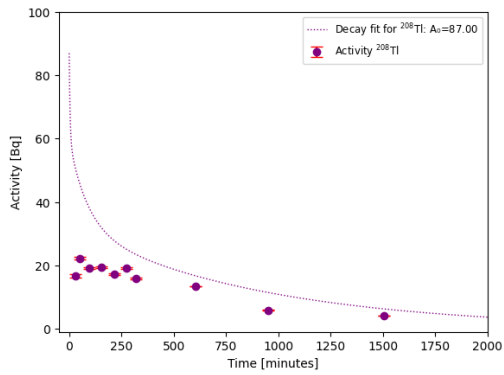
(a) Activity for all nuclides



(b) Activity of ^{212}Pb over time

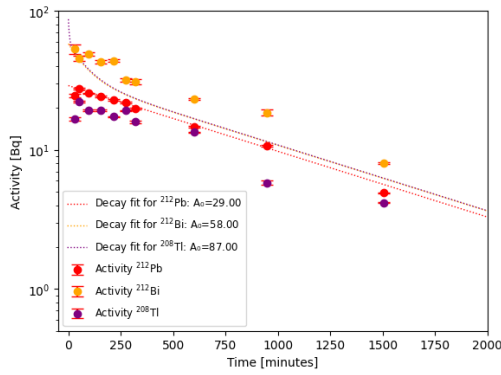


(c) Activity of ^{212}Bi over time

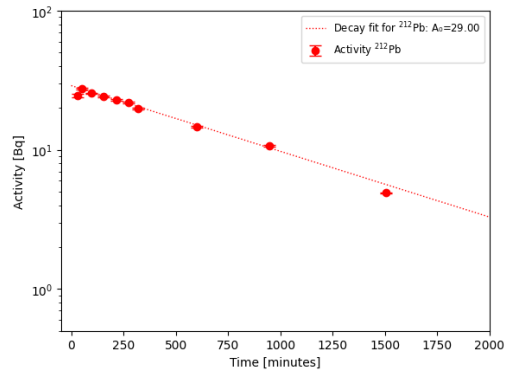


(d) Activity of ^{208}Tl over time

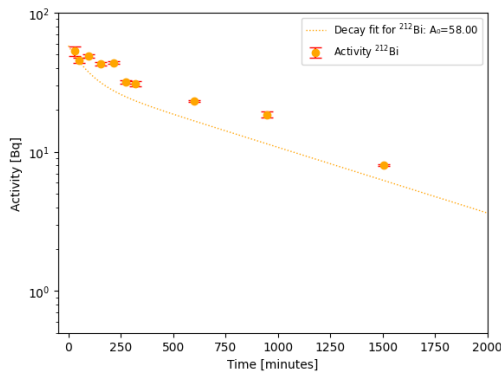
Figure 17: Plots showcasing the activity of ^{212}Pb , ^{212}Bi and ^{208}Tl over time with corresponding decay fits



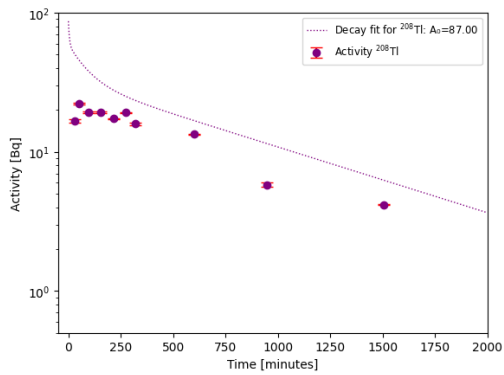
(a) Activity for all nuclides in logarithmic scale



(b) Activity of ^{212}Pb over time in logarithmic scale

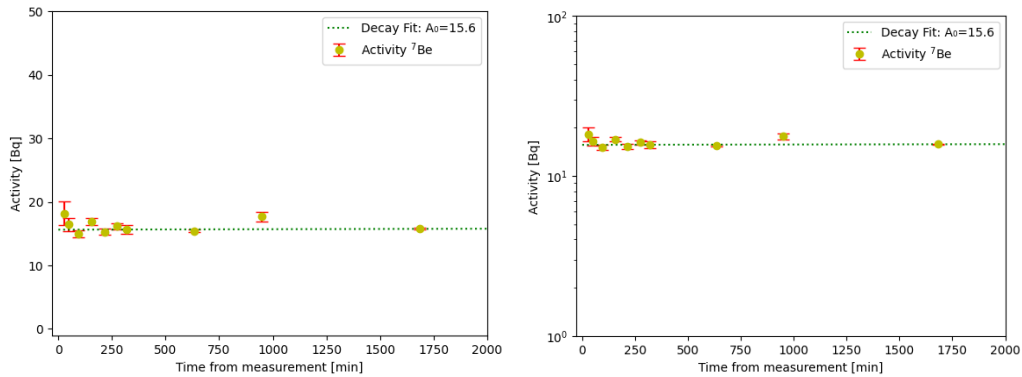


(c) Activity of ^{212}Bi over time in logarithmic scale



(d) Activity of ^{208}Tl over time in logarithmic scale

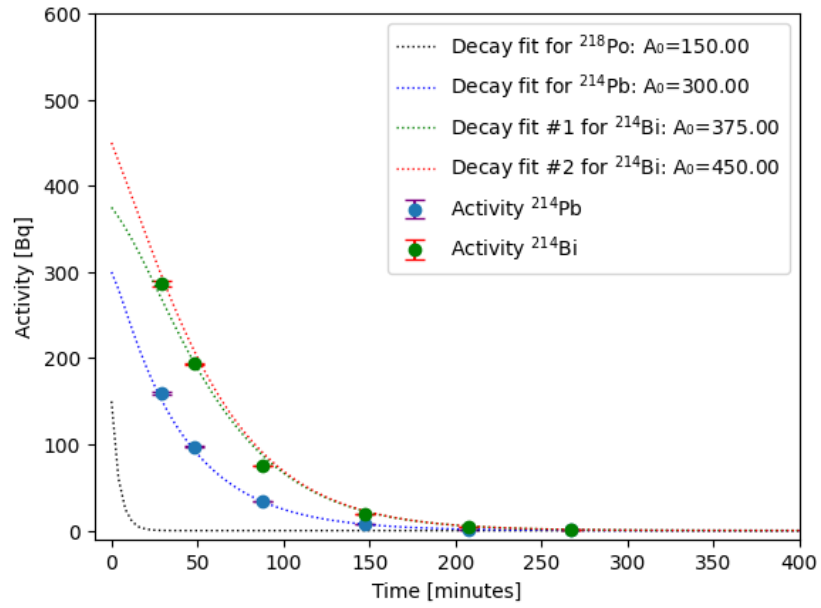
Figure 18: Plots showcasing the activity of ^{212}Pb , ^{212}Bi and ^{208}Tl over time with corresponding decay fits, with a logarithmic scale on the y-axis



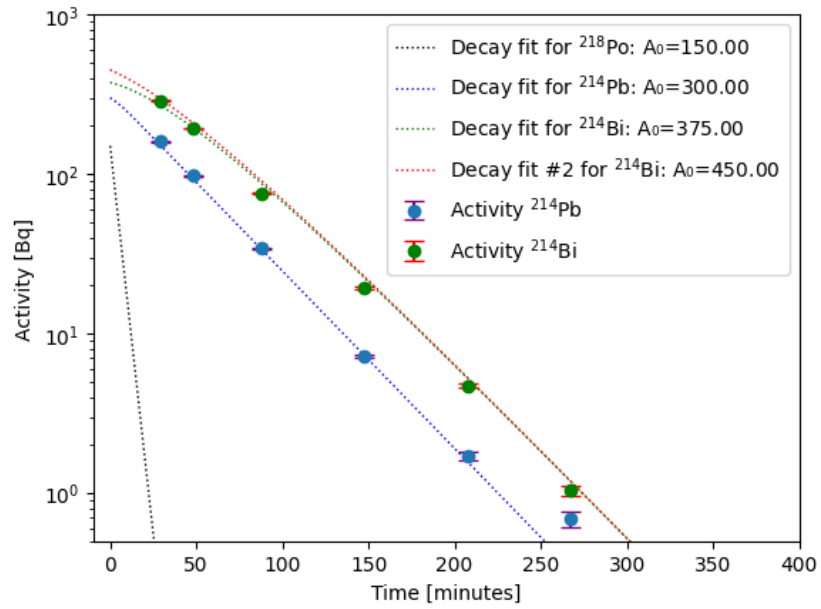
(a) Activity of ${}^7\text{Be}$ over time in linear scale (b) Activity of ${}^7\text{Be}$ over time in logarithmic scale

Figure 19: Plots showcasing the activity of ${}^7\text{Be}$ over time in linear and logarithmic scale with corresponding decay fit

The activities over time of the nuclides found in the decay chain of ${}^{238}\text{U}$ are shown in Figure 20. The analyzed peaks were at around 352 keV for ${}^{214}\text{Pb}$ and 610 keV for ${}^{214}\text{Bi}$. The dotted lines represents the decay from the equations found in Section 2.1.1. The relation between the activities of nuclides was calculated to be a 1:2:3 relationship for ${}^{218}\text{Po}$, ${}^{214}\text{Pb}$ and ${}^{214}\text{Bi}$. Decay fit #1 used a 1:2:2.5 relationship and decay fit #2 used the calculated 1:2:3 relationship.



(a) Measured activity of ^{214}Pb and ^{214}Bi with fitted decay curves.



(b) Same as above but with a logarithmic y-axis.

Figure 20: The activity of ^{214}Pb and ^{214}Bi on linear and logarithmic scales.

6 Discussion

In this section all the results of the previous section will be discussed, if they are reasonable or unreasonable and if so, why. Firstly the results of the measurement of the ^{232}Th foil will be discussed and then the results from the air filter measurement.

6.1 Discussion of ^{232}Th foil results

6.1.1 Spectrum analysis for the ^{232}Th foil

The nuclides found in the spectrum analysis are all nuclides that are known to be in the decay chain of ^{232}Th according to IAEA isotope browser [11]. The peaks that are highlighted in InterSpec [13] are not taking every event counted by the MCA into account. The software fits a normal distribution on top of the peak which also can be manually edited to try to fit the peak. This brings an error into the net area which also is calculated by the software. This can lead to the counts not being completely accurate.

6.1.2 Nuclide activity for the ^{232}Th foil

When calculating the activity of each peak for the different nuclide the result is that they are beneath the activity of ^{232}Th . Because its consistent for the three nuclides looked at, it would imply that something in equation 18 is off. N is the net area calculated from InterSpec [13] which discussed before can err, but it should not make this big of a difference to the activity. The error is too small to give this result as seen in Section D.2 in the Appendix. I is read out directly from IAEA [11] and this leads to the efficiency probably being the reason for this result of low activity. It can also imply that it has not reached secular equilibrium yet which would mean that this foil is very young. This does not seem plausible also because we see a pretty big amount of ^{208}Tl in the sample which indicates that decay chain has reached secular equilibrium. The peak at 583 keV for ^{208}Tl is very low compared to the rest of the data. Because the probability is very high for this peak (85%), the error from true coincidence will be greater which can explain the low activity.

6.1.3 HPGe detector efficiency assuming secular equilibrium

To be able to assume secular equilibrium, $t_{1/2_P} \gg t_{1/2_D}$ must hold [3]. In the case of the ^{232}Th with its daughter nuclides, it does. The time passed to practically reach secular equilibrium was approximately 38 years which is not so long. The foil is known to probably be older than that, and with that concludes that the decay chain should have reached secular equilibrium.

The scaled efficiency curve was scaled from the simulated efficiency curve with a factor of about 0.586. This number does seem low. On the other hand, as mentioned by Mattias Lantz (private communication, June 12, 2025) similar discrepancies has been found in other comparisons between FLUKA and measurements with UGGLA.

6.1.4 Nuclides activity scaled and mean values

The scaling factor was calculated using the data from the peaks identified as ^{228}Ac . So this leads to that the activities of ^{228}Ac will be around the activity of ^{232}Th . The activities of ^{212}Pb are however also around the activity of ^{232}Th which implies for a lower efficiency of the detector. For ^{208}Tl we see that it is found to be below the line and that can be explained by either: true coincidence or that the decay chain has not reached secular equilibrium through the whole decay chain.

6.2 Discussion of the Air filter results

6.2.1 Spectra analysis for the air filter

All the nuclides identified in the spectra from the air filter are reasonable to identify in the atmosphere. Both ^{212}Pb and ^{208}Tl are after ^{220}Rn in the decay chain of ^{232}Th which implies that there are radon in the air which decayed to the other nuclides. Because no nuclides before radon were identified in the decay chain, it implies that no visible amounts of dust from ^{232}Th resides in the air. The same argument can be held for the nuclides in the decay chain of ^{238}U . They are also after ^{222}Rn in the decay chain. The identification of ^7Be can be explained by high energy cosmic ray interactions in the atmosphere [17].

6.2.2 Nuclide activity for the air filter

The data points used are from measurements that are of different lengths. The data points are put at a weighted average time depending on the half-life of the nuclide as seen in Equation 17. This should provide a more accurate representation of the activity over time. On the other hand, the data points from nuclides after lead in each decay chain, were put at the same t as lead. This was done because of time shortage and can be improved by using the correct $A(t)$ for all the nuclides in the chain. That could lead to a more accurate representation of the data.

The scaled efficiency curve which is shown in Figure 11d is the one that was used when calculating the activity for the nuclides found in the air filter. The filter did not have the same geometry as the thorium foil which will alter the efficiency. The efficiency will probably be around half of the one used for the thorium foil. The absolute values are not what's most important here, but rather the behavior of the decay over time.

The activity for the peaks followed in ^{232}Th 's decay chain, at $t = 0$, are much lower than those of ^{238}U . This outcome can be explained by the half-lives of the different isotopes of radon. ^{220}Rn has a half-life of 55.6 s [11] which implies that it does not always reach the air before it has decayed into something else. This leaves the abundance of ^{220}Rn to be low. ^{222}Rn on the other hand has a half-life of about 3.8 days. This leads to it being able to reach the air and spread before it is decayed into something else. This increases the abundance of it in the air. Therefore there is a higher activity of the nuclides in ^{238}U 's decay chain.

The reason that ^{212}Pb and ^{208}Tl are still left after 1000 min is because of the long half-life of ^{212}Pb . It is about 10.6 h [11] which will make it linger longer in the filter. ^{208}Tl on the other hand has a short half-life at about 3 min. The reason it lingers is because it is after ^{212}Pb in the decay chain. This leads to it growing slowly from the decay of ^{212}Pb . The opposite is correct for ^{214}Pb and ^{214}Bi . Their half-lives are about 27 min and 20 min respectively [11]. This leads to them almost not being around already after 200 min.

The decay fits for the nuclides in ^{232}Th 's decay chain does not look so accurate. The assumptions that it is the same production term for the activity is probably faulty in this case. How many nuclides that get stuck in the filter can vary. They can get stuck on other particles such as dust in the air but may also have different probabilities to actually get stuck in the filter. There

has not been a correction for true coincidence summing in the case of ^{208}Tl . This correction should give a more accurate decay fit. The activity ^{212}Bi is a bit high which we also saw in the case of the thorium foil. The decay fits for both ^{214}Pb and ^{214}Bi appear accurate. This is likely because their production from the air can be assumed to be nearly identical. In this case, the contribution from the growth of ^{218}Po is minimal when the measurements begin, due to its short half-life. Because the growth from ^{218}Po was not accounted for in the decay fit for ^{214}Pb , the activity at $t = 0$ is probably underestimated.

The activity for ^7Be seems to be constant, which is reasonable considering that it has a half-life of about 53 days.

7 Conclusion

The conclusion that the detector is not as efficient as the FLUKA code simulated can be drawn if the assumption of secular equilibrium is holding. This should be able to hold because $t_{1/2_P} \gg t_{1/2_D}$ and 38 years should have passed since this ^{232}Th foil was made. It might be more efficient than the scaled curve because of geometric efficiency and human errors made along the way.

The nuclides identified when analyzing the spectra from the air filter can all be accounted for. They are after the step of radon in both the decay chains of ^{232}Th and ^{238}U which indicates that it was radon leaving the ground and not dust from the parent nuclides. The activity, and decay of activity, over time can be explained from the half-lives of the nuclides in the decay chains.

The curve fit that was made for the decay of the nuclides over time can be refined and improved. This will result in a more precise identification of the nuclides present in the air, as well as their concentrations. A good approximation can be said to have been achieved with the assumptions made in this project, but improvements can surely be done.

8 Future research

When this project started, the aim was not to draw conclusions about the efficiency of the HPGe detector at UGGLA. The project led there because of the results seen when handling the data from the ^{232}Th foil. Therefore it would be salutary to dedicate some study to the detector's efficiency. More measurements and tests along that line can be done and compared to the results from this project.

More filters can be examined to get a better understanding of how the graphs look and how the decay fits can be altered to get a more accurate result. The A_0 s for all the nuclides are probably more advanced and are taking more factors into account than predicted, which could be looked into more. The placement of the data points can be more accurate to get a more realistic view of the results which also can lead to a better approximation of the amount of radioactive nuclides in the atmosphere.

References

- [1] *Radiation and health*. <https://www.who.int/news-room/questions-and-answers/item/radiation-and-health>. [Accessed 08-06-2025].
- [2] *Radioactive Waste From Uranium Mining and Milling*. <https://www.epa.gov/radtown/radioactive-waste-uranium-mining-and-milling>. [Accessed 08-06-2025].
- [3] Gordon. Gilmore. *Practical gamma-ray spectrometry*. eng. 2nd ed. Chichester, England ; 2008. ISBN: 9780470861967.
- [4] Dobromir S. Pressyanov. “Short solution of the radioactive decay chain equations”. In: *American Journal of Physics* 70.4 (Apr. 2002), pp. 444–445. ISSN: 0002-9505. DOI: 10.1119/1.1427084. eprint: https://pubs.aip.org/aapt/ajp/article-pdf/70/4/444/8504277/444_1_online.pdf. URL: <https://doi.org/10.1119/1.1427084>.
- [5] ORTEC. *MAESTRO Multichannel Analyzer Emulation Software*. Version 6.0. 2008. URL: <https://www.ortec-online.com/products/software/maestro-mca>.
- [6] G.F. Knoll. *Radiation Detection and Measurement*. Wiley, 2010. ISBN: 9780470131480. URL: <https://books.google.se/books?id=4vTJ7UDe15IC>.
- [7] Frank E. Grubbs. “Sample Criteria for Testing Outlying Observations”. eng. In: *The Annals of mathematical statistics* 21.1 (1950), pp. 27–58. ISSN: 0003-4851.
- [8] Frank E. Grubbs. “Procedures for Detecting Outlying Observations in Samples”. eng. In: *Technometrics* 11.1 (1969), pp. 1–21. ISSN: 0040-1706.
- [9] Carl Nordling and Jonny Österman. *Physics handbook for science and engineering*. eng. 7., [rev.] ed. Lund: Studentlitteratur, 2004. ISBN: 9144031521.
- [10] H. Jäderström et al. “True coincidence summing correction and mathematical efficiency modeling of a well detector”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 784 (2015). Symposium on Radiation Measurements and Applications 2014 (SORMA XV), pp. 264–268. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/>

- j.nima.2014.08.032. URL: <https://www.sciencedirect.com/science/article/pii/S0168900214009619>.
- [11] IAEA. *IAEA Isotope Browser*. Version 3.57.88. URL: <https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html>.
 - [12] BRADLEY DOUGHERTY. “ON THE AVERAGE OF A FUNCTION AND THE MEAN VALUE THEOREM FOR INTEGRALS”. In: *Pi Mu Epsilon Journal* 14.4 (2016), pp. 251–254. ISSN: 0031952X. URL: <https://www.jstor.org/stable/48568127> (visited on 08/21/2025).
 - [13] Sandia National Laboratories. *InterSpec Spectral Radiation Analysis Software*. Version 1.0.13. Oct. 2, 2024. URL: <https://sandialabs.github.io/InterSpec/>.
 - [14] FLUKA Team. *FLUKA Official Website*. <https://fluka.cern>. Accessed: 2025-05-30.
 - [15] C. Ahdida et al. “New Capabilities of the FLUKA Multi-Purpose Code”. In: *Frontiers in Physics* 9 (2022).
 - [16] G. Battistoni et al. “Overview of the FLUKA code”. In: *Annals of Nuclear Energy* 82 (2015).
 - [17] A. J Cruikshank, G Cowper, and W. E Grummitt. “PRODUCTION OF Be7 IN THE ATMOSPHERE”. In: *Canadian journal of chemistry* 34.3 (1956), pp. 214–219. ISSN: 0008-4042.

A Efficiency Table for the HPGe detector

Table 3: Efficiency at different energies for the HPGe detector at UGGLA

E (keV)	Peak efficiency
50	0.01622
100	0.19752
110	0.2134
120	0.22951
130	0.2337
140	0.23989
150	0.2372
170	0.231
200	0.21515
300	0.15747
400	0.12194
500	0.10018
600	0.0855
800	0.0675
843.76	0.065073508
1000	0.05641
1014.52	0.055852432
1200	0.04873
1368.626	0.044075922
1500	0.04045
1700	0.03534677
2000	0.031304
2200	0.028663
2500	0.02526
2754.007	0.023118721
3000	0.021045
4000	0.015245

B Table of values for $T_{critical}$

4

FRANK E. GRUBBS

TABLE I

Table of Critical Values for T (One-sided Test) When Standard Deviation is Calculated from the Same Sample

Number of Observations n	5% Significance Level	2.5% Significance Level	1% Significance Level
3	1.15	1.15	1.15
4	1.46	1.48	1.49
5	1.67	1.71	1.75
6	1.82	1.89	1.94
7	1.94	2.02	2.10
8	2.03	2.13	2.22
9	2.11	2.21	2.32
10	2.18	2.29	2.41
11	2.23	2.36	2.48
12	2.29	2.41	2.55
13	2.33	2.46	2.61
14	2.37	2.51	2.66
15	2.41	2.55	2.71
16	2.44	2.59	2.75
17	2.47	2.62	2.79
18	2.50	2.65	2.82
19	2.53	2.68	2.85
20	2.56	2.71	2.88
21	2.58	2.73	2.91
22	2.60	2.76	2.94
23	2.62	2.78	2.96
24	2.64	2.80	2.99
25	2.66	2.82	3.01
30	2.75	2.91	
35	2.82	2.98	
40	2.87	3.04	
45	2.92	3.09	
50	2.96	3.13	
60	3.03	3.20	
70	3.09	3.26	
80	3.14	3.31	
90	3.18	3.35	
100	3.21	3.38	

Figure 21: Table of values for $T_{critical}$ with different number of data points and confidence intervals [8]

C Spectra from the air filter

The spectra are shown in the order they were measured.

C.1 ^{232}Th 's decay chain and ^7Be peaks

In this section, the identified peaks from the decay chain of ^{232}Th will be shown. In the same spectra, the identified peak of ^7Be also will be shown. In Figure 22, all identified peaks are shown, whereas in the subsequent spectra in this section, only the peaks tracked over time are marked.

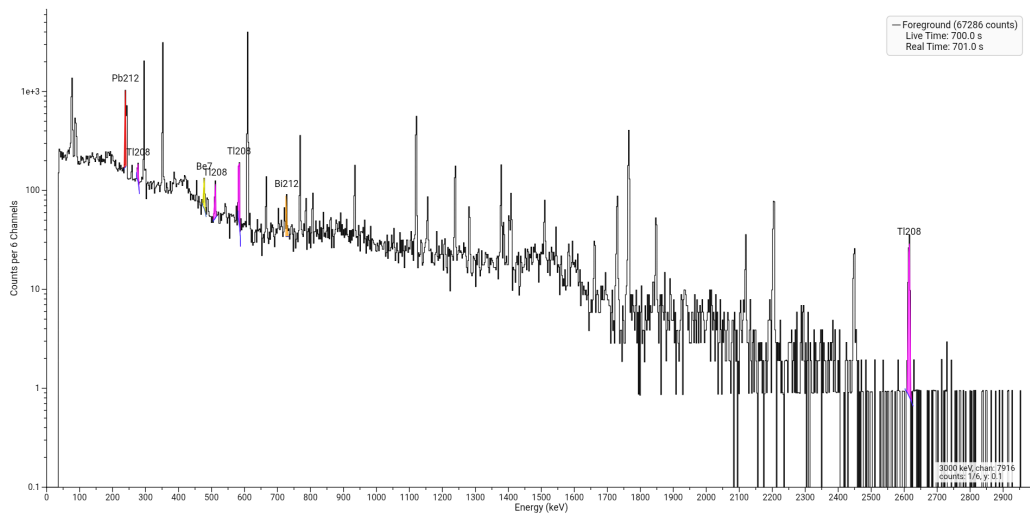


Figure 22: Spectrum 1a. 700 s measurement

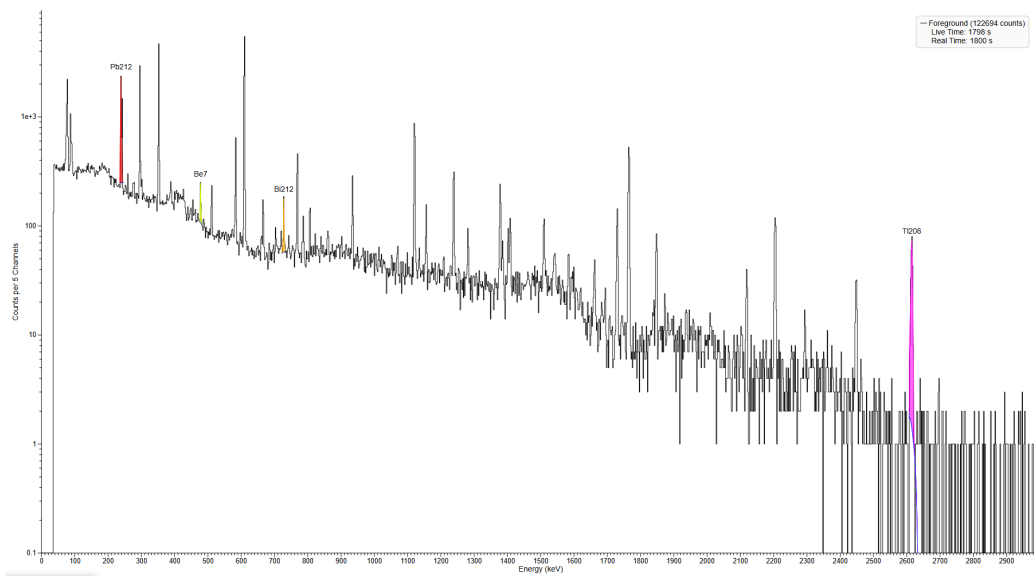


Figure 23: Spectrum 2a. 1800 s measurement

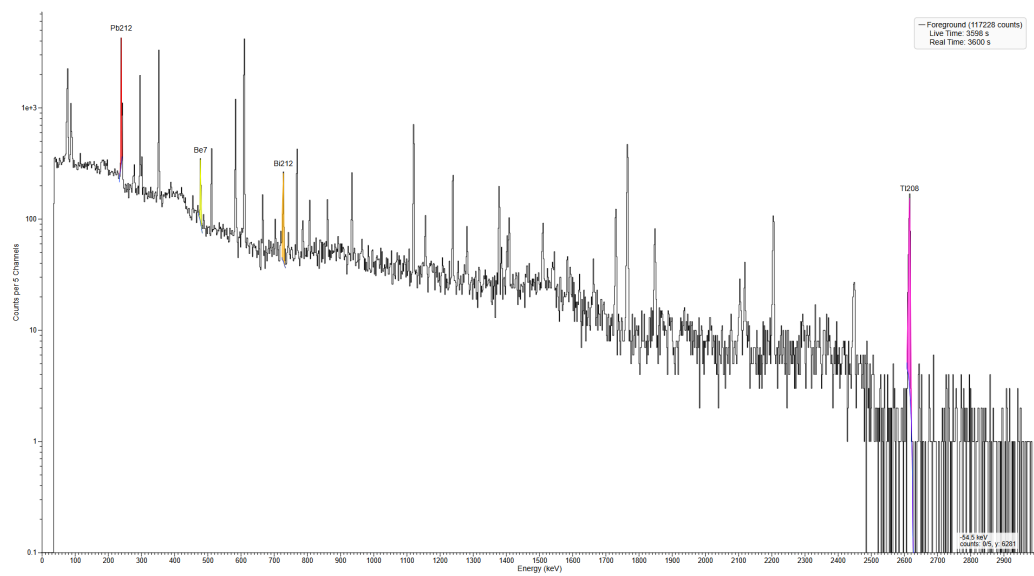


Figure 24: Spectrum 3a. 3600 s measurement

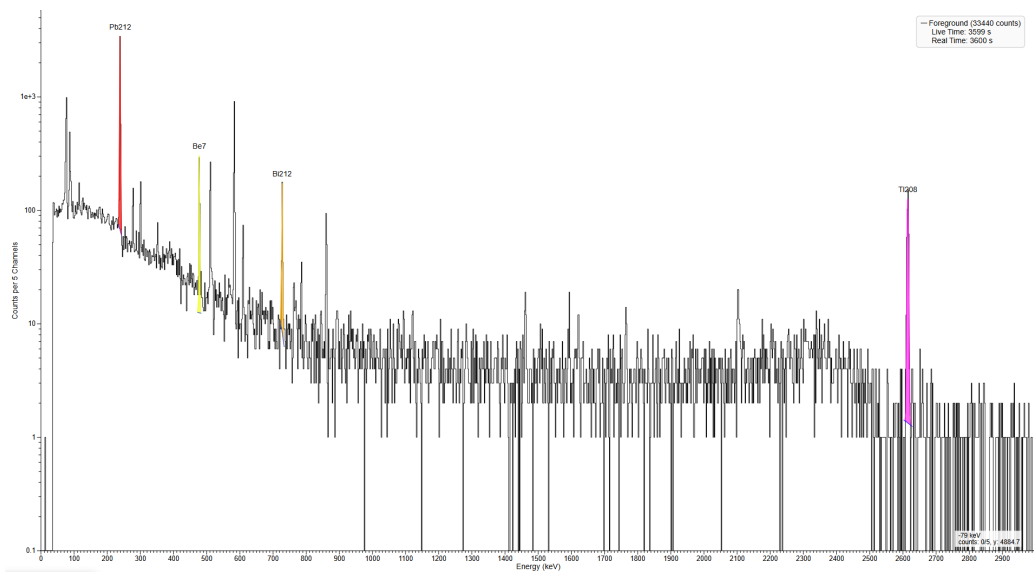


Figure 27: Spectrum 6a. 3600 s measurement

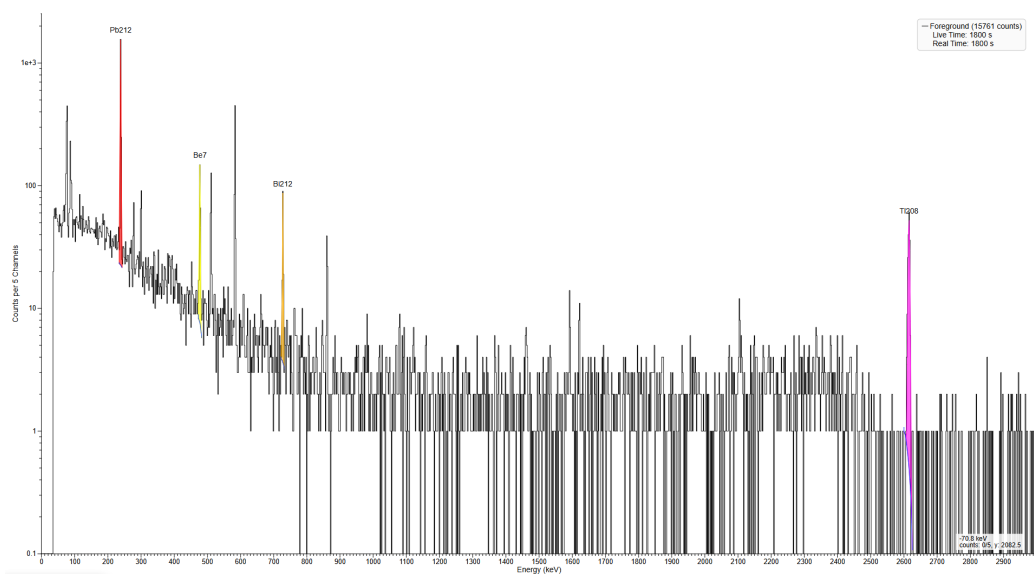


Figure 28: Spectrum 7a. 1800 s measurement

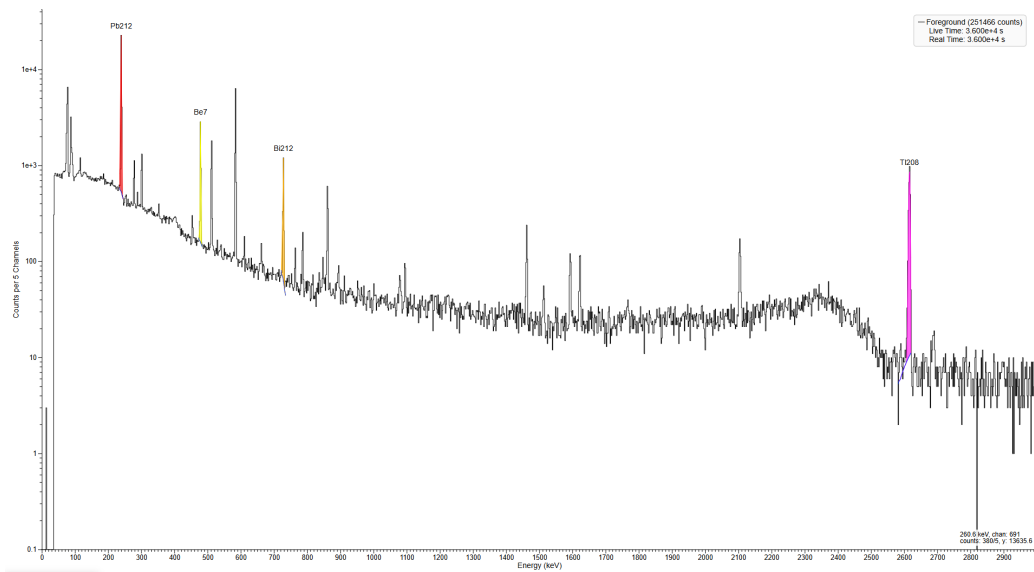


Figure 29: Spectrum 9a. 36000 s measurement

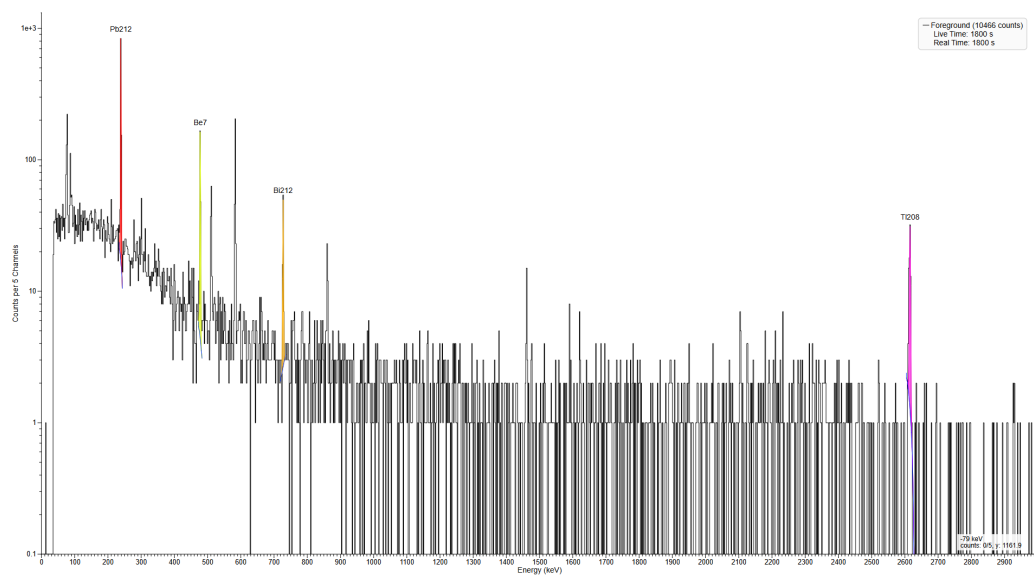


Figure 30: Spectrum 10a. 1800 s measurement

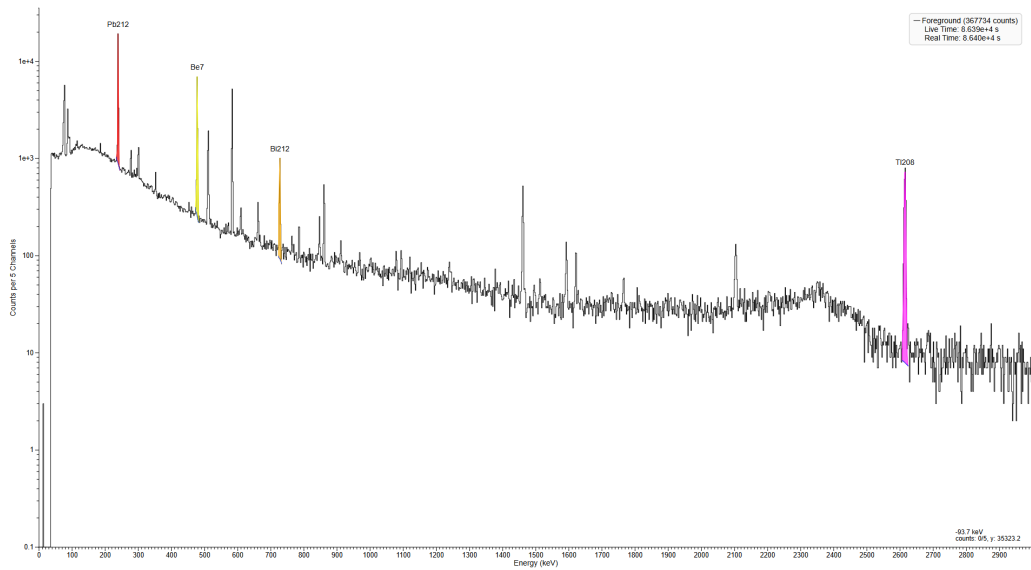


Figure 31: Spectrum 12a. 86400 s measurement

C.2 ^{238}U peaks

The peaks identified in the decay chain of ^{238}U will be shown in this section. In Figure 32, all identified peaks are shown, whereas in the subsequent spectra in this section, only the peaks tracked over time are marked.

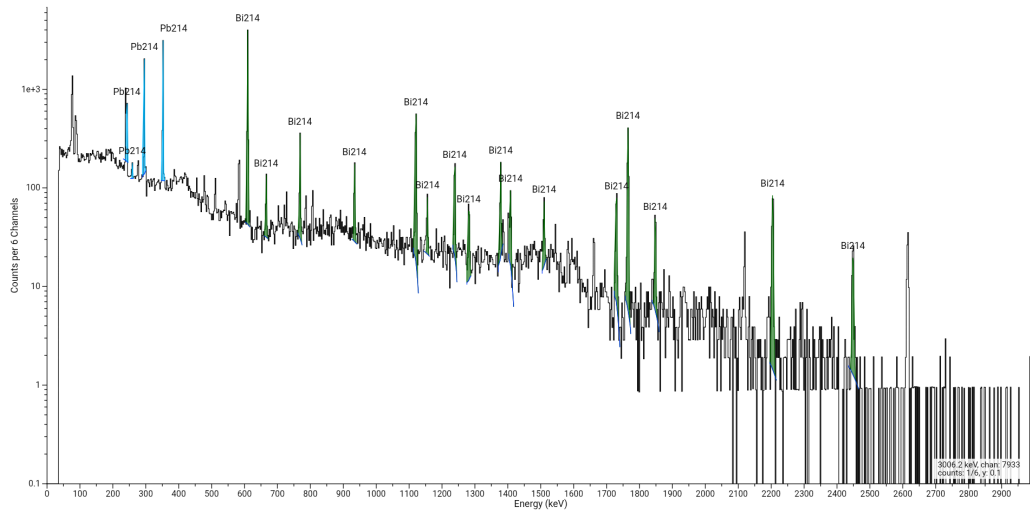


Figure 32: Spectrum 1b. 700 s measurement

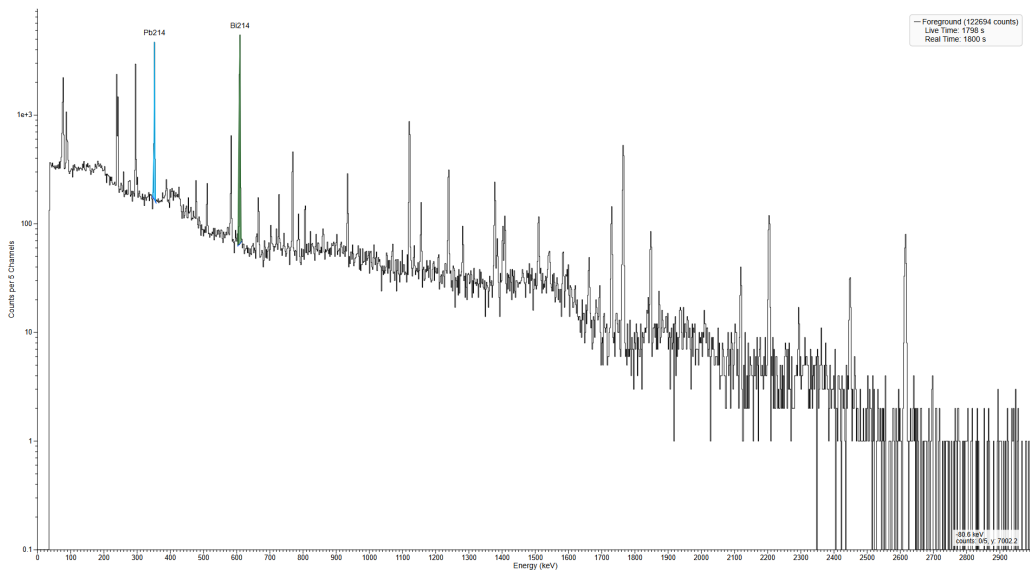


Figure 33: Spectrum 2b. 1800 s measurement

D Tables with all the data extracted from the .csv files

In this section, all the data from the peaks from the different measurements will be presented.

D.1 Background radiation

This is the data from the peaks in the background radiation spectrum. The live time for this measurement was 587800 s.

Centroid	Net_Area	Net_Area	Peak
keV	Counts	Uncertainty	CPS
63.24	955.7	110.2	1.6259e-003
92.67	4141.0	135.2	7.0448e-003
185.94	2546.3	166.2	4.3320e-003
238.82	1558.5	33.0	2.6514e-003
295.60	432.3	13.2	7.3548e-004
338.37	318.1	262.4	5.4110e-004
352.23	1000.6	188.1	1.7022e-003
511.18	8206.9	154.6	1.3962e-002
583.29	618.3	69.1	1.0519e-003
598.29	488.8	434.8	8.3154e-004
609.51	1221.2	96.3	2.0776e-003
661.88	1918.9	70.2	3.2645e-003
847.00	1075.3	53.2	1.8294e-003
911.60	567.4	87.3	9.6521e-004
969.53	281.7	219.2	4.7919e-004
1001.20	286.3	14.5	4.8713e-004
1120.73	210.2	62.8	3.5766e-004
1238.49	257.7	203.9	4.3837e-004
1461.21	2994.4	69.5	5.0942e-003
1591.40	182.2	48.0	3.1002e-004
1765.07	493.0	93.0	8.3872e-004
2103.31	190.3	146.3	3.2382e-004
2204.26	97.3	9.1	1.6560e-004
2614.88	1303.2	33.1	2.2170e-003

Table 4: Data from the peaks highlighted in the background spectrum

D.2 ^{232}Th foil

In this section, all the data from the measurement on the ^{232}Th foil will be presented. There are different tables for the different nuclides identified. The live time for this measurement was 102500 s.

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
99.83	36980.2	356.2
129.42	90234.4	401.3
154.36	33649.9	306.2
199.81	11528.0	273.6
209.67	182401.5	516.9
216.36	14863.4	276.0
270.67	122196.8	410.1
328.44	95756.0	382.7
332.80	11247.6	236.7
338.76	415267.4	691.6
341.43	10371.2	224.1
409.90	49374.5	300.8
463.46	102690.7	371.3
562.93	19542.5	228.0
755.81	15854.3	206.7
772.82	21048.7	219.7
782.66	8884.2	242.0
795.45	60640.4	288.9
831.04	7197.4	169.2
836.25	24580.1	231.0
840.91	12257.1	180.3
904.80	8630.0	164.3
911.76	407875.4	674.0
965.37	69230.1	82.9
969.53	231064.1	601.5
1247.22	7592.0	148.8
1459.97	8336.6	125.8
1496.74	9491.0	146.7
1502.37	4762.1	114.6
1581.37	6940.9	188.3
1589.15	36681.8	123.9
1631.44	15926.1	162.0
1639.16	4283.8	107.7

Table 5: Data from the ^{228}Ac peaks

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
241.38	201595.6	523.7

Table 6: Data from the ^{224}Ra peaks

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
115.52	23489.2	314.9
239.06	2245726.3	1545.7
300.56	124290.2	418.5

Table 7: Data from the ^{212}Pb peaks

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
727.76	133834.5	420.9

Table 8: Data from the ^{212}Bi peaks

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
277.82	71543.6	346.9
511.13	145591.4	435.0
583.62	563369.3	782.8
763.86	8931.1	185.5
861.08	71038.7	300.2
2615.52	163363.8	413.9

Table 9: Data from the ^{208}Tl peaks

D.3 Air filter

D.3.1 Data from peaks from the decay chain of ^{232}Th

The energy levels in the following tables correspond to different nuclides as: 238.93 keV is ^{212}Pb , 478.03 keV is ^7Be , 727.77 keV is ^{212}Bi and 2615.11 keV

is ^{208}Tl . This is also seen in Section 5.3.2 Table 2.

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.93	849.0	34.8
478.03	81.9	14.2
727.77	69.1	14.6
2615.11	61.2	8.6

Table 10: Data from Spectrum 1a. 700 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.96	2467.0	54.4
477.88	190.2	20.4
727.68	151.4	16.6
2615.16	208.7	15.9

Table 11: Data from Spectrum 2a. 1800 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.96	4541.0	73.0
478.06	347.3	22.7
727.52	325.5	23.0
2615.26	361.8	20.2

Table 12: Data from Spectrum 3a. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.95	4308.5	68.6
477.87	392.2	21.8
727.57	287.9	18.6
2615.07	364.0	19.9

Table 13: Data from Spectrum 4a. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.96	4053.2	65.2
477.93	353.7	20.0
727.49	293.4	17.7
2615.23	326.4	19.3

Table 14: Data from Spectrum 5a. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.94	3904.3	63.9
477.90	374.1	20.3
727.66	211.9	15.5
2614.86	352.1	19.5

Table 15: Data from Spectrum 6a. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.95	1762.8	42.7
477.91	181.6	14.3
727.67	103.3	10.8
2615.34	150.4	13.3

Table 16: Data from Spectrum 7a. 1800 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.95	26122.5	165.8
477.89	3572.9	63.3
727.57	1556.9	42.3
2615.10	2542.5	52.9

Table 17: Data from Spectrum 9a. 36000 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.92	954.7	31.6
477.92	204.7	14.8
727.53	62.0	8.5
2615.56	57.5	8.4

Table 18: Data from Spectrum 10a. 1800 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
238.88	21162.8	152.2
477.83	8758.5	98.0
727.54	1288.9	41.5
2614.89	2036.8	46.7

Table 19: Data from Spectrum 12a. 86400 s measurement

D.3.2 Data from peaks from the decay chain of ^{238}U

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.26	3263.5	59.2
609.56	4526.3	68.4

Table 20: Data from Spectrum 1b. 700 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.29	5093.5	74.2
609.62	7863.8	90.1

Table 21: Data from Spectrum 2b. 1800 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.24	3573.2	63.1
609.57	6124.2	79.8

Table 22: Data from Spectrum 3b. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.19	757.9	30.1
609.57	1572.6	40.9

Table 23: Data from Spectrum 4b. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.18	179.4	16.2
609.54	390.0	20.5

Table 24: Data from Spectrum 5b. 3600 s measurement

Centroid	Net_Area	Net_Area
keV	Counts	Uncertainty
352.35	77.8	14.0
609.57	91.7	10.7

Table 25: Data from Spectrum 6b. 3600 s measurement

E Python Code

E.1 ^{232}Th foil data analysis

```
1 # -*- coding: utf-8 -*-
2 """Th232_kandidatarbete.ipynb
3
4 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1hBOR1pczNZtYCn9iq4VmLvg-
8         mwG1SThg
9 """
10 from google.colab import files
11 import numpy as np
12 import pandas as pd
13 import matplotlib.pyplot as plt
14
15 ac = pd.read_csv('Ac228_peaks.CSV')
16 pb = pd.read_csv('Pb212_peaks.CSV')
17 tl = pd.read_csv('Tl208_peaks.CSV')
18 ra = pd.read_csv('Ra224_peaks.CSV')
19 bi = pd.read_csv('Bi212_peaks.CSV')
20 bg = pd.read_csv('Background_peaks.CSV')
21
22 from collections import Counter
23 bg.columns = bg.columns.str.replace(" ", "", regex=True)
24 bg.columns = bg.columns.dropna()
25
26 bg_new_columns = []
27 bg_counter = Counter()
28 for col in bg.columns:
29     bg_counter[col] += 1
30     bg_new_columns.append(col if bg_counter[col] == 1 else f"{col}_{
31         bg_counter[col] - 1}")
32 bg.columns = bg_new_columns
33
34 bg_E = [float(x) for x in list(bg['Centroid'][1:])]
35 bg_N = [float(x) for x in list(bg['Net_Area'][1:])]
36 bg_N_err = [float(x) for x in list(bg['Net_Area_1'][1:])]
37 bg_cps = [float(x) for x in list(bg['Peak'][1:])]
38 bg_cps_err = []
39
40 for i in range(len(bg_E)):
41     err = (bg_N_err[i]/bg_N[i])*bg_cps[i]
42     bg_cps_err.append(err)
43
44 """Efficiency Curve"""
45
46 ## Plot the Efficiency Curve
47 ef = pd.read_csv('Effektivitetskurva.csv')
48 ef = ef[ef['Peak efficiency'].notna()] ##Drops all NaN values
49 ef.plot(x='E (keV)', y='Peak efficiency', color = 'red', label = "Simulated
50     efficiency curve") ##Plots the efficiency vs energy
```

```

49 plt.xlabel('Energy [keV]')
50 plt.ylabel('Efficiency [%]')
51
52 """Ac228 - Peaks"""
53
54 ac.columns = ac.columns.str.replace(" ", "", regex=True)
55 ac.columns = ac.columns.dropna()
56
57 ## My columns have the same name, so I rename them
58 from collections import Counter
59
60 # Create a new list of column names
61 ac_new_columns = []
62 counter = Counter()
63
64 for col in ac.columns:
65     counter[col] += 1
66     ac_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[col] - 1}")
67
68 ac.columns = ac_new_columns
69
70 """Remove background from the Ac-228 peaks"""
71
72 tolerance = 1
73 t = 1.025*10**5
74 ac_E = [float(x) for x in list(ac['Centroid'][1:])]
75 ac_N = [float(x) for x in list(ac['Net_Area'][1:])]
76 ac_N_err = [float(x) for x in list(ac['Net_Area_1'][1:])]
77
78 del ac_E[5]
79 del ac_N[5]
80 del ac_N_err[5]
81
82
83 for i in ac_E:
84     for k in bg_E:
85         if 0 < i-k < tolerance:
86             ac_index = ac_E.index(i)
87             bg_index = bg_E.index(k)
88             ac_N[ac_index] = ac_N[ac_index] - bg_cps[bg_index]*t
89
90 ac_I = np.array([1.26, 2.42, 0.722, 0.315, 3.89, 3.46, 2.95, 0.40, 11.27,
91                 0.369, 1.92, 4.40, 0.87, 1, 1.49, 0.485, 4.25, 0.540, 1.61, 0.91, 0.77,
92                 25.8, 4.99, 15.8, 0.50, 0.83, 0.86, 0.46, 0.60, 3.22, 1.51, 0.47])/100
93 ac_I_err = np.array([0.07, 0.09, 0.021, 0.005, 0.07, 0.06, 0.12, 0.04, 0.19,
94                     0.21, 0.04, 0.07, 0.03, 0.03, 0.03, 0.019, 0.07, 0.021, 0.06, 0.04,
95                     0.03, 0.4, 0.09, 0.3, 0.03, 0.08, 0.04, 0.03, 0.04, 0.08, 0.04, 0.03])/100
96
97 """Calculate activity from every peak with error"""
98
99 Activity_ac_peaks = []
100 Activity_ac_peaks_err = []
101
102 t = 1.025*10**5

```

```

100 for i in range(len(ac_E)):
101     E = ac_E[i]
102     N = ac_N[i]
103     N_err = ac_N_err[i]
104     I = ac_I[i]
105     I_err = ac_I_err[i]
106     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
107     A = (N/(t*I*eff))
108     Activity_ac_peaks.append(A)
109
110     ## Error propagation
111     dAdN = 1/(t*I*eff)
112     dAdI = -N/(t*I**2*eff)
113     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
114     Activity_ac_peaks_err.append(delta_A)
115
116
117 Activity_ac_peaks = [float(x) for x in Activity_ac_peaks]
118 Activity_ac_peaks_err = [float(x) for x in Activity_ac_peaks_err]
119
120 ## Plot the errorbar
121 plt.errorbar(ac_E, Activity_ac_peaks, yerr = Activity_ac_peaks_err, fmt = '
122             yo', ecolor = 'purple', capsize = 5, ms = 3, label = 'Activity ac212')
123 plt.legend()
124 plt.xlabel('Energy [keV]')
125 plt.ylabel('Activity')
126 plt.ylim(0,800)
127 plt.xlim(0,3000)
128
129 """Check for, and delete, outliers with Grubb's Test. Plot without outliers.
130 """
131
132 ## Table on page 4 of Frank Grubbs article
133 Observations_n = [30, 35]
134 T_crit = [2.75, 2.82]
135 Activity_ac_peaks = np.array(Activity_ac_peaks)
136 Activity_ac_peaks_err = np.array(Activity_ac_peaks_err)
137 ac_E = np.array(ac_E)
138 ac_index_list = []
139
140 while True:
141     n = len(Activity_ac_peaks)
142
143     Activity_mean = np.mean(Activity_ac_peaks)
144     Activity_std = np.std(Activity_ac_peaks, ddof=1)
145
146     T = np.max(np.abs((Activity_ac_peaks - Activity_mean)/Activity_std))
147
148     if T < np.interp(n, Observations_n, T_crit):
149         break
150     else:
151         index = np.argmax(np.abs((Activity_ac_peaks - Activity_mean)/
152                                 Activity_std))
153         ## Print the outliers that were found
154         print("Outlier found, and deleted, at index: ", index)
155         ac_index_list.append(index)

```

```

154     print("Energy: ", ac_E[index])
155     print("Activity: ", Activity_ac_peaks[index])
156     print("_"*30)
157
158     Activity_ac_peaks = np.delete(Activity_ac_peaks, index)
159     Activity_ac_peaks_err = np.delete(Activity_ac_peaks_err, index)
160     ac_E = np.delete(ac_E, index)
161
162     plt.errorbar(ac_E, Activity_ac_peaks, yerr =Activity_ac_peaks_err, fmt = 'o',
163                 , ecolor = 'red', capsize = 5, ms = 3, label = 'Activity Ac228')
164     plt.legend()
165     plt.xlabel('Energy [keV]')
166     plt.ylabel('Activity')
167     plt.ylim(0,800)
168     plt.xlim(0,3000)
169     """Calculate the activity for Th232 and plotting it in the same plot as the
170         activity for Ac228 peaks"""
171
172     m_Th = 0.0932 ## From the lab
173     m_Th_err = 0.00005
174
175     N_a = 6.02214076E23 ## From Physics Handbook
176
177     M_Th = 232.04 ## From SI-data
178
179     lambda_Th = 1.569E-18 ## From IAEA
180     lambda_Th_err = 0.011E-18 ## From IAEA
181
182     N_Th = m_Th*N_a/M_Th
183     N_Th_err = np.sqrt(((N_a/M_Th)**2)*(m_Th_err**2))
184
185     Activity_Th = N_Th*lambda_Th
186     Activity_Th_err = np.sqrt((lambda_Th**2)*(N_Th_err**2)+(N_Th**2)*(
187         lambda_Th_err**2))
188
189     ## Plot
190     plt.errorbar(ac_E, Activity_ac_peaks, yerr =Activity_ac_peaks_err, fmt = 'o',
191                 , ecolor = 'red', capsize = 5, ms = 3, label = 'Activity $^{228}$Ac')
192     plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity $
193         ^{232}$Th')
194     plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
195                     Activity_Th + Activity_Th_err, color='black', alpha=0.2)
196     plt.legend()
197     plt.xlabel('Energy [keV]')
198     plt.ylabel('Activity [Bq]')
199     plt.ylim(0,800)
200     plt.xlim(0,3000)
201     """Assuming secular equilibrium and looking at the efficiency"""
202
203     t = 1.025*10**5
204     ac_eff = []
205     ac_eff_err = []
206     ac_index_list = [int(x) for x in ac_index_list]
207
208     for i in range(len(ac_N)):

```

```

205 I = ac_I[i]
206 eff = ac_N[i]/(t*I*Activity_Th)
207 ac_eff.append(eff)
208
209 ## Error propagation
210 dAdN = 1/(t*I*Activity_Th)
211 delta_N = ac_N_err[i]
212 dAdI = -ac_N[i]/(t*I**2*Activity_Th)
213 delta_I = ac_I_err[i]
214 dAdTh = -ac_N[i]/(t*I*Activity_Th**2)
215 delta_Th = Activity_Th_err
216 delta_eff = np.sqrt((dAdN**2)*(delta_N**2)+(dAdI**2)*(delta_I**2)+(dAdTh
**2)*(delta_Th**2))
217 ac_eff_err.append(delta_eff)
218
219 ac_eff = [float(x) for x in ac_eff]
220 ac_eff_err = [float(x) for x in ac_eff_err]
221 MAE_scale_error = sum(ac_eff_err)/len(ac_eff_err)
222
223 ## Delete the outliers we know about
224
225 for i in ac_index_list:
226     ac_eff.pop(i)
227     ac_eff_err.pop(i)
228
229 ac_eff = np.array(ac_eff)
230 ac_eff_err = np.array(ac_eff_err)
231
232 print(len(ac_E))
233 print(len(ac_eff))
234
235 ef.plot(x='E (keV)', y='Peak efficiency', color = 'red', label = 'Simulated
efficiency curve') ##Plots the efficiency vs energy
236 plt.errorbar(ac_E, ac_eff, yerr = ac_eff_err, fmt = 'o', ecolor = 'red',
capsize = 5, ms = 3, label = 'Efficiency  $^{228}\text{Ac}$ ')
237 plt.legend()
238 plt.xlabel('Energy [keV]')
239 plt.ylabel('Efficiency [%]')
240
241 """Scaling the efficiency with mean of the ratio"""
242
243 ## Mean of the ratio
244
245 Ratio_between_aceff_eff = []
246 for i in range(len(ac_eff)):
247     eff_curve_value = np.interp(ac_E[i], list(ef['E (keV)']), list(ef['Peak
efficiency']))
248     Ratio_between_aceff_eff.append(ac_eff[i]/eff_curve_value)
249
250 Ratio_between_aceff_eff = np.array(Ratio_between_aceff_eff)
251 ac_mean_ratio = np.mean(Ratio_between_aceff_eff)
252
253 ## Scaling
254 ef_peak_efficiency = [float(x) for x in list(ef['Peak efficiency'])]
255 ef_E = [float(x) for x in list(ef['E (keV)'])]
256 ef_peak_efficiency_scaled_ac = [float(x)*ac_mean_ratio for x in
ef_peak_efficiency]

```

```

257 ef_peak_efficiency_scaled_ac = [float(x) for x in
    ef_peak_efficiency_scaled_ac]
258
259 ## Plot
260
261 plt.plot(ef_E, ef_peak_efficiency_scaled_ac, color='red', label = '
    Simulated efficiency curve scaled') ##Plots the efficiency vs energy
262 plt.errorbar(ac_E, ac_eff, yerr = ac_eff_err, fmt = 'o', ecolor = 'red',
    capsize = 5, ms = 3, label = 'Efficiency  $^{228}\text{Ac}$ ')
263 plt.legend()
264 plt.xlabel('Energy [keV]')
265 plt.ylabel('Efficiency [%]')
266
267 """Scaling the activity"""
268
269 scaled_Activity_ac_peaks = Activity_ac_peaks/ac_mean_ratio
270 scaled_Activity_ac_peaks_err = Activity_ac_peaks_err/ac_mean_ratio
271
272 ## Plot
273 plt.errorbar(ac_E, scaled_Activity_ac_peaks, yerr =
    scaled_Activity_ac_peaks_err, fmt = 'o', ecolor = 'red', capsize = 5, ms
    = 3, label = 'Activity  $^{228}\text{Ac}$ ')
274 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
275 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
    Activity_Th + Activity_Th_err, color='black', alpha=0.2)
276 plt.legend()
277 plt.xlabel('Energy [keV]')
278 plt.ylabel('Activity [Bq]')
279 plt.ylim(0,800)
280 plt.xlim(0,3000)
281
282 """Pb212 Peaks"""
283
284 pb.columns = pb.columns.str.replace(" ", "", regex=True)
285 pb.columns = pb.columns.dropna()
286
287 ## My columns have the same name, so I rename them
288
289 # Create a new list of column names
290 pb_new_columns = []
291 counter = Counter()
292
293 for col in pb.columns:
294     counter[col] += 1
295     pb_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[col]
    } - 1}")
296
297 pb.columns = pb_new_columns
298
299 ## Change the values in the relevant columns to floats
300 pb_E = [float(x) for x in list(pb['Centroid'][1:])]
301 pb_N = [float(x) for x in list(pb['Net_Area'][1:])]
302 pb_N_err = [float(x) for x in list(pb['Net_Area_1'][1:])]
303
304 tolerance = 1
305

```

```

306 for i in pb_E:
307     for k in bg_E:
308         if 0 < i-k < tolerance:
309             pb_index = pb_E.index(i)
310             bg_index = bg_E.index(k)
311             pb_N[pb_index] = pb_N[pb_index] - bg_cps[bg_index]*t
312
313 ## All the intensities and errors
314 pb_I = np.array([0.596, 43.6, 3.30])/100
315 pb_I_err = np.array([0.009, 0.5, 0.04])/100
316
317 Activity_pb_peaks = []
318 Activity_pb_peaks_err = []
319 t = 1.025*10**5
320
321 for i in range(len(pb_E)):
322     E = pb_E[i]
323     N = pb_N[i]
324     N_err = pb_N_err[i]
325     I = pb_I[i]
326     I_err = pb_I_err[i]
327     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
328     A = (N/(t*I*eff))
329     Activity_pb_peaks.append(A)
330
331     ## Error propagation
332     dAdN = 1/(t*I*eff)
333     dAdI = -N/(t*I**2*eff)
334     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
335     Activity_pb_peaks_err.append(delta_A)
336
337
338 Activity_pb_peaks = [float(x) for x in Activity_pb_peaks]
339 Activity_pb_peaks_err = [float(x) for x in Activity_pb_peaks_err]
340
341 ## Plot the errorbar
342 plt.errorbar(pb_E, Activity_pb_peaks, yerr = Activity_pb_peaks_err, fmt = '
yo', ecolor = 'purple', capsize = 5, ms = 3, label = 'Activity Pb212')
343 plt.legend()
344 plt.xlabel('Energy [keV]')
345 plt.ylabel('Activity')
346 plt.ylim(0,800)
347 plt.xlim(0,3000)
348
349 plt.errorbar(pb_E, Activity_pb_peaks, yerr =Activity_pb_peaks_err, fmt = 'ro
', ecolor = 'red', capsize = 5, ms = 3, label = 'Activity 212Pb')
350 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity $
232Th')
351 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
Activity_Th + Activity_Th_err, color='black', alpha=0.2)
352 plt.legend()
353 plt.xlabel('Energy [keV]')
354 plt.ylabel('Activity [Bq]')
355 plt.ylim(0,800)
356 plt.xlim(0,3000)
357
358 """Scaled with Ac228 scaling"""

```

```

359 scaled_Activity_pb_peaks = Activity_pb_peaks/ac_mean_ratio
360 scaled_Activity_pb_peaks_err = Activity_pb_peaks_err/ac_mean_ratio
361
362
363 ## Plot
364 plt.errorbar(pb_E, scaled_Activity_pb_peaks, yerr =
    scaled_Activity_pb_peaks_err, fmt = 'ro', ecolor = 'red', capsize = 5, ms
    = 3, label = 'Activity  $^{212}\text{Pb}$ ')
365 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
366 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
    Activity_Th + Activity_Th_err, color='black', alpha=0.2)
367 plt.legend()
368 plt.xlabel('Energy [keV]')
369 plt.ylabel('Activity [Bq]')
370 plt.ylim(0,800)
371 plt.xlim(0,3000)
372
373 t = 1.025*10**5
374 pb_eff = []
375 pb_eff_err = []
376
377 for i in range(len(pb_N)):
378     N = pb_N[i]
379     I = pb_I[i]
380     N_err = pb_N_err[i]
381     I_err = pb_I_err[i]
382     eff = N/(t*I*Activity_Th)
383     pb_eff.append(eff)
384
385     ## Error propagation
386     dAdN = 1/(t*I*Activity_Th)
387     dAdI = -N/(t*I**2*Activity_Th)
388     dAdTh = -N/(t*I*Activity_Th**2)
389     delta_eff = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2)+(dAdTh**2)*
    Activity_Th_err**2)
390     pb_eff_err.append(delta_eff)
391
392 pb_eff = [float(x) for x in pb_eff]
393 pb_eff_err = [float(x) for x in pb_eff_err]
394
395 pb_eff = np.array(pb_eff)
396 pb_eff_err = np.array(pb_eff_err)
397
398 ef.plot(x='E (keV)', y='Peak efficiency', color = 'purple') ##Plots the
    efficiency vs energy
399 plt.errorbar(pb_E, pb_eff, yerr = pb_eff_err, fmt = 'yo', ecolor = 'purple',
    capsize = 5, ms = 3, label = 'Pb212 efficiency')
400 plt.legend()
401 plt.xlabel('Energy [keV]')
402 plt.ylabel('Efficiency')
403
404 """Tl208 Peaks"""
405
406 t1.columns = t1.columns.str.replace(" ", "", regex=True)
407 t1.columns = t1.columns.dropna()
408

```

```

409 ## My columns have the same name, so I rename them
410
411 # Create a new list of column names
412 tl_new_columns = []
413 counter = Counter()
414
415 for col in tl.columns:
416     counter[col] += 1
417     tl_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[col]
418                                     } - 1}")
419
420 tl.columns = tl_new_columns
421
422 ## Change the values in the relevant columns to floats
423 tl_E = [float(x) for x in list(tl['Centroid'][1:])]
424 tl_N = [float(x) for x in list(tl['Net_Area'][1:])]
425 tl_N_err = [float(x) for x in list(tl['Net_Area_1'][1:])]
426
427 tolerance = 1
428
429 for i in tl_E:
430     for k in bg_E:
431         if 0 < i-k < tolerance:
432             tl_index = tl_E.index(i)
433             bg_index = bg_E.index(k)
434             tl_N[tl_index] = tl_N[pb_index] - bg_cps[bg_index]*t
435
436 ## All the intensities and errors
437 tl_I = np.array([6.6, 22.6, 85.0, 1.79, 12.5, 99.754])/100
438 tl_I_err = np.array([0.3, 0.2, 0.3, 0.03, 0.1, 0.004])/100
439 decay_to_tl = 0.3594
440
441 Activity_tl_peaks = []
442 Activity_tl_peaks_err = []
443 t = 1.025*10**5
444
445 for i in range(len(tl_E)):
446     E = tl_E[i]
447     N = tl_N[i]
448     N_err = tl_N_err[i]
449     I = tl_I[i]
450     I_err = tl_I_err[i]
451     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
452     A = (N/(t*I*eff))/decay_to_tl
453     Activity_tl_peaks.append(A)
454
455     ## Error propagation
456     dAdN = 1/(t*I*eff)
457     dAdI = -N/(t*I**2*eff)
458     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
459     Activity_tl_peaks_err.append(delta_A)
460
461 Activity_tl_peaks = [float(x) for x in Activity_tl_peaks]
462 Activity_tl_peaks_err = [float(x) for x in Activity_tl_peaks_err]
463
464 ## Plot the errorbar

```

```

465 plt.errorbar(tl_E, Activity_tl_peaks, yerr =Activity_tl_peaks_err, fmt = 'ro
      ', ecolord = 'green', capsize = 5, ms = 3, label = 'Activity Tl208')
466 plt.legend()
467 plt.xlabel('Energy [keV]')
468 plt.ylabel('Activity')
469 plt.ylim(0,800)
470 plt.xlim(0,3000)
471
472 """Check for outliers with Grubb's"""
473
474 plt.errorbar(tl_E, Activity_tl_peaks, yerr =Activity_tl_peaks_err, fmt = 'o'
      , color = 'purple', ecolord = 'red', capsize = 5, ms = 3, label = '
      Activity  $\text{}^{208}\text{Tl}$ ')
475 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $\text{}^{232}\text{Th}$ ')
476 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
477 plt.legend()
478 plt.xlabel('Energy [keV]')
479 plt.ylabel('Activity [Bq]')
480 plt.ylim(0,800)
481 plt.xlim(0,3000)
482
483 """Scale the peaks with the scale factor and compare with Th232"""
484
485 scaled_Activity_tl_peaks = Activity_tl_peaks/ac_mean_ratio
486 scaled_Activity_tl_peaks_err = Activity_tl_peaks_err/ac_mean_ratio
487
488 ## Plot
489 plt.errorbar(tl_E, scaled_Activity_tl_peaks, yerr =
      scaled_Activity_tl_peaks_err, fmt = 'o', color = 'purple', ecolord = 'red'
      , capsize = 5, ms = 3, label = 'Activity  $\text{}^{208}\text{Tl}$ ')
490 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $\text{}^{232}\text{Th}$ ')
491 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
492 plt.legend()
493 plt.xlabel('Energy [keV]')
494 plt.ylabel('Activity [Bq]')
495 plt.ylim(0,800)
496 plt.xlim(0,3000)
497
498 t = 1.025*10**5
499 tl_eff = []
500 tl_eff_err = []
501
502 for i in range(len(tl_N)):
503     N = tl_N[i]
504     I = tl_I[i]
505     N_err = tl_N_err[i]
506     I_err = tl_I_err[i]
507     eff = (N/(t*I*Activity_Th))/decay_to_tl
508     tl_eff.append(eff)
509
510     ## Error propagation
511     dAdN = 1/(t*I*Activity_Th)
512     dAdI = -N/(t*I**2*Activity_Th)

```

```

513     dAdTh = -N/(t*I*Activity_Th**2)
514     delta_eff = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2)+(dAdTh**2)*(
515         Activity_Th_err**2))
516     tl_eff_err.append(delta_eff)
517
518     tl_eff = [float(x) for x in tl_eff]
519     tl_eff_err = [float(x) for x in tl_eff_err]
520
521     tl_eff = np.array(tl_eff)
522     tl_eff_err = np.array(tl_eff_err)
523
524     ef.plot(x='E (keV)', y='Peak efficiency', color='purple') ##Plots the
525         efficiency vs energy
526     plt.errorbar(tl_E, tl_eff, yerr = tl_eff_err, fmt = 'ro', ecolor = 'green',
527         capsize = 5, ms = 3, label = 'Tl208 efficiency')
528     plt.legend()
529     plt.xlabel('Energy [keV]')
530     plt.ylabel('Efficiency')
531
532     """Ra224"""
533
534     ra.columns = ra.columns.str.replace(" ", "", regex=True)
535     ra.columns = ra.columns.dropna()
536
537     ## My columns have the same name, so I rename them
538
539     # Create a new list of column names
540     ra_new_columns = []
541     counter = Counter()
542
543     for col in ra.columns:
544         counter[col] += 1
545         ra_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[col]
546             } - 1}")
547
548     ra.columns = ra_new_columns
549
550     ## Change the values in the relevant columns to floats
551     ra_E = [float(x) for x in list(ra['Centroid'])[1:]]
552     ra_N = [float(x) for x in list(ra['Net_Area'])[1:]]
553     ra_N_err = [float(x) for x in list(ra['Net_Area_1'])[1:]]
554
555     tolerance = 1
556
557     for i in ra_E:
558         for k in bg_E:
559             if 0 < i-k < tolerance:
560                 ra_index = ra_E.index(i)
561                 bg_index = bg_E.index(k)
562                 ra_N[ra_index] = ra_N[ra_index] - bg_cps[bg_index]*t
563
564     ## All the intensities and errors
565     ra_I = np.array([4.10])/100
566     ra_I_err = np.array([0.05])/100
567
568     Activity_ra_peaks = []
569     Activity_ra_peaks_err = []

```

```

566 t = 1.025*10**5
567
568 for i in range(len(ra_E)):
569     E = ra_E[i]
570     N = ra_N[i]
571     N_err = ra_N_err[i]
572     I = ra_I[i]
573     I_err = ra_I_err[i]
574     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
575     A = (N/(t*I*eff))
576     Activity_ra_peaks.append(A)
577
578     ## Error propagation
579     dAdN = 1/(t*I*eff)
580     dAdI = -N/(t*I**2*eff)
581     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
582     Activity_ra_peaks_err.append(delta_A)
583
584
585 Activity_ra_peaks = [float(x) for x in Activity_ra_peaks]
586 Activity_ra_peaks_err = [float(x) for x in Activity_ra_peaks_err]
587
588 ## Plot the errorbar
589 plt.errorbar(ra_E, Activity_ra_peaks, yerr = Activity_ra_peaks_err, fmt = '
go', ecolor = 'red', capsize = 5, ms = 3, label = 'Activity $^{224}$Ra')
590 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity $
^{232}$Th')
591 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
Activity_Th + Activity_Th_err, color='black', alpha=0.2)
592 plt.legend()
593 plt.xlabel('Energy [keV]')
594 plt.ylabel('Activity')
595 plt.ylim(0,800)
596 plt.xlim(0,3000)
597
598 """Scale and compare with Th232"""
599
600 scaled_Activity_ra_peaks = Activity_ra_peaks/ac_mean_ratio
601 scaled_Activity_ra_peaks_err = Activity_ra_peaks_err/ac_mean_ratio
602
603 ## Plot
604 plt.errorbar(ra_E, scaled_Activity_ra_peaks, yerr =
scaled_Activity_ra_peaks_err, fmt = 'go', ecolor = 'red', capsize = 5, ms
= 3, label = 'Activity $^{224}$Ra')
605 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity $
^{232}$Th')
606 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
Activity_Th + Activity_Th_err, color='black', alpha=0.2)
607 plt.legend()
608 plt.xlabel('Energy [keV]')
609 plt.ylabel('Activity')
610 plt.ylim(0,800)
611 plt.xlim(0,3000)
612
613 """Bi212"""
614
615 bi.columns = bi.columns.str.replace(" ", "", regex=True)

```

```

616 bi.columns = bi.columns.dropna()
617
618 ## My columns have the same name, so I rename them
619
620 # Create a new list of column names
621 bi_new_columns = []
622 counter = Counter()
623
624 for col in bi.columns:
625     counter[col] += 1
626     bi_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[col] - 1}")
627
628 bi.columns = bi_new_columns
629
630 ## Change the values in the relevant columns to floats
631 bi_E = [float(x) for x in list(bi['Centroid'][1:])]
632 bi_N = [float(x) for x in list(bi['Net_Area'][1:])]
633 bi_N_err = [float(x) for x in list(bi['Net_Area_1'][1:])]
634
635 tolerance = 1
636
637 for i in bi_E:
638     for k in bg_E:
639         if 0 < i-k < tolerance:
640             bi_index = bi_E.index(i)
641             bg_index = bg_E.index(k)
642             bi_N[ra_index] = bi_N[ra_index] - bg_cps[bg_index]*t
643
644 ## All the intensities and errors
645 bi_I = np.array([6.67])/100
646 bi_I_err = np.array([0.09])/100
647
648 Activity_bi_peaks = []
649 Activity_bi_peaks_err = []
650 t = 1.025*10**5
651
652 for i in range(len(bi_E)):
653     E = bi_E[i]
654     N = bi_N[i]
655     N_err = bi_N_err[i]
656     I = bi_I[i]
657     I_err = bi_I_err[i]
658     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
659     A = (N/(t*I*eff))
660     Activity_bi_peaks.append(A)
661
662     ## Error propagation
663     dAdN = 1/(t*I*eff)
664     dAdI = -N/(t*I**2*eff)
665     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
666     Activity_bi_peaks_err.append(delta_A)
667
668 Activity_bi_peaks = [float(x) for x in Activity_bi_peaks]
669 Activity_bi_peaks_err = [float(x) for x in Activity_bi_peaks_err]
670
671 ## Plot the errorbar

```

```

672 plt.errorbar(bi_E, Activity_bi_peaks, yerr = Activity_bi_peaks_err, fmt = 'o
      ', color = 'orange', ecolor = 'red', capsize = 5, ms = 3, label = '
      Activity  $^{212}\text{Bi}$ ')
673 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
674 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
675 plt.legend()
676 plt.xlabel('Energy [keV]')
677 plt.ylabel('Activity')
678 plt.ylim(0,800)
679 plt.xlim(0,3000)
680
681 scaled_Activity_bi_peaks = Activity_bi_peaks / ac_mean_ratio
682 scaled_Activity_bi_peaks_err = Activity_bi_peaks_err / ac_mean_ratio
683
684 ## Plot
685 plt.errorbar(bi_E, scaled_Activity_bi_peaks, yerr =
      scaled_Activity_bi_peaks_err, fmt = 'o', color = 'orange', ecolor = 'red
      ', capsize = 5, ms = 3, label = 'Activity  $^{212}\text{Bi}$ ')
686 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
687 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
688 plt.legend()
689 plt.xlabel('Energy [keV]')
690 plt.ylabel('Activity')
691 plt.ylim(0,800)
692 plt.xlim(0,3000)
693 print(scaled_Activity_bi_peaks)
694
695 plt.errorbar(ac_E, Activity_ac_peaks, yerr = Activity_ac_peaks_err, fmt = 'o
      ', ecolor = 'red', capsize = 5, ms = 3, label = 'Activity  $^{228}\text{Ac}$ ')
696 plt.errorbar(ra_E, Activity_ra_peaks, yerr = Activity_ra_peaks_err, fmt = '
      go', ecolor = 'red', capsize = 5, ms = 3, label = 'Activity  $^{224}\text{Ra}$ ')
697 plt.errorbar(tl_E, Activity_tl_peaks, yerr = Activity_tl_peaks_err, fmt = 'o
      ', color = 'purple', ecolor = 'red', capsize = 5, ms = 3, label = '
      Activity  $^{208}\text{Tl}$ ')
698 plt.errorbar(pb_E, Activity_pb_peaks, yerr = Activity_pb_peaks_err, fmt = '
      ro', ecolor = 'red', capsize = 5, ms = 3, label = 'Activity  $^{212}\text{Pb}$ ')
699 plt.errorbar(bi_E, Activity_bi_peaks, yerr = Activity_bi_peaks_err, fmt = 'o
      ', color = 'orange', ecolor = 'red', capsize = 5, ms = 3, label = '
      Activity  $^{212}\text{Bi}$ ')
700 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
701 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
702 plt.legend(fontsize='small')
703 plt.xlabel('Energy [keV]')
704 plt.ylabel('Activity [Bq]')
705 plt.ylim(0,800)
706 plt.xlim(0,3000)
707
708 plt.errorbar(ac_E, scaled_Activity_ac_peaks, yerr =
      scaled_Activity_ac_peaks_err, fmt = 'o', ecolor = 'red', capsize = 5, ms
      = 3, label = 'Scaled Activity  $^{228}\text{Ac}$ ')

```

```

709 plt.errorbar(ra_E, scaled_Activity_ra_peaks, yerr =
      scaled_Activity_ra_peaks_err, fmt = 'go', ecol = 'red', capsize = 5, ms
      = 3, label = 'Activity  $^{224}\text{Ra}$ ')
710 plt.errorbar(tl_E, scaled_Activity_tl_peaks, yerr =
      scaled_Activity_tl_peaks_err, fmt = 'o', color = 'purple', ecol = 'red',
      , capsize = 5, ms = 3, label = 'Activity  $^{208}\text{Tl}$ ')
711 plt.errorbar(pb_E, scaled_Activity_pb_peaks, yerr =
      scaled_Activity_pb_peaks_err, fmt = 'ro', ecol = 'red', capsize = 5, ms
      = 3, label = 'Activity  $^{212}\text{Pb}$ ')
712 plt.errorbar(bi_E, scaled_Activity_bi_peaks, yerr =
      scaled_Activity_bi_peaks_err, fmt = 'o', color = 'orange', ecol = 'red',
      , capsize = 5, ms = 3, label = 'Activity  $^{212}\text{Bi}$ ')
713 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
714 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
715 plt.legend(fontsize='small')
716 plt.xlabel('Energy [keV]')
717 plt.ylabel('Activity [Bq]')
718 plt.ylim(0,800)
719 plt.xlim(0,3000)
720
721 mean_activity_ac = np.mean(scaled_Activity_ac_peaks)
722 mean_activity_pb = np.mean(scaled_Activity_pb_peaks)
723 mean_activity_tl = np.mean(scaled_Activity_tl_peaks)
724
725 mean_err_activity_ac = np.mean(scaled_Activity_ac_peaks_err)
726 mean_err_activity_pb = np.mean(scaled_Activity_pb_peaks_err)
727 mean_err_activity_tl = np.mean(scaled_Activity_tl_peaks_err)
728
729
730 plt.axhline(Activity_Th, color='black', linestyle='--', label='Activity  $^{232}\text{Th}$ ')
731 plt.fill_between(np.linspace(0,3000,10), Activity_Th - Activity_Th_err,
      Activity_Th + Activity_Th_err, color='black', alpha=0.2)
732
733 plt.axhline(mean_activity_ac, color='blue', linestyle='--', label='Mean
      Activity  $^{228}\text{Ac}$ ')
734 plt.fill_between(np.linspace(0,3000,10), mean_activity_ac -
      mean_err_activity_ac, mean_activity_ac + mean_err_activity_ac, color='
      blue', alpha=0.2)
735
736 plt.axhline(mean_activity_pb, color='red', linestyle='--', label='Mean
      Activity  $^{212}\text{Pb}$ ')
737 plt.fill_between(np.linspace(0,3000,10), mean_activity_pb -
      mean_err_activity_pb, mean_activity_pb + mean_err_activity_pb, color='
      red', alpha=0.2)
738
739 plt.axhline(mean_activity_tl, color='purple', linestyle='--', label='Mean
      Activity  $^{208}\text{Tl}$ ')
740 plt.fill_between(np.linspace(0,3000,10), mean_activity_tl -
      mean_err_activity_tl, mean_activity_tl + mean_err_activity_tl, color='
      purple', alpha=0.2)
741
742 plt.legend(fontsize='small')
743 plt.xlabel('Energy [keV]')
744 plt.ylabel('Activity [Bq]')

```

```

745 plt.ylim(0,800)
746 plt.xlim(0,3000)
747
748 print(mean_activity_ac)
749 print(mean_activity_pb)
750 print(mean_activity_t1)
751 print(Activity_Th)
752
753 print(mean_err_activity_ac)
754 print(mean_err_activity_pb)
755 print(mean_err_activity_t1)

```

Listing 1: Code used for data analysis of the ^{232}Th foil data

E.2 Air filter data analysis

```

1 # -*- coding: utf-8 -*-
2 """Luftfilter_kandidatarbete.ipynb
3
4 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1
8     q_HEjpP15kmK0s15wmS0u0TT8F0dMNc0
9
10 from google.colab import files
11 import numpy as np
12 import pandas as pd
13 import matplotlib.pyplot as plt
14 from scipy.optimize import curve_fit
15
16 lf_700s = pd.read_csv('Luftfilter_700s.CSV')
17 lf_1800s_1 = pd.read_csv('Luftfilter_1800s_1.CSV')
18 lf_1800s_2 = pd.read_csv('Luftfilter_1800s_2.CSV')
19 lf_1800s_3 = pd.read_csv('Luftfilter_1800s_3.CSV')
20 lf_3600s = pd.read_csv('Luftfilter_3600s_0.CSV')
21 lf_3600s_1 = pd.read_csv('Luftfilter_3600s_1.CSV')
22 lf_3600s_2 = pd.read_csv('Luftfilter_3600s_2.CSV')
23 lf_3600s_3 = pd.read_csv('Luftfilter_3600s_3.CSV')
24 lf_36000s = pd.read_csv('Luftfilter_36000s.CSV')
25 lf_86400s = pd.read_csv('Luftfilter_86400s_1.CSV')
26 ef = pd.read_csv('Effektivitetskurva.csv')
27 bg =pd.read_csv('Background_peaks.CSV')
28
29 lf_700s_U238 = pd.read_csv('Luftfilter_700s_U238.CSV')
30 lf_1800s_1_U238 = pd.read_csv('Luftfilter_1800s_1_U238.CSV')
31 lf_3600s_0_U238 = pd.read_csv('Luftfilter_3600s_0_U238.CSV')
32 lf_3600s_1_U238 = pd.read_csv('Luftfilter_3600s_1_U238.CSV')
33 lf_3600s_2_U238 = pd.read_csv('Luftfilter_3600s_2_U238.CSV')
34 lf_3600s_3_U238 = pd.read_csv('Luftfilter_3600s_3_U238.CSV')
35
36 from collections import Counter
37 bg.columns = bg.columns.str.replace(" ", "", regex=True)

```

```

38 bg.columns = bg.columns.dropna()
39
40 bg_new_columns = []
41 bg_counter = Counter()
42 for col in bg.columns:
43     bg_counter[col] += 1
44     bg_new_columns.append(col if bg_counter[col] == 1 else f"{col}_{
45         bg_counter[col] - 1}")
46
47 bg.columns = bg_new_columns
48
49 bg_E = [float(x) for x in list(bg['Centroid'][1:])]
50 bg_N = [float(x) for x in list(bg['Net_Area'][1:])]
51 bg_N_err = [float(x) for x in list(bg['Net_Area_1'][1:])]
52 bg_cps = [float(x) for x in list(bg['Peak'][1:])]
53 bg_cps_err = []
54
55 for i in range(len(bg_E)):
56     err = (bg_N_err[i]/bg_N[i])*bg_cps[i]
57     bg_cps_err.append(err)
58
59 """Organisera mina filer"""
60
61 def parent_decay(NP0, lamP, t):
62     NP = NP0 * np.exp(-lamP * t)
63     return NP
64
65 def daughter_decay(NP0, NDO, lamP, lamD, t):
66     ND = (NP0 * (lamP / (lamD - lamP)) * (np.exp(-lamP * t) - np.exp(-lamD * t
67         )) + NDO * np.exp(-lamD * t))
68     return ND
69
70 def granddaughter_decay(NP0, NDO, NGO, lamP, lamD, lamG, t):
71     NG = (NP0 * lamP * lamD * ((np.exp(-lamP * t) / ((lamD - lamP) * (lamG -
72         lamP))) + (np.exp(-lamD * t) / ((lamP - lamD) * (lamG - lamD)))) + (np.
73         exp(-lamG * t) / ((lamP - lamG) * (lamD - lamG)))) + NDO * (lamD / (
74         lamG - lamD)) * (np.exp(-lamD * t) - np.exp(-lamG * t)) + NGO * np.exp
75         (-lamG * t))
76     return NG
77
78 def granddaughter_decay_branch(NP0, NDO, NGO, lamP, lamD, lamG, t, branch):
79     term1 = NP0 * lamP * lamD * (
80         (np.exp(-lamP * t) / ((lamD - lamP) * (lamG - lamP))) +
81         (np.exp(-lamD * t) / ((lamP - lamD) * (lamG - lamD))) +
82         (np.exp(-lamG * t) / ((lamP - lamG) * (lamD - lamG)))
83     )
84     term2 = NDO * (lamD / (lamG - lamD)) * (np.exp(-lamD * t) - np.exp(-lamG
85         * t))
86     term3 = NGO * np.exp(-lamG * t)
87     NG = branch * (term1 + term2) + term3
88     return NG
89
90 list_of_data = [lf_700s, lf_1800s_1, lf_3600s, lf_3600s_1, lf_3600s_2,
91     lf_3600s_3, lf_1800s_2, lf_3600s, lf_1800s_3, lf_86400s]
92 list_of_data_U238 = [lf_700s_U238, lf_1800s_1_U238, lf_3600s_0_U238,
93     lf_3600s_1_U238, lf_3600s_2_U238, lf_3600s_3_U238]

```

```

186 list_of_times = ['700s', '1800s_1', '3600s', '3600s_1', '3600s_2', '3600s_3',
187                 '1800s_2', '36000s', '1800s_3', '86400s']
188 list_of_times_U238 = ['700s_U238', '1800s_1_U238', '3600s_0_U238', '3600
189                        s_1_U238', '3600s_2_U238', '3600s_3_U238']
190 list_of_values = ['_E', '_N', '_N_err', '_Nuclide']
191
192 for lf in list_of_data:
193     lf.columns = lf.columns.str.replace(" ", "", regex=True)
194     lf.columns = lf.columns.dropna()
195
196     ## My columns have the same name, so I rename them
197     from collections import Counter
198
199     # Create a new list of column names
200     lf_new_columns = []
201     counter = Counter()
202
203     for col in lf.columns:
204         counter[col] += 1
205         lf_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[
206             col] - 1}")
207
208     lf.columns = lf_new_columns
209
210 ## Change the values in the relevant columns to floats
211 lf_700s_E = [float(x) for x in list(lf_700s['Centroid'][1:])]
212 lf_700s_N = [float(x) for x in list(lf_700s['Net_Area'][1:])]
213 lf_700s_N_err = [float(x) for x in list(lf_700s['Net_Area_1'][1:])]
214 lf_700s_Nuclide = list(lf_700s['Nuclide'][1:])
215
216 lf_1800s_1_E = [float(x) for x in list(lf_1800s_1['Centroid'][1:])]
217 lf_1800s_1_N = [float(x) for x in list(lf_1800s_1['Net_Area'][1:])]
218 lf_1800s_1_N_err = [float(x) for x in list(lf_1800s_1['Net_Area_1'][1:])]
219 lf_1800s_1_Nuclide = list(lf_1800s_1['Nuclide'][1:])
220
221 lf_1800s_2_E = [float(x) for x in list(lf_1800s_2['Centroid'][1:])]
222 lf_1800s_2_N = [float(x) for x in list(lf_1800s_2['Net_Area'][1:])]
223 lf_1800s_2_N_err = [float(x) for x in list(lf_1800s_2['Net_Area_1'][1:])]
224 lf_1800s_2_Nuclide = list(lf_1800s_2['Nuclide'][1:])
225
226 lf_1800s_3_E = [float(x) for x in list(lf_1800s_3['Centroid'][1:])]
227 lf_1800s_3_N = [float(x) for x in list(lf_1800s_3['Net_Area'][1:])]
228 lf_1800s_3_N_err = [float(x) for x in list(lf_1800s_3['Net_Area_1'][1:])]
229 lf_1800s_3_Nuclide = list(lf_1800s_3['Nuclide'][1:])
230
231 lf_3600s_E = [float(x) for x in list(lf_3600s['Centroid'][1:])]
232 lf_3600s_N = [float(x) for x in list(lf_3600s['Net_Area'][1:])]
233 lf_3600s_N_err = [float(x) for x in list(lf_3600s['Net_Area_1'][1:])]
234 lf_3600s_Nuclide = list(lf_3600s['Nuclide'][1:])
235
236 lf_3600s_1_E = [float(x) for x in list(lf_3600s_1['Centroid'][1:])]
237 lf_3600s_1_N = [float(x) for x in list(lf_3600s_1['Net_Area'][1:])]
238 lf_3600s_1_N_err = [float(x) for x in list(lf_3600s_1['Net_Area_1'][1:])]
239 lf_3600s_1_Nuclide = list(lf_3600s_1['Nuclide'][1:])
240
241 lf_3600s_2_E = [float(x) for x in list(lf_3600s_2['Centroid'][1:])]
242 lf_3600s_2_N = [float(x) for x in list(lf_3600s_2['Net_Area'][1:])]

```

```

140 lf_3600s_2_N_err = [float(x) for x in list(lf_3600s_2['Net_Area_1'][1:])]
141 lf_3600s_2_Nuclide = list(lf_3600s_2['Nuclide'][1:])
142
143 lf_3600s_3_E = [float(x) for x in list(lf_3600s_3['Centroid'][1:])]
144 lf_3600s_3_N = [float(x) for x in list(lf_3600s_3['Net_Area'][1:])]
145 lf_3600s_3_N_err = [float(x) for x in list(lf_3600s_3['Net_Area_1'][1:])]
146 lf_3600s_3_Nuclide = list(lf_3600s_3['Nuclide'][1:])
147
148 lf_36000s_E = [float(x) for x in list(lf_36000s['Centroid'][1:])]
149 lf_36000s_N = [float(x) for x in list(lf_36000s['Net_Area'][1:])]
150 lf_36000s_N_err = [float(x) for x in list(lf_36000s['Net_Area_1'][1:])]
151 lf_36000s_Nuclide = list(lf_36000s['Nuclide'][1:])
152
153 lf_86400s_E = [float(x) for x in list(lf_86400s['Centroid'][1:])]
154 lf_86400s_N = [float(x) for x in list(lf_86400s['Net_Area'][1:])]
155 lf_86400s_N_err = [float(x) for x in list(lf_86400s['Net_Area_1'][1:])]
156 lf_86400s_Nuclide = list(lf_86400s['Nuclide'][1:])
157
158 for lf in list_of_data_U238:
159     lf.columns = lf.columns.str.replace(" ", "", regex=True)
160     lf.columns = lf.columns.dropna()
161
162     ## My columns have the same name, so I rename them
163     from collections import Counter
164
165     # Create a new list of column names
166     lf_new_columns = []
167     counter = Counter()
168
169     for col in lf.columns:
170         counter[col] += 1
171         lf_new_columns.append(col if counter[col] == 1 else f"{col}_{counter[
172             col] - 1}")
173
174     lf.columns = lf_new_columns
175
176 lf_700s_U238_E = [float(x) for x in list(lf_700s_U238['Centroid'][1:])]
177 lf_700s_U238_N = [float(x) for x in list(lf_700s_U238['Net_Area'][1:])]
178 lf_700s_U238_N_err = [float(x) for x in list(lf_700s_U238['Net_Area_1'][1:])]
179 lf_700s_U238_Nuclide = list(lf_700s_U238['Nuclide'][1:])
180
181 lf_1800s_1_U238_E = [float(x) for x in list(lf_1800s_1_U238['Centroid'][1:])]
182 lf_1800s_1_U238_N = [float(x) for x in list(lf_1800s_1_U238['Net_Area'][1:])]
183 lf_1800s_1_U238_N_err = [float(x) for x in list(lf_1800s_1_U238['Net_Area_1'][1:])]
184 lf_1800s_1_U238_Nuclide = list(lf_1800s_1_U238['Nuclide'][1:])
185
186 lf_3600s_0_U238_E = [float(x) for x in list(lf_3600s_0_U238['Centroid'][1:])]
187 lf_3600s_0_U238_N = [float(x) for x in list(lf_3600s_0_U238['Net_Area'][1:])]
188 lf_3600s_0_U238_N_err = [float(x) for x in list(lf_3600s_0_U238['Net_Area_1'][1:])]
189 lf_3600s_0_U238_Nuclide = list(lf_3600s_0_U238['Nuclide'][1:])

```

```

189
190 lf_3600s_1_U238_E = [float(x) for x in list(lf_3600s_1_U238['Centroid'])[1:]]
191 lf_3600s_1_U238_N = [float(x) for x in list(lf_3600s_1_U238['Net_Area'])[1:]]
192 lf_3600s_1_U238_N_err = [float(x) for x in list(lf_3600s_1_U238['Net_Area_1']
193 ] [1:])]
194 lf_3600s_1_U238_Nuclide = list(lf_3600s_1_U238['Nuclide'])[1:]
195 lf_3600s_2_U238_E = [float(x) for x in list(lf_3600s_2_U238['Centroid'])[1:]]
196 lf_3600s_2_U238_N = [float(x) for x in list(lf_3600s_2_U238['Net_Area'])[1:]]
197 lf_3600s_2_U238_N_err = [float(x) for x in list(lf_3600s_2_U238['Net_Area_1']
198 ] [1:])]
199 lf_3600s_2_U238_Nuclide = list(lf_3600s_2_U238['Nuclide'])[1:]
200 lf_3600s_3_U238_E = [float(x) for x in list(lf_3600s_3_U238['Centroid'])[1:]]
201 lf_3600s_3_U238_N = [float(x) for x in list(lf_3600s_3_U238['Net_Area'])[1:]]
202 lf_3600s_3_U238_N_err = [float(x) for x in list(lf_3600s_3_U238['Net_Area_1']
203 ] [1:])]
204 lf_3600s_3_U238_Nuclide = list(lf_3600s_3_U238['Nuclide'])[1:]
205
206 time_from_measurement = np.array([0, 700, 2500, 6100, 9700, 13300, 16900,
207 20500, 56500, 60100])/60
208 time_from_measurement_U238 = np.array([0, 700, 2500, 6100, 9700, 13300])/60
209 time_U238 = [700, 1800, 3600, 3600, 3600, 3600]
210 time = [700, 1800, 3600, 3600, 3600, 3600, 1800, 36000, 1800, 86400]
211
212
213 # Start time in minutes
214 start_time_0 = 23*60 + 15
215
216 t1 = [start_time_0]
217 t2 = [start_time_0 + time[0]]
218
219 for i in range(1, len(time)):
220     t1.append(t2[i-1])
221     t2.append(t1[i] + time[i])
222
223 # For 'time_U238'
224 t1_U238 = [start_time_0]
225 t2_U238 = [start_time_0 + time_U238[0]]
226
227 for i in range(1, len(time_U238)):
228     t1_U238.append(t2_U238[i-1])
229     t2_U238.append(t1_U238[i] + time_U238[i])
230
231 tolerance = 1
232
233 for i in list_of_times:
234     for j in eval('lf_'+i+'_E'):
235         for k in bg_E:

```

```

236     if 0 < j - k < tolerance:
237         index_bg = bg_E.index(k)
238         index_lf = eval('lf_'+i+'_E').index(j)
239         eval('lf_'+i+'_N')[index_lf] = eval('lf_'+i+'_N')[index_lf] - bg_cps
           [index_bg]*time[list_of_times.index(i)]
240         break
241
242 for i in list_of_times_U238:
243     for j in eval('lf_'+i+'_E'):
244         for k in bg_E:
245             if 0 < j - k < tolerance:
246                 index_bg = bg_E.index(k)
247                 index_lf = eval('lf_'+i+'_E').index(j)
248                 eval('lf_'+i+'_N')[index_lf] = eval('lf_'+i+'_N')[index_lf] - bg_cps
           [index_bg]*time[list_of_times_U238.index(i)]
249                 break
250
251 ef = ef[ef['Peak efficiency'].notna()] ##Drops all NaN values
252
253 def avg_time(lambd, t1, t2):
254     return (1/lambd) - (t2 - t1) / (np.exp(lambd * (t2 - t1)) - 1) + t1
255
256 """Look at Be7 and what happens over time"""
257
258 Activity_be7_peaks = []
259 Activity_be7_peaks_err = []
260 ac_mean_ratio = 0.5867057862300689
261 be7_I = 0.1044
262 be7_I_err = 0.0004
263
264 for i in range(len(list_of_times)):
265     index_be7 = eval('lf_'+list_of_times[i]+'_Nuclide').index('Be7')
266     E = eval('lf_'+list_of_times[i]+'_E')[index_be7]
267     N = eval('lf_'+list_of_times[i]+'_N')[index_be7]
268     N_err = eval('lf_'+list_of_times[i]+'_N_err')[index_be7]
269     I = be7_I
270     I_err = be7_I_err
271     t = time[i]
272     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
273     A = (N/(t*I*eff*ac_mean_ratio))
274     Activity_be7_peaks.append(A)
275
276     ## Error propagation
277     dAdN = 1/(t*I*eff)
278     dAdI = -N/(t*I**2*eff)
279     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
280     Activity_be7_peaks_err.append(delta_A)
281
282 Activity_be7_peaks = [float(x) for x in Activity_be7_peaks]
283 Activity_be7_peaks_err = [float(x) for x in Activity_be7_peaks_err]
284
285 lambd_be7 = 1.5074E-7
286 weighted_avg_times_be7 = [avg_time(lambd_be7, start, end) for start, end in
           zip(t1, t2)]
287 weighted_avg_times_be7 = np.array(weighted_avg_times_be7)/60
288 weighted_avg_times_be7 = [float(x) for x in weighted_avg_times_be7]
289 ## Fit a curve to the data points

```

```

290 # x and y data
291 x_data = weighted_avg_times_be7 # in minutes, for example
292 y_data = Activity_be7_peaks
293 y_err = Activity_be7_peaks_err
294
295 start = 15.6
296 lamP = lamdb_be7*60
297 NPO = start/lamP
298 t = np.linspace(0, 2000, 1000)
299 NP = parent_decay(NPO, lamP, t)
300
301 plt.errorbar(x_data, y_data, yerr=y_err, fmt='yo', ecolor='red', capsize=5,
302             label='Activity  ${}^{\{7\}}\text{Be}$ ')
303 plt.plot(t, lamP * NP, 'g:', linewidth=1, label=f'Decay fit for  ${}^{\{7\}}\text{Be}$ :
304           A = {(start):.2f}')
305 plt.xlabel('Time from measurement [min]')
306 plt.ylabel('Activity [Bq]')
307 plt.legend()
308 plt.ylim(-1, 50)
309 plt.xlim(-30, 2000)
310 plt.show()
311
312 plt.yscale('log')
313 plt.errorbar(x_data, y_data, yerr=y_err, fmt='yo', ecolor='red', capsize=5,
314             label='Activity  ${}^{\{7\}}\text{Be}$ ')
315 plt.plot(t, lamP * NP, 'g:', linewidth=1, label=f'Decay fit for  ${}^{\{7\}}\text{Be}$ :
316           A = {(start):.2f}')
317 plt.xlabel('Time from measurement [min]')
318 plt.ylabel('Activity [Bq]')
319 plt.legend()
320 plt.ylim(1, 100)
321 plt.xlim(-30, 2000)
322 plt.show()
323
324 print(weighted_avg_times_be7)
325
326 """Looking at Pb212"""
327
328 Activity_pb212_peaks = []
329 Activity_pb212_peaks_err = []
330 ac_mean_ratio = 0.5867057862300689
331 pb212_I = 0.436
332 pb212_I_err = 0.005
333
334 for i in range(len(list_of_times)):
335     index_pb212 = eval('lf_'+list_of_times[i]+'_Nuclide').index('Pb212')
336     E = eval('lf_'+list_of_times[i]+'_E')[index_pb212]
337     N = eval('lf_'+list_of_times[i]+'_N')[index_pb212]
338     N_err = eval('lf_'+list_of_times[i]+'_N_err')[index_pb212]
339     I = pb212_I
340     I_err = pb212_I_err
341     t = time[i]
342     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
343     A = (N/(t*I*eff*ac_mean_ratio))
344     Activity_pb212_peaks.append(A)
345
346 ## Error propagation

```

```

343     dAdN = 1/(t*I*eff)
344     dAdI = -N/(t*I**2*eff)
345     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
346     Activity_pb212_peaks_err.append(delta_A)
347
348 Activity_pb212_peaks = [float(x) for x in Activity_pb212_peaks]
349 Activity_pb212_peaks_err = [float(x) for x in Activity_pb212_peaks_err]
350
351 lambda_pb212 = 1.8127E-5
352 weighted_avg_times_pb212 = [avg_time(lambda_pb212, start, end) for start, end
353                               in zip(t1, t2)]
354 weighted_avg_times_pb212 = np.array(weighted_avg_times_pb212)/60
355
356 # x and y data
357 x_data = weighted_avg_times_pb212 # in minutes, for example
358 y_data = Activity_pb212_peaks
359 y_err = Activity_pb212_peaks_err
360
361 plt.errorbar(x_data, y_data, yerr=y_err, fmt='ro', ecolor='red', capsize=5,
362             label='Activity  $^{212}\text{Pb}$ ')
363 plt.xlabel('Time from measurement [min]')
364 plt.ylabel('Activity [Bq]')
365 plt.legend()
366 plt.ylim(-1, 50)
367 plt.xlim(-50, 2000)
368 plt.show()
369
370 plt.yscale('log')
371 plt.errorbar(x_data, y_data, yerr=y_err, fmt='ro', ecolor='red', capsize=5,
372             label='Activity  $^{212}\text{Pb}$ ')
373 plt.xlabel('Time from measurement [min]')
374 plt.ylabel('Activity [Bq]')
375 plt.legend()
376 plt.ylim(1, 100)
377 plt.xlim(-50, 2000)
378 plt.show()
379
380 """Bi212 air filter"""
381
382 Activity_bi212_peaks = []
383 Activity_bi212_peaks_err = []
384 ac_mean_ratio = 0.5867057862300689
385 bi212_I = 0.0667
386 bi212_I_err = 0.0009
387
388 for i in range(len(list_of_times)):
389     E = eval('lf_'+list_of_times[i]+'_E')[-2]
390     N = eval('lf_'+list_of_times[i]+'_N')[-2]
391     N_err = eval('lf_'+list_of_times[i]+'_N_err')[-2]
392     I = bi212_I
393     I_err = bi212_I_err
394     t = time[i]
395     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
396     A = (N/(t*I*eff*ac_mean_ratio))/0.6406
397     Activity_bi212_peaks.append(A)
398

```

```

397     ## Error propagation
398     dAdN = 1/(t*I*eff)
399     dAdI = -N/(t*I**2*eff)
400     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
401     Activity_bi212_peaks_err.append(delta_A)
402
403     Activity_bi212_peaks = [float(x) for x in Activity_bi212_peaks]
404     Activity_bi212_peaks_err = [float(x) for x in Activity_bi212_peaks_err]
405
406     lambd_bi212 = 1.9079E-4
407     weighted_avg_times_bi212 = [avg_time(lambd_bi212, start, end) for start, end
408                                 in zip(t1, t2)]
409     weighted_avg_times_bi212 = np.array(weighted_avg_times_bi212)/60
410     weighted_avg_times_bi212 = [float(x) for x in weighted_avg_times_bi212]
411
412     x_data = weighted_avg_times_bi212 # in minutes, for example
413     y_data = Activity_bi212_peaks
414     y_err = Activity_bi212_peaks_err
415
416     plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='orange', ecolor='
417                  red', capsize=5, label='Activity  $^{212}\text{Bi}$ ')
418     plt.xlabel('Time from measurement [min]')
419     plt.ylabel('Activity [Bq]')
420     plt.legend()
421     plt.ylim(-1, 60)
422     plt.xlim(-50, 2000)
423     plt.show()
424
425     plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='orange', ecolor='
426                  red', capsize=5, label='Activity  $^{212}\text{Bi}$ ')
427     plt.xlabel('Time from measurement [min]')
428     plt.ylabel('Activity [Bq]')
429     plt.legend()
430     plt.ylim(1, 100)
431     plt.xlim(-50, 2000)
432     plt.yscale('log')
433     plt.show()
434
435     """Looking at Tl208"""
436
437     Activity_tl208_peaks = []
438     Activity_tl208_peaks_err = []
439     ac_mean_ratio = 0.5867057862300689
440     tl208_I = 0.99754
441     tl208_I_err = 0.00004
442
443     for i in range(len(list_of_times)):
444         #index_tl208 = eval('lf_'+list_of_times[i]+'_Nuclide').index('Tl208')
445         E = eval('lf_'+list_of_times[i]+'_E')[-1]
446         N = eval('lf_'+list_of_times[i]+'_N')[-1]
447         N_err = eval('lf_'+list_of_times[i]+'_N_err')[-1]
448         I = tl208_I
449         I_err = tl208_I_err
450         t = time[i]
451         eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
452         A = (N/(t*I*eff*ac_mean_ratio))/0.3594
453         Activity_tl208_peaks.append(A)

```

```

451
452     ## Error propagation
453     dAdN = 1/(t*I*eff)
454     dAdI = -N/(t*I**2*eff)
455     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
456     Activity_t1208_peaks_err.append(delta_A)
457
458 Activity_t1208_peaks = [float(x) for x in Activity_t1208_peaks]
459 Activity_t1208_peaks_err = [float(x) for x in Activity_t1208_peaks_err]
460
461 lambda_t1208 = 3.784E-3
462 weighted_avg_times_t1208 = [avg_time(lambda_t1208, start, end) for start, end
463                               in zip(t1, t2)]
464 weighted_avg_times_t1208 = np.array(weighted_avg_times_t1208)/60
465 weighted_avg_times_t1208 = [float(x) for x in weighted_avg_times_t1208]
466
467 x_data = weighted_avg_times_t1208 # in minutes, for example
468 y_data = Activity_t1208_peaks
469 y_err = Activity_t1208_peaks_err
470
471 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='pink', ecolor='red',
472             ', capsize=5, label='Activity $^{208}$Tl')
473 plt.xlabel('Time from measurement [min]')
474 plt.ylabel('Activity [Bq]')
475 plt.legend()
476 plt.ylim(-1, 50)
477 plt.xlim(-50, 2000)
478 plt.show()
479
480 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='pink', ecolor='red',
481             ', capsize=5, label='Activity $^{208}$Tl')
482 plt.xlabel('Time from measurement [min]')
483 plt.ylabel('Activity [Bq]')
484 plt.legend()
485 plt.ylim(1, 100)
486 plt.xlim(-50, 2000)
487 plt.yscale('log')
488 plt.show()
489
490 """Everything in the same plot"""
491
492 start = 29
493 lamP = lambda_pb212*60
494 lamD = lambda_bi212*60
495 lamG = lambda_t1208*60
496 NPO = start/lamP
497 NDO = start*2/lamD
498 NGO = start*3/lamG
499 #branch = 0.3594
500
501 t = np.linspace(0, 2000, 1000)
502 NP = parent_decay(NPO, lamP, t)
503 ND = daughter_decay(NPO, NDO, lamP, lamD, t)
504 NG = granddaughter_decay(NPO, NDO, NGO, lamP, lamD, lamG, t)
505
506 ## Fit a curve to the data points
507 # x and y data

```

```

505 x_data = weighted_avg_times_pb212 # in minutes, for example
506 y_data = Activity_pb212_peaks
507 y_err = Activity_pb212_peaks_err
508
509 plt.errorbar(x_data, y_data, yerr=y_err, fmt='ro', ecolor='red', capsizes=5,
              label='Activity  $^{212}\text{Pb}$ ')
510
511 x_data = weighted_avg_times_pb212 # in minutes, for example
512 y_data = Activity_bi212_peaks
513 y_err = Activity_bi212_peaks_err
514
515 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='orange', ecolor='
              red', capsizes=5, label='Activity  $^{212}\text{Bi}$ ')
516
517 x_data = weighted_avg_times_pb212 # in minutes, for example
518 y_data = Activity_tl208_peaks
519 y_err = Activity_tl208_peaks_err
520
521 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='purple', ecolor='
              red', capsizes=5, label='Activity  $^{208}\text{Tl}$ ')
522
523 plt.plot(t, lamP * NP, 'r:', linewidth=1, label=f'Decay fit for  $^{212}\text{Pb}$ 
              : A ={(start):.2f}') # Thin dotted black line
524 plt.plot(t, lamD * ND, ':', color='orange', linewidth=1, label=f'Decay fit
              for  $^{212}\text{Bi}$ : A ={(start*2):.2f}') # Thin dotted red line
525 plt.plot(t, lamG * NG, ':', color='purple', linewidth=1, label=f'Decay fit
              for  $^{208}\text{Tl}$ : A ={(start*3):.2f}') # Thin dotted blue line
526
527 plt.legend(fontsize='small')
528 plt.xlabel('Time [minutes]')
529 plt.ylabel('Activity [Bq]')
530 plt.ylim(-1, 100)
531 plt.xlim(-50, 2000)
532 plt.show()
533
534 ## Fit a curve to the data points
535 # x and y data
536 x_data = weighted_avg_times_pb212 # in minutes, for example
537 y_data = Activity_pb212_peaks
538 y_err = Activity_pb212_peaks_err
539
540 plt.errorbar(x_data, y_data, yerr=y_err, fmt='ro', ecolor='red', capsizes=5,
              label='Activity  $^{212}\text{Pb}$ ')
541
542 x_data = weighted_avg_times_pb212 # in minutes, for example
543 y_data = Activity_bi212_peaks
544 y_err = Activity_bi212_peaks_err
545
546 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='orange', ecolor='
              red', capsizes=5, label='Activity  $^{212}\text{Bi}$ ')
547
548 x_data = weighted_avg_times_pb212 # in minutes, for example
549 y_data = Activity_tl208_peaks
550 y_err = Activity_tl208_peaks_err
551
552 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', color='purple', ecolor='
              red', capsizes=5, label='Activity  $^{208}\text{Tl}$ ')

```

```

553 plt.plot(t, lamP * NP, 'r:', linewidth=1, label=f'Decay fit for  $^{212}\text{Pb}$ 
554 : A ={(start):.2f}') # Thin dotted black line
555 plt.plot(t, lamD * ND, ':', color = 'orange', linewidth=1, label=f'Decay fit
556 for  $^{212}\text{Bi}$ : A ={(start*2):.2f}') # Thin dotted red line
557 plt.plot(t, lamG * NG, ':', color = 'purple', linewidth=1, label=f'Decay fit
558 for  $^{208}\text{Tl}$ : A ={(start*3):.2f}') # Thin dotted blue line
559
560 plt.legend(fontsize='small')
561 plt.xlabel('Time [minutes]')
562 plt.ylabel('Activity [Bq]')
563 plt.ylim(0.5, 100)
564 plt.xlim(-50, 2000)
565 plt.yscale('log')
566 plt.show()
567
568 """Bi214 from U238"""
569
570 Activity_pb214_peaks = []
571 Activity_pb214_peaks_err = []
572 ac_mean_ratio = 0.5867057862300689
573 pb214_I = 0.3572
574 pb214_I_err = 0.0024
575 time_U238 = [700, 1800, 3600, 3600, 3600, 3600]
576 time_from_measurement_U238 = np.array([0, 700, 2500, 6100, 9700, 13300])/60
577
578 for i in range(len(list_of_times_U238)):
579     #index_pb214 = eval('lf_'+list_of_times_U238[i]+'_Nuclide').index('Pb212')
580     E = eval('lf_'+list_of_times_U238[i]+'_E')[0]
581     N = eval('lf_'+list_of_times_U238[i]+'_N')[0]
582     N_err = eval('lf_'+list_of_times_U238[i]+'_N_err')[0]
583     I = pb214_I
584     I_err = pb214_I_err
585     t = time_U238[i]
586     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
587     A = (N/(t*I*eff*ac_mean_ratio))
588     Activity_pb214_peaks.append(A)
589
590     ## Error propagation
591     dAdN = 1/(t*I*eff)
592     dAdI = -N/(t*I**2*eff)
593     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
594     Activity_pb214_peaks_err.append(delta_A)
595
596 Activity_pb214_peaks = [float(x) for x in Activity_pb214_peaks]
597 Activity_pb214_peaks_err = [float(x) for x in Activity_pb214_peaks_err]
598
599 lamdb_pb214 = 4.269E-4
600 weighted_avg_times_pb214 = [avg_time(lamdb_pb214, start, end) for start, end
601 in zip(t1_U238, t2_U238)]
602 weighted_avg_times_pb214 = np.array(weighted_avg_times_pb214)/60
603 weighted_avg_times_pb214 = [float(x) for x in weighted_avg_times_pb214]
604
605 ## Fit a curve to the data points
606 # x and y data
607 x_data = weighted_avg_times_pb214 # in minutes, for example
608 y_data = Activity_pb214_peaks

```

```

606 y_err = Activity_pb214_peaks_err
607
608 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', ecolor='purple', capsize
        =5, label='Activity  $^{214}\text{Pb}$ ')
609
610 Activity_bi214_peaks = []
611 Activity_bi214_peaks_err = []
612 ac_mean_ratio = 0.5867057862300689
613 bi214_I = 0.4544
614 bi214_I_err = 0.0019
615
616 for i in range(len(list_of_times_U238)):
617     #index_pb214 = eval('lf_'+list_of_times_U238[i]+'_Nuclide').index('Pb212')
618     E = eval('lf_'+list_of_times_U238[i]+'_E')[1]
619     N = eval('lf_'+list_of_times_U238[i]+'_N')[1]
620     N_err = eval('lf_'+list_of_times_U238[i]+'_N_err')[1]
621     I = bi214_I
622     I_err = bi214_I_err
623     t = time_U238[i]
624     eff = np.interp(E, list(ef['E (keV)']), list(ef['Peak efficiency']))
625     A = (N/(t*I*eff*ac_mean_ratio))
626     Activity_bi214_peaks.append(A)
627
628     ## Error propagation
629     dAdN = 1/(t*I*eff)
630     dAdI = -N/(t*I**2*eff)
631     delta_A = np.sqrt((dAdN**2)*(N_err**2)+(dAdI**2)*(I_err**2))
632     Activity_bi214_peaks_err.append(delta_A)
633
634 Activity_bi214_peaks = [float(x) for x in Activity_bi214_peaks]
635 Activity_bi214_peaks_err = [float(x) for x in Activity_bi214_peaks_err]
636
637 lambd_bi214 = 5.861E-4
638 weighted_avg_times_bi214 = [avg_time(lambd_bi214, start, end) for start, end
        in zip(t1_U238, t2_U238)]
639 weighted_avg_times_bi214 = np.array(weighted_avg_times_bi214)/60
640 weighted_avg_times_bi214 = [float(x) for x in weighted_avg_times_bi214]
641
642 # Time points (you will need to replace this with your data)
643 x_data = weighted_avg_times_pb214 # Add your time data
644 y_data = Activity_bi214_peaks # Add your observed daughter activity data
645 y_err = Activity_bi214_peaks_err
646
647 plt.errorbar(x_data, y_data, yerr=y_err, fmt='go', ecolor='red', capsize=5,
        label='Activity  $^{214}\text{Bi}$ ')
648
649 lambd_po218 = 3.730E-3
650
651 start = 150
652 lamP = lambd_po218*60
653 lamD = lambd_pb214*60
654 lamG = lambd_bi214*60
655 NP0 = start/lamP
656 NDO = start*2/lamD
657 NGO = start*2.5/lamG
658 NGO2 = start*3/lamG
659

```

```

660 t = np.linspace(0, 400, 100)
661 NP = parent_decay(NP0, lamP, t)
662 ND = daughter_decay(NP0, ND0, lamP, lamD, t)
663 NG = granddaughter_decay(NP0, ND0, NGO, lamP, lamD, lamG, t)
664 NG2 = granddaughter_decay(NP0, ND0, NGO2, lamP, lamD, lamG, t)
665
666 plt.plot(t, lamP * NP, 'k:', linewidth=1, label=f'Decay fit for  $^{218}\text{Po}$ 
      : A ={(start):.2f}') # Thin dotted black line
667 plt.plot(t, lamD * ND, 'b:', linewidth=1, label=f'Decay fit for  $^{214}\text{Pb}$ 
      : A ={(start*2):.2f}') # Thin dotted red line
668 plt.plot(t, lamG * NG, 'g:', linewidth=1, label=f'Decay fit #1 for  $^{214}\text{Bi}$ 
      $Bi: A ={(start*2.5):.2f}') # Thin dotted blue line
669 plt.plot(t, lamG * NG2, 'r:', linewidth=1, label=f'Decay fit #2 for  $^{214}\text{Bi}$ 
      ^{214}$Bi: A ={(start*3):.2f}') # Thin dotted blue line
670
671
672 plt.legend()
673 plt.xlabel('Time [minutes]')
674 plt.ylabel('Activity [Bq]')
675 plt.ylim(-10, 600)
676 plt.xlim(-10, 400)
677 plt.show()
678
679 print(Activity_pb214_peaks)
680 print(Activity_bi214_peaks)
681
682 ## Fit a curve to the data points
683 # x and y data
684 x_data = weighted_avg_times_pb214 # in minutes, for example
685 y_data = Activity_pb214_peaks
686 y_err = Activity_pb214_peaks_err
687
688 plt.errorbar(x_data, y_data, yerr=y_err, fmt='o', ecolor='purple', capsize
      =5, label='Activity  $^{214}\text{Pb}$ ')
689
690 x_data = weighted_avg_times_pb214 # Add your time data
691 y_data = Activity_bi214_peaks # Add your observed daughter activity data
692 y_err = Activity_bi214_peaks_err
693
694 plt.errorbar(x_data, y_data, yerr=y_err, fmt='go', ecolor='red', capsize=5,
      label='Activity  $^{214}\text{Bi}$ ')
695
696 plt.plot(t, lamP * NP, 'k:', linewidth=1, label=f'Decay fit for  $^{218}\text{Po}$ 
      : A ={(start):.2f}') # Thin dotted black line
697 plt.plot(t, lamD * ND, 'b:', linewidth=1, label=f'Decay fit for  $^{214}\text{Pb}$ 
      : A ={(start*2):.2f}') # Thin dotted red line
698 plt.plot(t, lamG * NG, 'g:', linewidth=1, label=f'Decay fit for  $^{214}\text{Bi}$ 
      : A ={(start*2.5):.2f}') # Thin dotted blue line
699 plt.plot(t, lamG * NG2, 'r:', linewidth=1, label=f'Decay fit #2 for  $^{214}\text{Bi}$ 
      ^{214}$Bi: A ={(start*3):.2f}') # Thin dotted blue line
700
701 plt.legend()
702 plt.xlabel('Time [minutes]')
703 plt.ylabel('Activity [Bq]')
704 plt.ylim(0.5, 1000)
705 plt.xlim(-10, 400)
706 plt.yscale('log')

```

```
707 plt.show()
```

Listing 2: Code used for data analysis of the air filter data