THE EFFECTS OF RADIATION ESCAPE ON ACCURACY AND PRECISION IN ISOTOPIC COMPOSITION DETERMINATION OF URANIUM AND PLUTONIUM WITH DECAY ENERGY SPECTROSCOPY

By

Timothy R. Ockrin

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

Houghton University

May 2024

Signature of Author……
Department of Physics
May 3, 2024

……
Dr. Katrina Koehler
Assistant Professor of Physics
Research Supervisor

……
Dr. Brandon Hoffman
Professor of Physics

THE EFFECTS OF RADIATION ESCAPE ON ACCURACY AND PRECISION IN ISOTOPIC COMPOSITION DETERMINATION OF URANIUM AND PLUTONIUM SAMPLES WITH DECAY ENERGY SPECTROSCOPY

By

Timothy R. Ockrin

Submitted to the Department of Physics
on May 3, 2024 in partial fulfillment of the
requirement for the degree of
Bachelor of Science

Abstract

Decay Energy Spectroscopy (DES) results in high energy resolution (1-5 keV @ 5 MeV) spectra of decay energies where the energy of each decay is measured as a single event as opposed to individual measurements of each decay particle. In order to accomplish this, the measured source is not external to the absorber, but embedded within it. DES can be used for nuclear safeguards, metrology, and medical isotope development, but measurements are affected by incomplete energy capture occurring when decay particles escape the absorber. In order to reduce escape likelihood, absorbers can be capped with a layer of gold. Geant4, a Monte-Carlo simulation software capable of handling energy transport between particles, is used to simulate DES measurements with varying thicknesses of absorber cap. Analysis of these spectra shows that the biggest benefit of capping can be realized by adding 5 μm of gold to all dimensions, reducing the escape of most alpha-decaying uranium and plutonium radionuclides by an order of magnitude.

Thesis Supervisor: Dr. Katrina Koehler
Title: Assistant Professor of Physics

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

INTRODUCTION

1.1. Decay Energy Spectroscopy: An Early History

Isotopic composition of radioactive samples can be determined using several well-established methods, such as alpha spectrometry [1], beta spectrometry, and gamma spectrometry [2]. One of the more novel methods is Decay Energy Spectroscopy (DES), which measures the total decay energy, rather than the energy of individual particles. An excellent history of DES measurements can be found in [3]. The history of the field under this name began in 2012 [4], although research into the technique had already begun in earnest before that point.

1.1.1. Microcalorimeters

Microcalorimeters are a type of Low Temperature Detector (LTD), any detector which is kept at very low temperatures (<1 K) in order to measure the energy of particles. Microcalorimeters accomplish this using an absorber with an embedded source to thermalize the decay products and measuring the temperature increase in some way, which can differ between unique apparatus.

Microcalorimeters are the apparatus by which DES is performed, but they can also be used for generalized alpha and beta spectrometry, as well as gamma applications [5].

1.1.2. Absorbers

An absorber is the part of a DES detector that thermalizes all particles from a decay. The material an absorber is made of determines the energy resolution of the spectrum, likelihood of event pileup, incomplete thermalization, and peak splitting, so selecting a satisfactory material for an absorber is both critical and non-trivial.

1.1.3. Transition Edge Sensors

Transition Edge Sensors (TESs) are made of various metals that become superconducting at very low temperatures [6]. In order to determine the amount of thermalization due to a given decay, they are kept just at temperatures just above superconducting (about 80 mK) on the region between superconducting and normal. In this region, the relationship between temperature and resistance is linear and measurable, which means that at a constant current or voltage bias, the energy thermalized in the absorber will be proportional to the current or voltage measured in the TES.

1.1.4. Metallic Magnetic Calorimeters

Metallic Magnetic Calorimeters (MMCs) also measure energy by detecting an increase in temperature. However, instead of measuring change in resistance, MMCs make use of paramagnets and detect a change in magnetization of a paramagnetic sensor using a Superconducting QUantum Interference Device (SQUID) [7].

1.1.5. Overview of How DES Works

DES measurements take advantage of the proportionality of several quantities to measure the energy of decays, as detailed in Figure 1. When a particle is thermalized in an absorber, the absorber temperature increases by a proportional amount. The absorber is thermally linked to a Transition Edge Sensor (TES), which is kept on the transition between normal and superconducting so the relationship between resistance and temperature is measurable and linear. By operating at a constant voltage or current bias, then, the energy of each decay can be read as proportional to the height of a pulse in either current or voltage, whichever is not being kept constant.

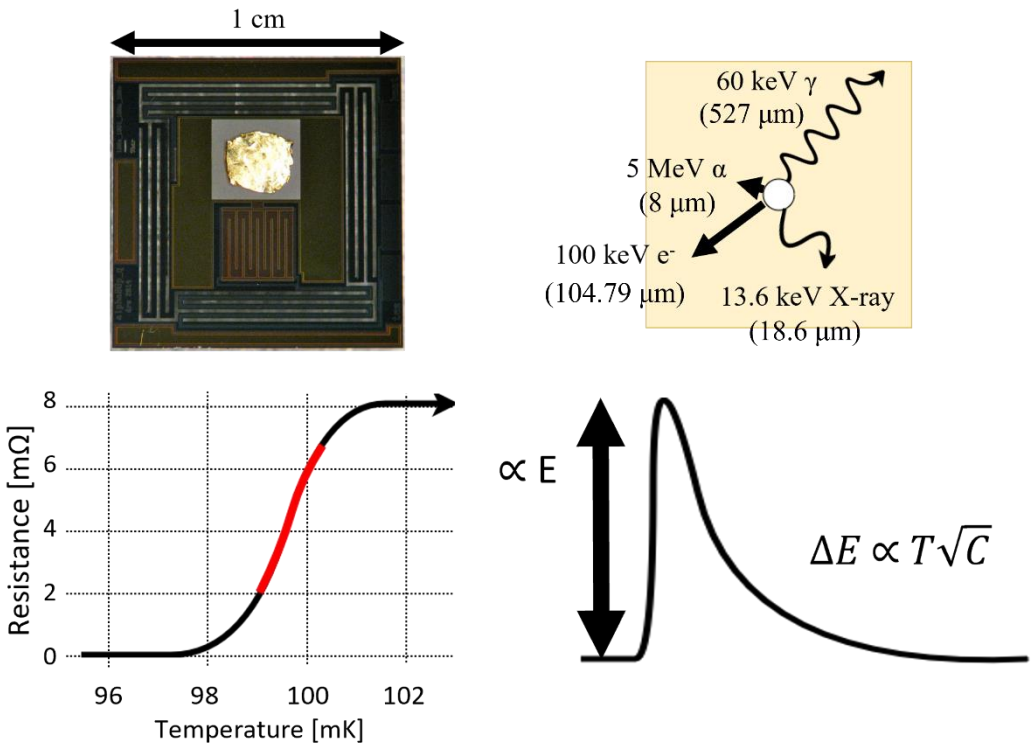


Figure 1. A detailing of the process of Decay Energy Spectroscopy (DES). A picture of a DES absorber shows the two prominent features to be a gold absorber and a superconducting Transition Edge Sensor (TES), thermally linked (upper left). The gold absorber contains an embedded source so that when a decay occurs, all particles released will be thermalized as one total event with the energy measured being the sum of all individual energies. Average stopping distance in gold of each particle type is displayed for each particle, determined using SRIM [8] (upper right). The TES is kept on the transition between normal (higher temperatures, higher resistance) and superconducting (lower temperatures, zero resistance), where the relationship between temperature and resistance is measurable and linear (lower left). At a constant current or voltage bias, the energy thermalized from a decay will be directly proportional to the height of a current or voltage pulse. The energy resolution will be proportional to the temperature at which the detector is kept and to the square root of the heat capacity of the detector (lower right).

1.2. What is the Perfect Absorber?

1.2.1. Pros and Cons of DES

DES has the potential for very high energy resolution (1-5 keV at 5 MeV), can measure extremely low activity sources (as low as 1 Bq), does not require chemical separation to measure several isotopes simultaneously, and has simplified spectra when compared to methods of spectroscopy that yield a peak in the final spectrum for each measured particle.

However, precise physical conditions are required for success. Any fluctuation in the operating temperature of the detector can yield inconsistencies in measured decay energies, inhomogeneities in the absorber resulting from embedding the source can lead to incomplete thermalizations and splitting peaks, and there is a delicate balance between reasons to make the absorber larger and keeping it smaller due to particle escapes and energy resolution loss, respectively.

1.2.2. Escapes and the Size of Absorbers

If a particle escapes the absorber without being thermalized, the energy of that particle will be lost from that decay measurement. This can lead to a separate peak for the same isotope at some amount of energy lower than the main peak that results from a common particle with that energy frequently escaping the absorber. In order to minimize escape probability, then, it would be helpful to make the absorber larger. However, the width of peaks in a DES spectrum is proportional to the specific heat of the detector, so increasing the size of the absorber will also degrade energy resolution. The traditional response to these conflicting issues is to maintain a small absorber and correct for expected escapes. The challenges and methods of doing so will be discussed in Sections 3.5.6 and 3.5.7. An alternative method for preventing escapes is introducing a cap to the absorber, which is any material surrounding the absorber with the intent of reducing particle escapes.

1.2.3. Lattice Damage and Kneading Absorbers

As previously mentioned, one issue that can cause an absorber to perform poorly is related to poorly embedding a source within it. One of the most common methods of embedding a source is to dissolve it into a solution, then letting that solution evaporate onto the absorber which is then folded over to encapsulate the source. This method is used so frequently due to the fact that it can be performed with any source and any absorber. However, it is possible that a residue from the solvent may remain in the form of a crystalline structure. If this occurs, part of the energy from a decay may be deposited not in the absorber itself, but in one of these structures. This phenomenon can lead to tailing and even peak splitting, which dramatically reduces the quality of the measurement being performed. In order to prevent this, a strategy being employed at Los Alamos National Laboratory (LANL) is to knead the

absorber after the source has been deposited [9]. This can help to break up any crystalline structures present after this deposition and lower the likelihood that energy will be stored in one of them instead of being thermalized properly.

1.2.4. The Search for the Perfect Absorber

Throughout the history of DES, several different absorber materials and deposition methods have been proposed and developed, many of which have resulted in poorer energy resolution than expected for a variety of the above reasons. This history has been documented very well in [3], but will be summarized here. The first absorber utilized for DES was made of Re, which is superconducting at about 1.7 K and naturally contains the primarily beta decaying isotope ^{187}Re, which decays primarily through beta decay. Several attempts to use this sort of detector were made by multiple groups, but results with expected energy resolutions never manifested. Another low decay energy isotope, ^{163}Ho, became the subject of focus for electron capture measurements to try to measure the neutrino mass. The initial absorber material chosen was epoxy and tin foil with a thermal link, which also suffered disappointing results. However, later measurements involving both beta and electron capture measurements began utilizing a gold foil absorber with various methods of embedding sources, including diffusion welding, dissolving and drying, and a new method involving a nanoporous gold absorber, whose small pores would limit the size of any crystalline structures formed from co-deposition to 50-100 nm. Similar methods of deposition have been utilized for high energy measurements of α-decaying particles more recently, with the recent technique mentioned above of kneading an absorber to break up crystalline deposits. Recently, the most common absorber material has remained gold foil, with the most common method being the dried drop method, likely due to its simplicity.

1.3. How Is DES Used?

1.3.1. Nuclear Safeguards

Nuclear safeguards are, according to the International Atomic Energy Agency (IAEA), a set of technical measures that allow the IAEA to independently verify a state's legal commitment not to divert nuclear material from peaceful nuclear activities to nuclear weapons or other nuclear explosive devices. Since the Treaty on Non-Proliferation of Nuclear Weapons (NPT)

in 1970, all states except for the United States, Russia, France, the United Kingdom, and China have been classified as non-nuclear-weapon states (NNWSs) and those who are party to the NPT are prohibited from manufacturing or obtaining nuclear weapons. In order to verify that this treaty is being honored by all parties, measurements of the masses and isotopic compositions of radioactive materials from NNWSs are performed to verify that no materials are being diverted from peaceful nuclear processes to weapons programs. Traditional methods for determining isotopic compositions can be separated into either Destructive Analysis methods (DA) or Nondestructive Assay (NDA). DA methods include mass spectrometry and alpha spectrometry, while the most common NDA method is gamma spectroscopy. Measuring the mass of samples requires knowledge of the isotopic composition and is usually done through neutron multiplicity counting or calorimetry. An excellent overview of these methods and their applications can be found in [10] and will be summarized here.

In general, DA methods of isotopic composition determination are more expensive and time consuming to implement, but yield more precise results. For example, mass spectrometry only has about 0.1% total error, but cannot be performed multiple times on the same source and requires extensive sample preparation each time it is utilized. Alpha spectrometry can be measured multiple times for the same sample, but has far higher error, about 10%. On the other hand, NDA methods tend to be relatively quick, cheap, and repeatedly measurable, but with lower accuracy. Gamma spectroscopy does not modify the sample being measured and takes very little time, with an accuracy of about 2%. Because of how simple a gamma measurement is, this is by far the most commonly used method of determining isotopic composition, with DA being utilized most commonly when accuracy and precision are the most important concern.

The DES method of determining isotopic composition falls somewhere between DA and NDA characteristics. It promises very high accuracy close to that of DA, with less required sample preparation. It does not approach the simplicity of gamma measurements, but does offer the ability to measure far smaller samples than possible in other methods with activities as low as 1 Bq (1 decay per second).

1.3.2. Absolute Activity

The National Institute of Standards and Technology (NIST) is a facility that develops Certified Reference Materials (CRMs) for various applications. CRMs are any sample about which the specifications are known incredibly precisely. A new CRM type that could theoretically be developed is one whose "massic activity" (unit activity per unit mass) is known to great precision [11]. In order to accomplish this, both the mass and the absolute activity of the sample must be known with less uncertainty than that of the desired massic activity. The current process by which the mass of a sample will be precisely known utilizes a gravimetric inkjet dispenser, while the activity of the sample can be determined using DES measurements due to the extremely high energy resolution promised by this method.

1.3.3. Searching for Neutrinos

One aspect of modern particle physics that eludes complete understanding is that of the neutrino. Neutrinos, unlike many particles currently being studied, are incredibly common in nature, with millions passing through an area the size of a fingernail per second [12]. Despite this abundance, they very rarely interact with matter. This has made the task of determining their resting mass one of great difficulty, as it is impractical to measure directly through these extremely infrequent interactions with matter. Recent attempts to determine an upper limit on this mass have resulted in its steady lowering from a value of 1.1 eV with a 90% confidence level in 2019 [13] to the current value of 0.8 eV with a 90% confidence level in 2022 [14]. The way that DES can help to contribute to this field is by precisely determining the functional form of beta and electron capture spectra. Since neutrinos so infrequently interact with matter, they will escape the absorber in nearly all cases. If every other product from a decay were then to be perfectly measured and compared to the total decay energy for that isotope, the mass of the escaped neutrino could be known with very low uncertainty. In order to minimize uncertainties, though, the form of the spectrum must be determined very precisely as it nears the total decay energy for the isotope being measured, since this is where the value of the rest mass of the neutrino can be determined, as seen in Figure 2. Therefore, the potential for increased precision that comes with DES may help to further determine the mass of the neutrino, as is described in [15].

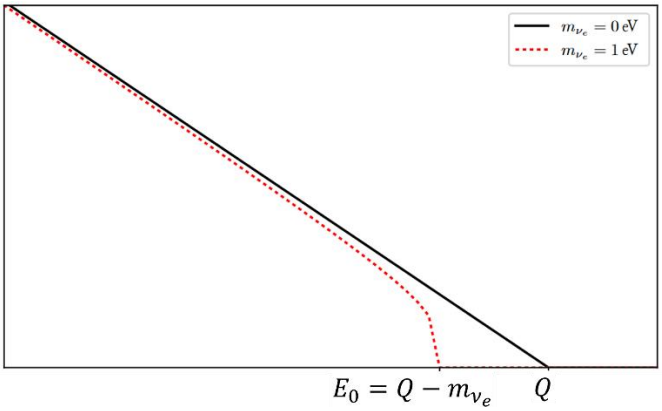


Figure 2. The theoretical end region of a beta spectrum. If the rest mass of a neutrino was 0 eV, the spectrum would end precisely at the Q value for the isotope being measured. However, since the neutrino escapes with its rest mass regardless of its kinetic energy, the spectrum actually approaches the Q value minus the rest mass of the neutrino. Precisely knowing the functional form of the high-energy region of a beta spectrum can help to determine this mass. Figure modified from Ref. [16].

1.4. How Do These Applications Matter for This Project?

The goal of this project is improving both the accuracy and precision of measuring composition of specifically alpha emitting samples, which lends itself most immediately to nuclear safeguards and absolute activity determination. The improved accuracy and precision in determining the composition of uranium samples, for example, can be used to more confidently confirm or question the declared enrichment level of a uranium sample in a nuclear facility. This would also allow the absolute activity to be measured more precisely and accurately, benefiting the study of massic activity and the implementation of nuclear safeguards.

1.5. Introduction to DESSIMATE

1.5.1. What is DESSIMATE?

Decay Energy Spectroscopy SIMulation for Absolute Total Efficiency (DESSIMATE) is a Python software that is capable of displaying spectra from simulations created using Geant4 [17,18,19] a Monte-Carlo simulation software capable of handling energy transport between particles. In addition to displaying these spectra, it is capable of incorporating detector broadening due to physical phenomena and therefore not present in these simulated data.

This allows users to see what an experimental spectrum measured using DES might look like, given certain physical parameters.

1.5.2. What Can DESSIMATE Do?

Upon inputting a spectrum created using Geant4, the data and metadata from the file will be read and displayed as a histogram, as well as any other spectra that have been read in during the same session. DESSIMATE is capable of determining from the metadata of a spectrum what the activity percentage of each isotope should be by default, and from this, determining the percent composition by mass of each isotope. Users can manipulate either the composition by activity or the composition by mass of the sample, which will automatically be normalized and used to calculate the other and display the newly determined spectrum visualization. DESSIMATE can also, given a region by the user, determine the actual number of simulated counts in this region that were from each isotope present in the total simulation. This allows for a study of how much overlap is coming from escapes when overlapping regions are a concern, which can be used to improve analysis methods of the experimental data resembling these simulated spectra to the overall goal of improving accuracy and precision in determinations of the isotopic compositions of measured samples.

Chapter 2

THEORY

2.1. What is Alpha Decay?

2.1.1. History of Alpha Decay

Alpha (α) decay was discovered by Ernest Rutherford [20], who differentiated between two types of radiation from uranium compounds and called them alpha and beta decay. He found that one form of radiation (alpha particles) was stopped very quickly by an aluminum barrier and that the other (electrons from beta decay) more readily penetrated aluminum.

Alpha decay occurs when a parent nucleus ejects an α-particle (^{4}He nucleus). The daughter nucleus will then have a lower Z number and atomic number A according to the formula

$$^{A}_{Z}\mathrm{X} \rightarrow {}^{A-4}_{Z-2}\mathrm{Y} + {}^{4}_{2}\mathrm{He},$$

where X and Y denote any two elements whose Z numbers are 2 apart, A represents some atomic number, and Z represents the Z number of element X. The primary particles released in this decay are the daughter nucleus and the α particle as in Figure 3, as well as any gamma-rays released during the deexcitation of the daughter nucleus. If the parent nucleus has bound electrons, the deexcitation of the atomic configuration can lead to ejected electrons or X-rays as well. Then, the total decay energy, determined from the mass difference between daughter and parent nuclei and also called the Q-value, is simply the energy of all of these ejected particles summed together.

2.2. Decay Product Interactions with Matter

2.2.1. Why Do We Care?

In order to accurately measure the energy of all ejected particles, these particles must not escape the absorber without being measured. One of the factors that determines the likelihood of a particle escaping a DES absorber is its energy. Higher energy particles are more likely to escape with all or some of their energy. Understanding the reasons behind this may help to predict the behavior of these particles, which can improve analysis methods. In

order to understand why this is, some of the most common interaction methods will be discussed, all of which involve the transferal of heat between particles.

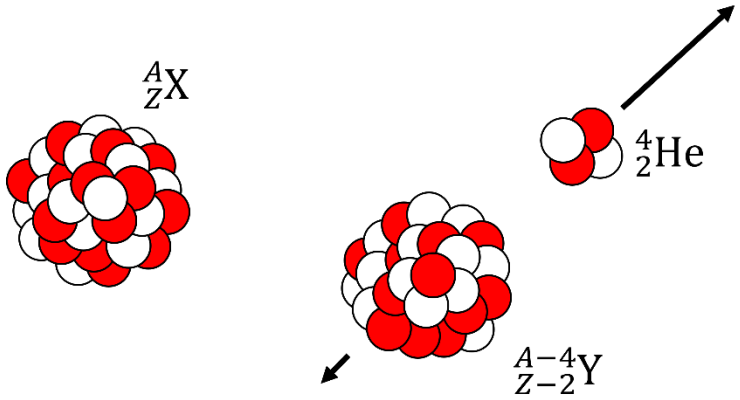


Figure 3. Visual representation of alpha decay. The leftmost particle represents the parent nucleus and the right side displays an α particle being ejected and the daughter nucleus recoiling in the opposite direction with less speed to conserve momentum.

2.2.2. What is Heat?

Thermal energy is the energy of a material that is stored in kinetic and vibrational modes. The primary energy carriers of heat are electrons (kinetic) and phonons (vibrational). In DES, thermalization refers to the process by which the energy of decay products is transferred into these modes. Thermal energy can move through the absorber and the speed at which this energy moves through a material is described by thermal conductance (G). The decay constant τ of the detector is inversely proportional to G, meaning that as G increases pulses last longer. If thermal conductance within an absorber is low enough that pulses last long enough to pileup and interfere with each other, the measurement is negatively impacted. However, if the pulses are too short, some of the energy may not arrive in the sensing mechanism quickly enough to be measured as part of the proper decay.

2.2.3. Compton Scattering

Compton scattering occurs when an incident photon interacts with some charged particle, transferring some portion of its energy to this particle resulting in a deflection path to conserve momentum, as shown in Figure 4. This phenomenon can be chained together

several times, leading to a gradual decrease in the photon's energy to the point that the energy is eventually transferred into either a vibrational or kinetic mode within the material of the absorber itself, at which point the energy is considered thermalized. However, if the photon escapes after having lost some of its energy due to Compton scattering, a Compton background can appear in the spectrum resulting from irregular amounts of energies escaping the absorber.

2.2.4. Photoelectric Effect

A similar interaction type to Compton scattering that is more favorable to measurement accuracy is the photoelectric effect, which occurs when an incident photon interacts with a material that has electrons present. The photon is absorbed and its energy transferred to an electron from that material, which is then released from its atomic structure as shown in Figure 4. This phenomenon, unlike Compton scattering, results in the full absorption of the incident photon energy. In addition, if the released photoelectron came from an orbital that was not the outermost in the atom, it is possible that the atomic structure may deexcite, causing X-rays or auger electrons to be released in addition to the photoelectron. Because few or no photons are released in this process, this usually results in the full thermalization of all resulting particles and so does not often impact the final spectrum.

2.3. Heavy Particle Interaction and Lattice Damage

The two types of heavy particles (α-particles and daughter nuclei) tend to interact with the absorber frequently, having a lower escape probability than other decay particles. However, another issue that can arise from these interactions is that of lattice damage stemming from collisions of ions produced in decays with nuclei in the absorber [21]. Some of these collisions will result in all energy being thermalized similarly to the processes for photons, but some result in energy being stored within the absorber itself as lattice damage. Because the operating temperature is so low, this damage remains indefinitely. This means that any energy stored in this lattice damage will not be thermalized and is lost in much the same way as it would have been had it escaped the absorber. When lattice damage occurs, the energy of a peak in a spectrum is lower than expected and suffers from broadening and tailing due to random distribution in amount of energy loss.

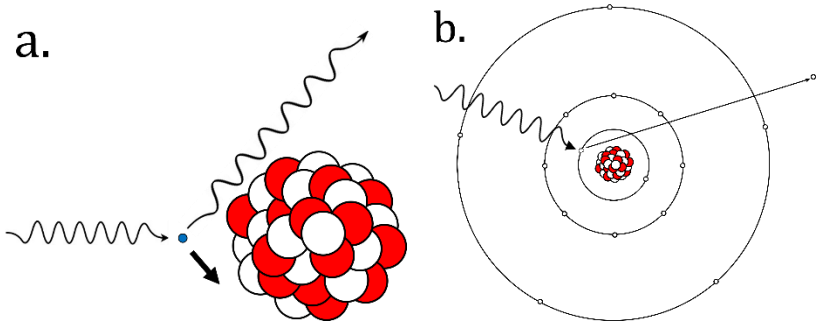


Figure 4. Pictorial representation of Compton scattering and the photoelectric effect. a) An incident photon impacts an electron and transfers momentum and kinetic energy to it, resulting in a lower energy scattered photon. b) An incident photon transfers all of its energy to a bound electron such that its kinetic energy is the energy of the incident photon minus its original binding energy. In this case, since the photoelectron did not come from the innermost orbital, it is possible that the atomic structure may deexcite, causing X-rays or auger electrons to be released as well as the photoelectron.

2.4. Geant4 Introduction

Geant4 is a Monte-Carlo simulation software capable of handling energy transport between particles. It does this by using tabulated nuclear and atomic data, such as decay branching ratios and atomic masses of isotopes. For each particle released in a simulated decay, the distribution of distance such a particle will travel through the material of the absorber will be determined, and the distance the particle will travel through the absorber is generated from this distribution. If the particle reaches this distance and is still within the absorber, an interaction with the absorber is simulated and the resulting particles continue this process until all energy has either escaped the absorber or been thermalized. For each decay, the total amount of energy thermalized in the absorber can then be recorded. With this information, one can determine what a DES spectrum might look like for the simulated scenario by graphing these energies in a histogram. The assumptions involved in this process include a perfect thermal link from the absorber to the TES, no temperature fluctuations in the detector that introduce noise to the measurement of the energy thermalized, and no detector broadening effect on the spectrum, accuracy in the tabulated branching ratios, independence between isotopes (no decay products from one isotope can interact with any other isotope present in the absorber), and no interference between decays while measuring (in other words, a complete time independence, including lack of dead time consideration).

2.5. DESSIMATE Required Functionalities

Actually visualizing the histograms of measured energies provided by Geant4, among other tasks, is the role of DESSIMATE. In order for DESSIMATE to successfully be implemented, it must be able to sum multiple spectra from individual isotopic simulations. It must also be able to determine either the activity or mass by isotope from the spectrum using a user input of relative mass or activity percentages. Finally, it must be able to determine how many counts within a given energy region correspond to which isotope.

2.6. Necessary Decay Equations

2.6.1. Generic Decay Equations

In order to determine the mass of an isotope by its activity, a relationship between the two must be established. If the atomic mass of an isotope $m_{{}^{A}_{Z}\mathrm{X}}$ and the number of present atoms of that isotope $N_{{}^{A}_{Z}\mathrm{X}}$ are known, the total present mass of that isotope $M_{{}^{A}_{Z}\mathrm{X}}$ is

$$M_{{}^{A}_{Z}\mathrm{X}} = m_{{}^{A}_{Z}\mathrm{X}} \times N_{{}^{A}_{Z}\mathrm{X}},$$

Given the half-life of an isotope ($t_{1/2}$) and defining the quantity λ (the "decay constant", or the probability of decay per nucleus per second) as $\frac{\ln 2}{t_{1/2}}$ for a particular isotope, the change in the number of atoms with respect to time $\frac{dN(t)}{dt}$ can be defined as

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

which, when integrated, yields

$$N(t) = N_0 e^{-\lambda t},$$

where N_0 is the number of nuclei present at time $t = 0$. Using the expanded form of λ now gives

$$N(t) = N_0 \times \left(\frac{1}{2}\right)^{\frac{t}{t_{1/2}}},$$

where $\frac{dN(t)}{dt}$ is the activity of the isotope, measured in decays per second or Becquerels (Bq). However, this determination assumes that a certain number of nuclei of an isotope are present at the start of the scenario and that no new parent nuclei are created, something that

can be false if the isotope in question is itself the product of another decay. In order to address this issue, the issue of decay chains must be explored.

2.6.2. Decay Chain Equations

The simple version of a decay chain is one where an unstable isotope decays into another, still unstable isotope. This decay continues until a stable isotope is reached. An example of such a decay chain is that of ^{238}U, which through a series of alpha and beta decays through elements such as thorium, polonium, and bismuth, eventually decays into stable ^{206}Pb. The numbers of atoms of each intermediary isotope at any given time is less trivial to calculate, but can be determined using differential equations. For example, the first intermediate isotope in the ^{238}U decay chain is ^{234}Th. The rate of change of the number of atoms of ^{234}Th present in a sample is

$$\frac{dN_{Th-234}(t)}{dt} = \lambda_{U-238}\mathrm{N}_{U-238}(\mathrm{t}) - \lambda_{Pa-234}\mathrm{N}_{Pa-234}(\mathrm{t}), \qquad (2)$$

where $N_{\mathrm{X}}(t)$ is the number of atoms of element X. This is because the number of atoms of ^{234}Th decreases each time a ^{234}Pa atom is created via ^{234}Th decay and increases every time a ^{238}U atom decays. This value can be calculated for any of the isotopes present in the decay chain of ^{238}U, pictured in Figure 5, at any given time if there is a time at which the isotopic composition of the sample is known.

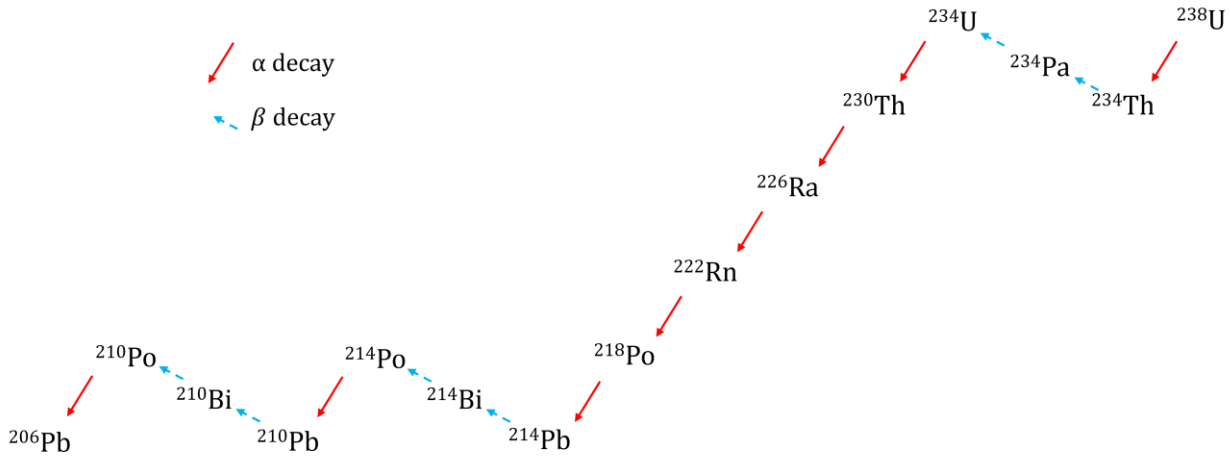


Figure 5. The most common decay chain for ^{238}U. Each isotope present in this diagram will be present in some amount in any source that contains ^{238}U until all of the ^{238}U has decayed. The decay mode of each isotope is listed and

labeled. Using the Bateman equation for isotopic composition determination, it is possible to determine the amount of each isotope present at any point in time given a time when the overall composition is known.

For uranium samples, the isotopes formed during this chain can largely be ignored, since they are either not the focus of measurements of these samples, or, in the case of other uranium isotopes, subject to long half lives in both the thorium isotope created by the initial decay (~24 days) and the uranium isotope that would be created in this chain (~4 billion years). However, for plutonium samples, one of the products of ^{241}Pu is ^{241}Am, the relative frequency of which must be derived. The complication of a plutonium sample composition over time is shown in Figure 6. In addition to this complication, it is possible that the time at which the plutonium isotopics are measured and the time at which the americium composition is known are two different times. In order to determine the amount of americium present in such a sample, then, a more complicated decay correction must be performed.

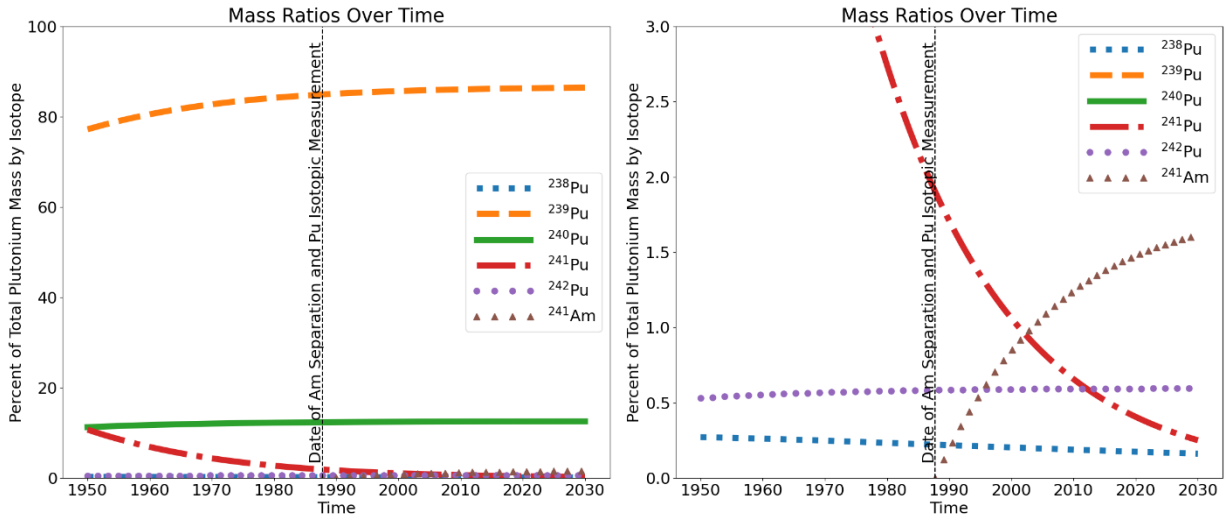


Figure 6. Plutonium isotopic composition over time. (left) all isotopes shown as a function of time, with the date at which the composition was measured precisely. (right) a zoom to the isotopes that make up less than 3% of the total plutonium mass. In this plot, the ^{241}Am can be seen growing into the sample as the ^{241}Pu present gradually decays into it.

2.6.3. Plutonium Decay Chain Equation Derivation

How to determine isotopic composition over time in plutonium sources is detailed by T. Sampson and J. Parker [22], whose process will be followed here. Implementing Equation (2) in the case of ^{241}Am yields

$$\frac{dN_{\mathrm{Am-241}}(t)}{dt} = -\lambda_{\mathrm{Am-241}} N_{\mathrm{Am-241}}(t) + K_{\mathrm{Pu-241}} \lambda_{\mathrm{Pu-241}} N_{\mathrm{Pu-241}}(t),$$

where K_{Pu-241} is the fraction of ^{241}Pu decays that lead to ^{241}Am, which is 0.9999754 [22]. Assume a solution with form

$$\mathrm{N}_{\mathrm{Am-241}}(\mathrm{t}) = Be^{-\lambda_{\mathrm{Am-241}} t} + Ce^{-\lambda_1 t}$$

and

$$\mathrm{N}_{\mathrm{Pu-241}}(t) = \mathrm{N}^{a}_{\mathrm{Pu-241}} e^{-\lambda_{\mathrm{Pu-241}}(t-t^a)},$$

where the superscript a represents a quantity at the time of the measurement of the ^{241}Am/Pu ratio and B and C are constants. The initial conditions for this solution would be that at $t = t^a$,

$$N_{\mathrm{Am-241}}(t^a) = N^{a}_{\mathrm{Am-241}}$$

and

$$N_{Pu-241}(t^a) = N^{a}_{Pu-241}.$$

The complete solution is then

$$N_{\mathrm{Am-241}}(t) = N^{a}_{\mathrm{Am-241}} e^{-\lambda_{\mathrm{Am-241}}(t-t^a)} + K_{Pu-241}\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{\mathrm{Am-241}}}\right) N^{a}_{Pu-241}\left(e^{-\lambda_{\mathrm{Am-241}}(t-t^a)} - e^{-\lambda_{Pu-241}(t-t^a)}\right), \quad (3)$$

which can be placed in terms of mass instead of atoms by noting that

$$N_{\mathrm{X}} = \frac{m_{\mathrm{X}} \cdot N_A}{A_{\mathrm{X}}},$$

where N_{X} is the number of atoms, m_{X} is the mass of element X in grams, A_{X} is the atomic mass in grams/mole, and N_A is Avogadro's number (6.022045×10^{23} mol^{-1}). Then, A_{Am-241}, the atomic mass of ^{241}Am, is 241.0568273(12) amu [23] and A_{Pu-241}, the atomic mass of ^{241}Pu, is 241.0568496(12) amu [24]. This yields the simplification of Equation (3):

$$\frac{m_{Am-241}(t)}{A_{Am-241}} = \frac{m^a_{Am-241}}{A_{Am-241}} e^{-\lambda_{Am-241}(t-t^a)} + K_{Pu-241}\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{Am-241}}\right)\frac{m^a_{Pu-241}}{A_{Pu-241}} \left(e^{-\lambda_{Am-241}(t-t^a)} - e^{-\lambda_{Pu-241}(t-t^a)}\right).$$

Defining $K_2 \equiv \frac{A_{Am-241}}{A_{Pu-241}} = 0.999607(29)$ yields $K_{Pu-241}K_2 = 0.999583(29)$, and the mass of ^{241}Am is described by

$$m_{Am-241}(t) = m^a_{Am-241}e^{-\lambda_{Am-241}(t-t^a)} + K_{Pu-241}K_2\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{Am-241}}\right)m^a_{Pu-241} \left(e^{-\lambda_{Am-241}(t-t^a)} - e^{-\lambda_{Pu-241}(t-t^a)}\right). \quad (4)$$

However, the things directly measured are not absolute masses, but the mass ratios to total plutonium. Therefore, defining m as the total plutonium mass in the sample at a time denoted by its superscript,

$$R_{Am-241}(t) = \frac{m_{Am-241}(t)}{m(t)}, \quad (5)$$

where $R_{Am-241}(t)$ is the mass ratio of ^{241}Am to all plutonium in the sample at some time t. It should be noted also that the mass decays with the same functional form as the number of atoms; that is,

$$m_i(t) = m^a_i e^{-\lambda_i(t-t^a)}$$

for isotope i of plutonium with m^a_i grams present at time $t = t^a$. The total mass of plutonium present in the sample is simply the sum of the mass of each isotope present, or

$$m(t) = \sum_i m_i(t) = \sum_i m^a_i e^{-\lambda_i(t-t^a)}, \quad (6)$$

where summing over i represents including all plutonium isotopes present in the sample. Since there are now expressions for both $m_{Am-241}(t)$ and $m(t)$ in Equations (4) and (6), respectively, Equation (5) can be rewritten as

$$R_{Am-241}(t) = \frac{m^a_{Am-241}e^{-\lambda_A(t-t^a)} + K_{Pu-241}K_2\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_A}\right)m^a_{Pu-241}e_{\text{diff}}}{\sum_i m^a_i e^{-\lambda_i(t-t^a)}}, \quad (7)$$

where e_{diff} represents the difference in the exponentials from Equation (4). Dividing the numerator and denominator by $m^a = \sum_i m_i^a$ yields

$$R_{Am-241}(t) = \frac{\frac{m^a_{Am-241}}{m^a} e^{-\lambda_{Am-241}(t-t^a)} + K_{Pu-241}K_2\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{Am-241}}\right)\frac{m^a_{Pu-241}}{m^a} e_{\text{diff}}}{\sum_i \frac{m_i^a}{m^a} e^{-\lambda_i(t-t^a)}}, \quad (8)$$

which can be further simplified using the definition of $R_i \equiv m_i/m$ to represent the mass ratio of any plutonium isotope i and total plutonium content. With this simplification, Equation (8) becomes

$$R_{Am-241}(t) = \frac{R^a_{Am-241} e^{-\lambda_{Am-241}(t-t^a)} + K_{Pu-241}K_2\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{Am-241}}\right) R^a_{Pu-241} e_{\text{diff}}}{\sum_i R_i^a e^{-\lambda_i(t-t^a)}}. \quad (9)$$

This format contains one well-known quantity (R^a_{Am-241}, or the ratio of ^{241}Am/Pu at the time of Am measurement), but also contains quantities that are not generally known already (the isotopic fractions of plutonium at the time of measurement of the americium composition). In order to account for this, we must decay correct the plutonium isotopics from the time at which they were measured to the time at which the Am content was measured. In other terms, we need to find R_i^a in terms of R_i^I, where the superscript I represents a quantity at the time of the Pu isotopic measurement. The first step in this process is simply decay correcting isotopic masses from time t^I to time t, which can simply be represented as

$$m_i(t) = m_i^I e^{-\lambda_i(t-t^I)}.$$

Since $m(t) = \sum_i m_i(t)$,

$$m(t) = \sum_i m_i^I e^{-\lambda_i(t-t^I)}.$$

Then, since $R_i(t) = \frac{m_i(t)}{m(t)}$,

$$R_i(t) = \frac{m_i^I e^{-\lambda_i(t-t^I)}}{\sum_i m_i^I e^{-\lambda_i(t-t^I)}}.$$

Similarly to the process between Equations (7) and (9), both sides of the fraction can be divided by m^I to yield

$$R_i(t) = \frac{R_i^I e^{-\lambda_i\left(t-t^I\right)}}{\sum_i R_i^I e^{-\lambda_i(t-t^I)}}. \tag{10}$$

Considering Equation (9) again, the denominator can be rewritten using this decay correction as

$$\sum_i R_i^a e^{-\lambda_i(t-t^a)} = \frac{\sum_i R_i^I e^{-\lambda_i(t^a-t^I)} e^{-\lambda_i(t-t^a)}}{\sum_i R_i^I e^{-\lambda_i(t^a-t^I)}},$$

which simplifies to

$$\sum_i R_i^a e^{-\lambda_i(t-t^a)} = \frac{\sum_i R_i^I e^{-\lambda_i(t-t^I)}}{\sum_i R_i^I e^{-\lambda_i(t^a-t^I)}}.$$

Defining

$$F(a) \equiv \sum_i R_i^I e^{-\lambda_i(t^a-t^I)}$$

and

$$F(t) \equiv \sum_i R_i^I e^{-\lambda_i(t-t^I)}$$

as the sum of the fraction of plutonium of each isotope at time a and t, respectively, means that Equation (9) has become

$$\begin{aligned} R_{Am-241}(t) = \frac{F(a)}{F(t)}\Big[R_{Am-241}^a e^{-\lambda_A(t-t^a)} + \\ K_{Pu-241}K_2\left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241}-\lambda_{Am-241}}\right)R_{Pu-241}^a e_{\text{diff}}\Big]. \end{aligned} \tag{11}$$

Having removed R_i^a from the equation, the next step is to use Equation (10) to also remove R_{Pu-241}^a in much the same way. Using the same method,

$$R_{Pu-241}^a = \frac{R_{Pu-241}^I e^{-\lambda_{Pu-241}\left(t^a-t^I\right)}}{\sum_i R_{Pu-241}^I e^{-\lambda_{Pu-241}(t^a-t^I)}} = \frac{R_{Pu-241}^I e^{-\lambda_{Pu-241}\left(t^a-t^I\right)}}{F(a)}.$$

This allows the removal of R_{Pu-241}^a in Equation (11), yielding

$$R_{Am-241}(t) = \frac{F(a)}{F(t)} R^{a}_{Am-241} e^{-\lambda_{Am-241}(t-t^{a})} + \frac{K_{Pu-241}K_2}{F(t)} \left(\frac{\lambda_{Pu-241}}{\lambda_{Pu-241} - \lambda_{Am-241}} \right) R^{I}_{Pu-241} e^{-\lambda_{Pu-241}(t^{a}-t^{I})} \left[e^{-\lambda_{Am-241}(t-t^{a})} - e^{-\lambda_{Pu-241}(t-t^{a})} \right]. \quad (12)$$

This format allows for the ratio ^{241}Am/Pu to be calculated at any point in time using only quantities commonly known due to measurement. The final quantity to be calculated, then, is the mass of plutonium at some time given that at one point the mass was known and at some other point the isotopic composition was known. The beginning of this derivation is the decay of the mass of each isotope as

$$m_i(t) = m_i^{p} e^{-\lambda_i(t-t^{p})},$$

where the superscript p refers to a quantity at the time when the plutonium mass was measured. The total mass is therefore

$$m(t) = \sum_i m_i(t) = \sum_i m_i^{p} e^{-\lambda_i(t-t^{p})},$$

and multiplying the right side by a $\frac{m^{p}}{m^{p}}$ factor yields

$$m(t) = m^{p} \sum_i \frac{m_i^{p}}{m^{p}} e^{-\lambda_i(t-t^{p})} = m^{p} \sum_i R_i^{p} e^{-\lambda_i(t-t^{p})}. \quad (13)$$

Then, from Equation (10),

$$R_i(t^{p}) = R_i^{p} = \frac{R_i^{I} e^{-\lambda_i\left(t^{p}-t^{I}\right)}}{\sum_i R_i^{I} e^{-\lambda_i(t^{p}-t^{I})}}. \quad (14)$$

Considering both Equations (13) and (14) then yields

$$m(t) = m^{p} \frac{\sum_i R_i^{I} e^{-\lambda_i(t^{p}-t^{I})} e^{-\lambda_i(t-t^{p})}}{\sum_i R_i^{I} e^{-\lambda_i(t^{p}-t^{I})}},$$

which, with $F(m) \equiv \sum_i R_i^{I} e^{-\lambda_i(t^{p}-t^{I})}$, can be expressed as

$$m(t) = \frac{m^{p} \sum_i R_i^{I} e^{-\lambda_i(t-t^{I})}}{F(m)}. \quad (15)$$

With Equation (10), Equation (12), and Equation (15), then, the plutonium isotopic composition, ^{241}Am/Pu ratio, and the plutonium mass, respectively, can be calculated at any time.

2.6.4. DESSIMATE Decay Chain Considerations

In order to see what a measurement of a source with known composition at one point in time might look like at some other point in time, DESSIMATE would need to be able to use these equations to perform a decay correction process. For a uranium source, the simpler decay chain could be used, but for a plutonium source, the method derived by Sampson would be necessary to implement. Using these corrections, though, it would be possible to find what the isotopic composition of any sample would be at any time given that it was known at some point in time.

Chapter 3

DES ANALYSIS USING MODIFIED ROI METHOD

3.1. DES Detector Overview

The central piece of a DES experiment is the absorber. Critically, the source being measured will be embedded in this absorber so that the energy of all particles released in a decay will be thermalized simultaneously. When this energy is thermalized, the absorber heats up by an amount proportional to the thermalized energy. The absorber is in thermal contact with a superconducting Transition Edge Sensor (TES), which causes this sensor to also heat up. The temperature this absorber-sensor system is kept at is the region where the TES is on the transition between normal and superconducting, usually about 80 mK, because in this region, the relationship between temperature and resistance of the TES is both measurable and linear. Therefore, at a constant voltage or current bias, a pulse can be registered from each decay measured, with the pulse's height being directly proportional to the energy measured in that decay.

3.2. Operating Temperature

In order to achieve such a low operating temperature, a cryostat with several temperature stages is implemented. The first stage goes from room temperature to ~4 K. This stage utilizes compressing and decompressing gaseous helium. The second stage utilizes an adiabatic demagnetization refrigerator (ADR), which manipulates entropy to remove heat from the location of the detector until the temperature is about 60 mK. The locations of these temperature stages in a DES cryostat are shown in Figure 7. This process actually renders the system at slightly lower temperatures than desired, so the level of the voltage bias at which the TES is maintained is determined such that the operating temperature is within the region between normal and superconducting for the TES.

3.3. Data Stream Analysis

Pictured in Figure 8 is the spectrum of measured energies determined from raw output data of a DES measurement. This spectrum is a histogram of measured energies and is the beginning of the final step in analyzing a DES measurement, but several corrections are made in order for the data to arrive in this form.

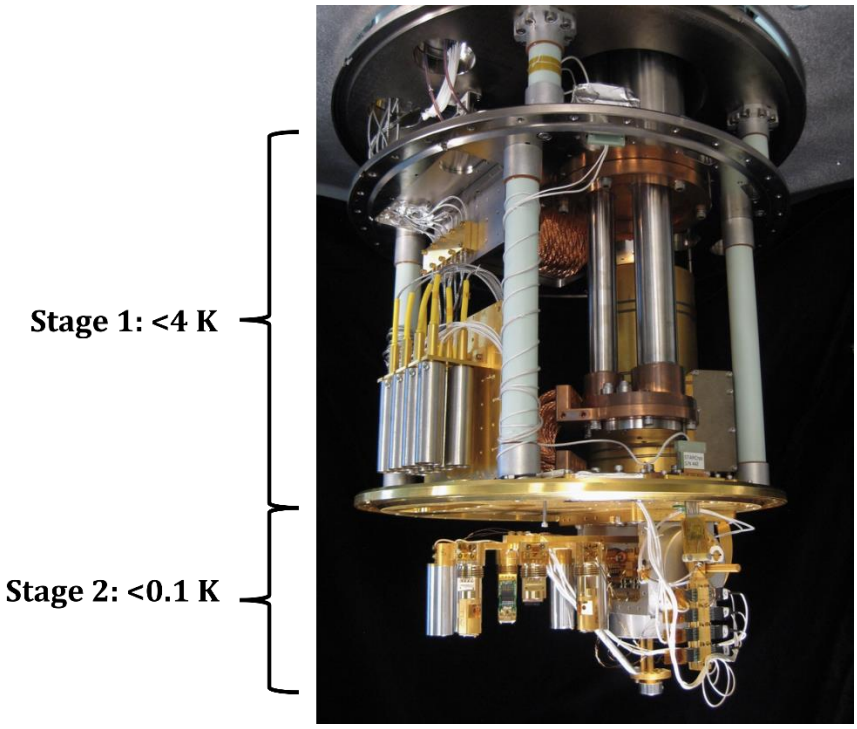


Figure 7. A picture of a DES cryostat with labeled temperature stages. The actual detector and absorber with embedded source would be located in the lowest stage, where the TES can be in the temperature region between normal and superconducting.

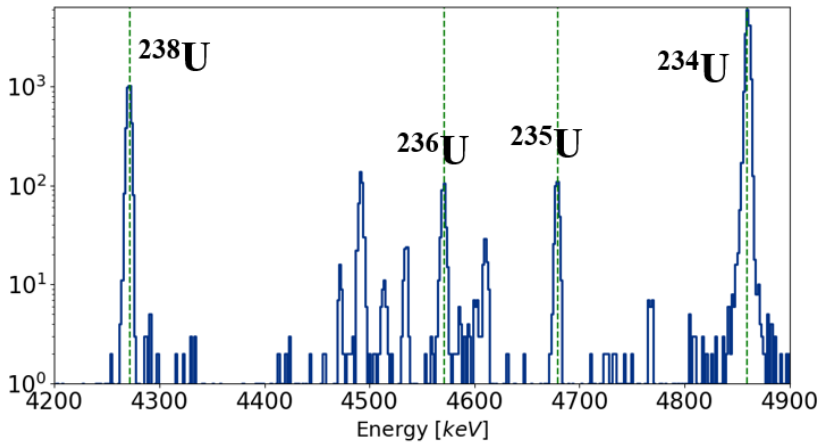


Figure 8. An experimentally measured uranium DES spectrum. The plot is a histogram of measured energies, with the x-axis representing energy and the y-axis representing the number of counts in each energy bin. The location of the Q-value for each isotope is shown as a dotted line.

3.3.1. Time Dependance

One such correction involves how decays are measured over time, which directly relates to thermal energy. Thermal energy is the energy of a material that is stored in kinetic and vibrational modes. The primary energy carriers of heat are electrons (kinetic) and phonons (vibrational). The thermal energy can move through the absorber and the speed at which this energy moves through a material is described by thermal conductance (G). If thermal conductance within an absorber is low enough that a decay occurs and its energy is thermalized while the energy from a previous decay is still moving from the absorber to the sensor, it is possible that the pulses relating to these decays will interfere with each other through pile-up. The method of handling this in analysis varies from simply disregarding all such pulses to attempting to deconvolve the counts. Therefore, any reduction in thermal conductance of the absorber will decrease energy resolution, but may also lead to increased pileup levels and dead time, because the pulse decay time has an exponential decay constant of G, impacting spectral measurements.

3.4. Geant4 Simulation Input/Output

In order to simulate a DES measurement, Geant4 input files provide the dimensions of the absorber, the material of the absorber, the distribution of the source throughout the absorber, and the isotope to be simulated, as seen in Figure 9 and *Appendix A*. The output file

contains some metadata about the simulation, such as the dimensions and material of the absorber and source within, as well as the histogram of measured energies, as seen in Figure 10.

3.5. DESSIMATE Features

3.5.1. Loading Spectra

One of the most basic features DESSIMATE must be able to perform is summing multiple spectra together. However, even this has some complications in its implementation. DESSIMATE needs to be able to confirm that all spectra are compatible with each other; that is, their energy bins must have the same widths and be aligned with each other. For this reason, when a new spectrum is loaded into DESSIMATE, the energy of its first bin and its bin width are recorded. These values are compared with the currently loaded spectra, if there are any, and the spectrum is accepted if and only if the bin width is identical and the first bin energy level is a multiple of this bin width from the current first bin energy level. If the first bin or last bin have different values from the currently existing values, any histograms that need to be extended are padded with zeroes until all spectra have the same energy bins. This allows for a visualization of all spectra together, along with a summation of them all.

```
/TrueBq01/det/setAbsorberMate G4_Au
/TrueBq01/det/setAbsorberThickness 100 um
/TrueBq01/det/setAbsorberSide 1000 um

/TrueBq01/det/setActivityThickness 0 um
/TrueBq01/det/setActivitySide 900 um
/TrueBq01/det/setActivityZOffset 0 mm

/TrueBq01/det/setChipThickness 275 um
/TrueBq01/det/setChipLength 5.0 mm
/TrueBq01/det/setChipWidth 3.0 mm
```

Figure 9. Portion of Geant4 input file example. This file shows the setting of the material and dimensions of the absorber, the dimensions of the source, and the dimensions of the chip the absorber is located on. The full input file is located in *Appendix A*.

3.5.2. Convolving Spectra

In order to simulate the peak broadening due to a physical detector performing a measurement, a convolution with a Gaussian is performed. The width of this Gaussian is adjustable by the user at any point.

3.5.3. Populating Mass/Activity Table

As spectra are uploaded to DESSIMATE, the isotopic composition of the spectrum is determined by both mass and activity and displayed in the table shown in Figure 11. In order to initially populate this table, the activities for each isotope are assumed to be the number of counts simulated. The time it would take for these counts to be simulated is determined for each isotope, the lowest of these times is considered the length of the measurement, and the actual number of counts measured in this time for each isotope is determined. These activities are normalized and converted to mass percentages by dividing by the appropriate decay constant (λ), which by Equation (1) will give the number of atoms present for each isotope. Then, simply multiplying this number by each isotope's atomic mass leaves mass amounts for each isotope, which can be normalized to determine mass percentages. The process of determining activity percentages from mass percentages simply reverses this process.

```
#title  Energy_deposit_(MeV)_in_the_absorber_and_chip
#partle Pu240
#E_MeV  0
#absorb G4_Au
#Abs s  1
#Abs t  0.097
#Act s  0
#Act t  0
#n_done 10000000
#n_set  1.000000e+07
#nbins  4002

#E_l (MeV)  counts
0   525
5.2 0
5.20002 0
5.20004 0
5.20006 0
5.20008 0
```

Figure 10. Geant4 output file example. The metadata of the simulation that created this spectrum is listed, including the isotope simulated, source distribution within the absorber, number of particles simulated, and the number of bins in the histogram. After the metadata, the file contains only the bin energies and the number of counts with energies measured in these bins as a two-column text file.

3.5.4. Updating Mass/Activity Table

In order to make DESSIMATE more versatile, this mass/activity table is editable by users. If this is done, the same process needs to be performed to determine either mass composition or activity composition given the other. This process is done identically to the initialization version, but now utilizes values input by the user.

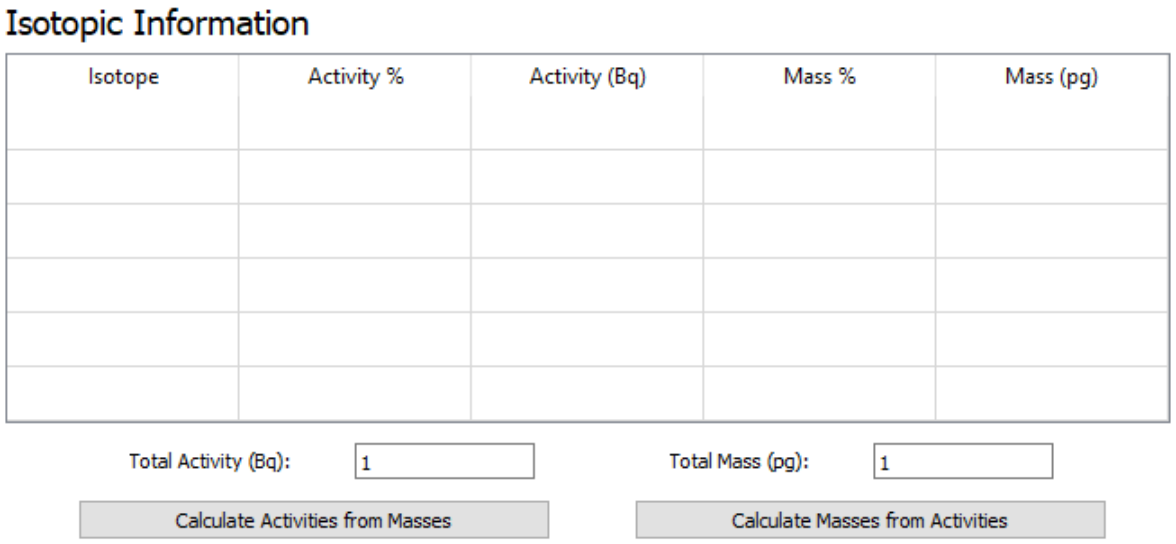


Figure 11. DESSIMATE table for isotopic composition by mass or activity. For each isotope, the raw and percentage of both mass and activity is displayed. Either of these can be updated and used to determine the other. If this happens, the percentages will be automatically normalized.

3.5.5. Determining Counts in Region

The process of determining how many counts are located in a particular energy region is critical in particular to the current DES uranium analysis method. Since each isotope has a simulated spectrum discrete from other isotopes, DESSIMATE is capable of determining not only the number of counts in a region but also the number of counts of each isotope in that region. This is done by, for each spectrum, simply summing the values in each bin located within the given energy region. While trivial, this process is critical to actually analyzing spectra from uranium sources.

3.5.6. Uranium Analysis Regions

Because there are so many escape peaks from ^{235}U, using a peak-fitting method is not the current method of analyzing uranium spectra. Instead, counts are assigned to the energy region (region of interest, or ROI) in which they are located. In order for this method to work, regions should have most of the counts of the isotope they represent and few of the counts from other isotopes. An example of such regions is shown in Figure 12. The most obvious issue with these regions is the fact that the ^{236}U region is fully embedded in the ^{235}U region.

This could be a simple fix of removing the number of counts in the ^{236}U region from the number of counts in the ^{235}U region to determine the counts of both isotopes. Unfortunately, there are escape peaks from ^{235}U that are located in the ^{236}U region, and this means that there will need to be a correction factor to determine which counts are from ^{235}U and which are from ^{236}U.

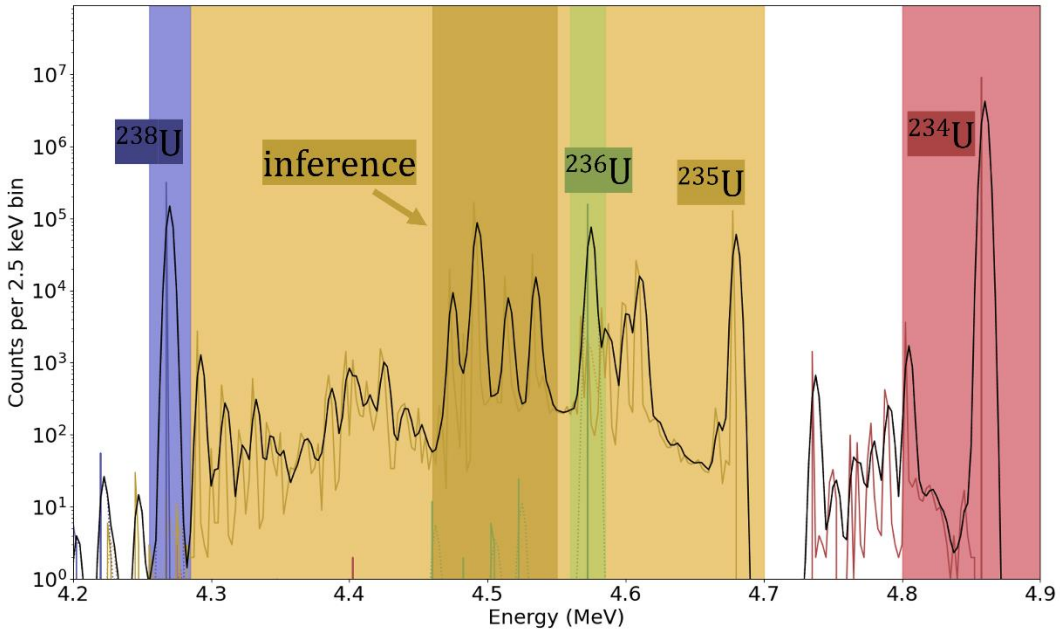


Figure 12. A simulated spectrum of a DES measurement with energy regions. The region from 4.8-4.9 MeV represents ^{234}U, 4.285-4.7 MeV represents ^{235}U, 4.56-4.585 MeV represents ^{236}U, and 4.255-4.285 MeV represents ^{238}U, with an inference region from 4.46-4.55 MeV that is used to determine the ^{235}U and ^{236}U count numbers.

3.5.7. Correction Factor

For an experimental spectrum, the breakdown of counts in a region by isotope source is not known. However, if the simulation provides an accurate representation of the spectrum from a DES measurement, the ratio

$$\frac{(235 \text{ in } 236)}{(235 \text{ in inference})}$$

should be the same in experimental spectra as in simulated spectra. Then, since the numbers of ^{235}U counts in the ^{236}U and inference regions are known for the simulated spectrum and the number of ^{235}U counts in the inference region is known for the experimental spectrum, the number of ^{235}U counts in the ^{236}U region can be determined. This can be subtracted from all counts in the ^{236}U region to determine total ^{236}U counts, which can be subtracted from all counts in the ^{235}U region to determine total ^{235}U counts and the correction factor is then complete.

Chapter 4

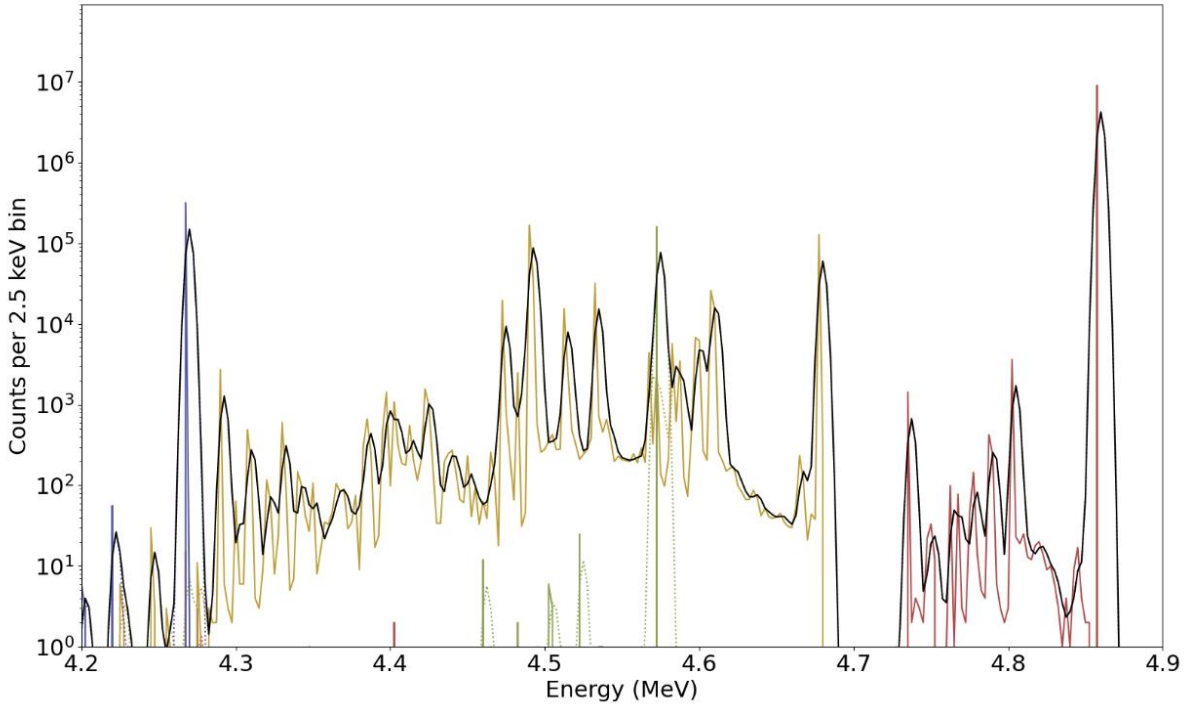
SELF-CONSISTENCY AND SENSITIVITY RESULTS

4.1. Overview

With this modified ROI analysis method for DES data explained, it follows to confirm that it will be able to determine the isotopic composition of DES spectra. However, due to lack of access to experimentally measured spectra, simulated experimental spectra from Geant4 will be analyzed using correction spectra, which will also be simulated using Geant4. This test will determine whether the modified ROI analysis method is missing any large, inaccurate assumptions. In addition, it is important to determine how dependent this ability is on the physical setup of the simulated experimental spectra being analyzed by using correction spectra with different detector dimensions than the simulated experimental spectra.

4.2. Suitability of ROIs

One of the most important assumptions of analyzing data from alpha-emitting uranium sources using DES is that the regions selected for each isotope contain most of the counts from that isotope. In high-count simulations, not capping the absorber leads to only about 96% of counts from ^{234}U being measured at energies within its region, as shown in Figure 14, which not only decreases the accuracy of ^{234}U mass determination, but also that of all other isotopes present, since the escaped counts are being measured in the regions of those isotopes, as can be seen in a comparison of Figure 13 and the figures in *Appendix C*. This means that using the ^{234}U mass as the denominator for all mass ratios is not a viable option without an absorber cap. However, adding a cap of only 10 μm raises the proportion of ^{234}U counts present in the appropriate energy region to more than 99.97%, which allows for accurate isotopic composition determination of samples by relying on ^{234}U as the denominator for mass ratios. Increasing the cap to 50 μm does not show significant improvement in this metric compared to 10-μm cap.



	Isotopic Makeup of Region					Percent of Isotope's Counts in Region			
ROI	**Total**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**
^{234}U	9015729	9015729 (100%)	0 (0%)	0 (0%)	0 (0%)	99.97%	0%	0%	0%
^{235}U	665077	3 (0%)	504686 (75.88%)	160388 (24.12%)	0 (0%)	0%	99.98%	100%	0%
^{236}U	173850	0 (0%)	13515 (7.77%)	160335 (92.23%)	0 (0%)	0%	2.68%	99.97%	0%
^{238}U	316507	3 (0%)	38 (0.01%)	0 (0%)	316466 (99.99%)	0%	0.01%	0%	99.97%

Figure 13. ROI breakdown of counts by isotope. (Above) The spectrum in question, with labeled ROIs. This spectrum represents a measurement by an absorber with 50 μm of cap of a 20% enriched source. There are 10 million total counts in this spectrum, which at 1 Bq would take ~116 days to measure. (Below) A table of counts in each ROI by isotope, with both percentage of all counts in region and percentage of all counts of that isotope.

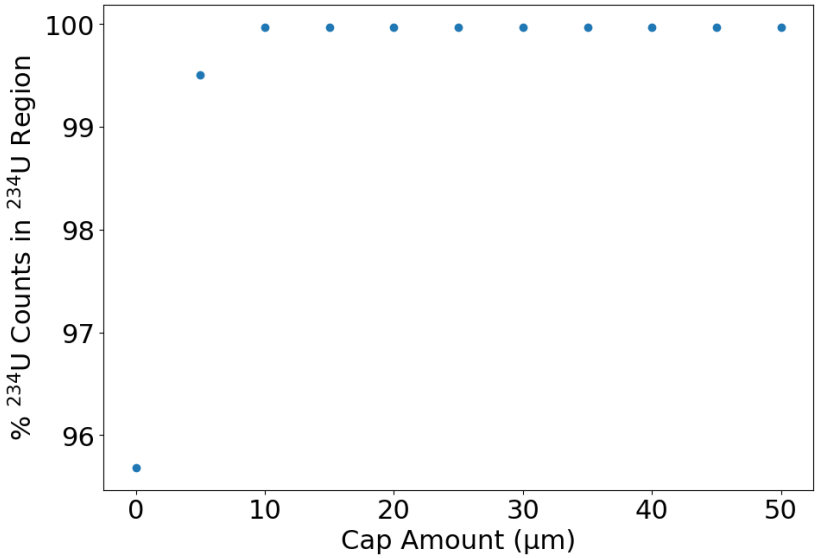


Figure 14. Normalized ^{234}U ROI counts vs. absorber cap thickness. Uncertainties are smaller than the symbol size. The percentage of ^{234}U decays in the ^{234}U ROI (4.8 – 4.9 MeV) is given for increasing absorber cap thickness. No noticeable improvement occurs in the resulting percentage between 10 μm and 50 μm cap.

4.3. Analyzing the Analysis Method

4.3.1. Self-Consistency Test

In order to ensure that the modified ROI analysis method is self-consistent, a test was performed to confirm that correction spectra can accurately determine the composition of simulated experimental spectra. These simulated experimental spectra contained 50,000 counts each across all isotopes, which represents 14-hour measurements at 1 Bq. All absorbers were simulated with dimensions 1000 μm x 1000 μm x 100 μm, with absorber cap varying from 0 μm to 50 μm. Spectra with enrichments (^{235}U percentages) of 0.02%, 1%, 1.5%, 20%, and 97% were simulated. These spectra were then used to test the analysis method, with the same spectrum being used as the simulated experimental spectrum for which mass ratios were determined and for the correction spectrum. The results from this test for 1.5% and 20% enriched sources are shown in Figure 15, while the results from all source compositions are found in *Appendix D*. In addition, accuracy and precision from this test are shown Figure 16. Results of ^{235}U/^{234}U mass ratio determination from spectra from absorbers with greater than 10-μm absorber cap show agreement to within 0.25% in all spectral compositions except the 0.02% enriched source. Results of ^{236}U/^{234}U mass ratio

determination from spectra from absorbers with greater than 10-μm absorber cap show agreement to within 1% when the spectrum contains more than trace amounts of ^{236}U. Results of $^{238}U/^{234}U$ mass ratio determination from spectra from absorbers with greater than 10-μm absorber cap show agreement to within 2% when the spectrum contains more than trace amounts of ^{238}U. The precision of these isotopic ratios is <4% for $^{235}U/^{234}U$, <6% for $^{236}U/^{234}U$, and <2.5% for $^{238}U/^{234}U$ for spectra whose compositions contain more than trace amounts of all isotopes present. This does not change appreciably with capping as it is primarily driven by the low statistics of a 14-hour measurement. The reason for decreased accuracy at 0 μm and 5 μm cap thickness is the high Compton background present in the ^{234}U trace, which adds to the counts in all ROIs, as demonstrated in Figure 17.

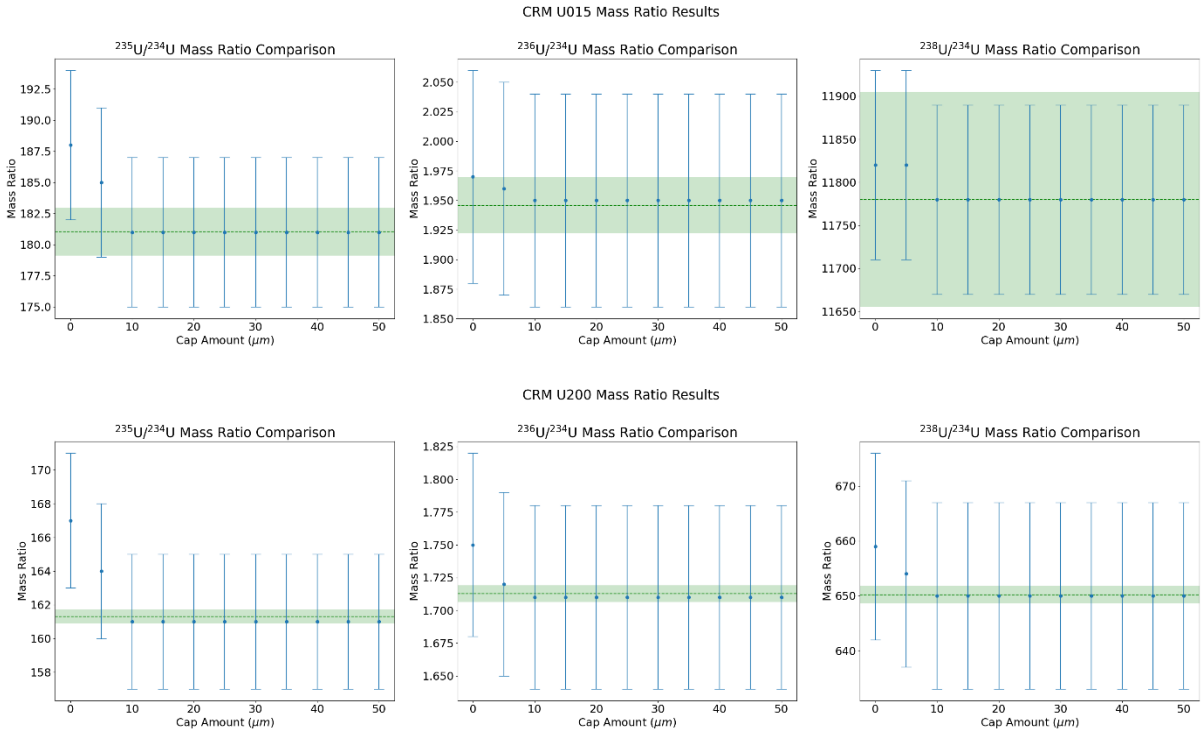


Figure 15. Results of self-consistency check for Geant4 uranium analysis method. Each column represents the mass ratio of an isotope of uranium to ^{234}U. The x-axes represent cap thickness of absorbers, from 0 μm to 50 μm, and the y-axes represent mass ratio values. The error bars come from propagated statistical and systematic uncertainties in physical constants, being dominated by statistical contributions. The mass ratios come from analysis of simulated experimental spectra with the corrected ^{235}U and ^{236}U ROIs. These values are compared to the simulated composition, denoted by the green horizontal line. The green shaded region shows the uncertainty in mass ratios that a precise experimental measurement would be compared to.

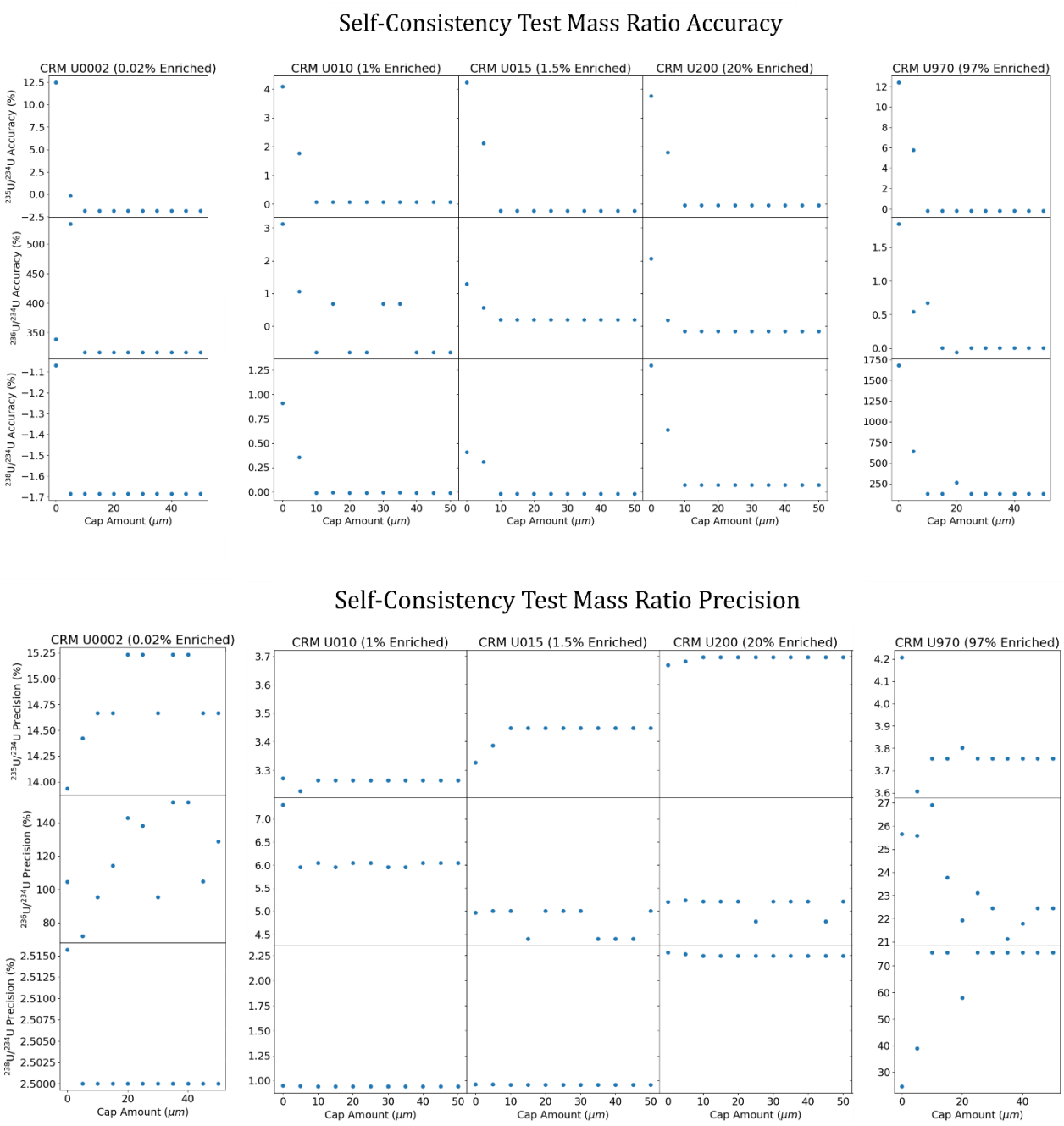


Figure 16. Accuracy and precision of self-consistency test results. Isotopes whose percentage by activity are below 0.002% are removed as trace isotopes, as their accuracy and precision are worse than those isotopes that are present in higher quantities.

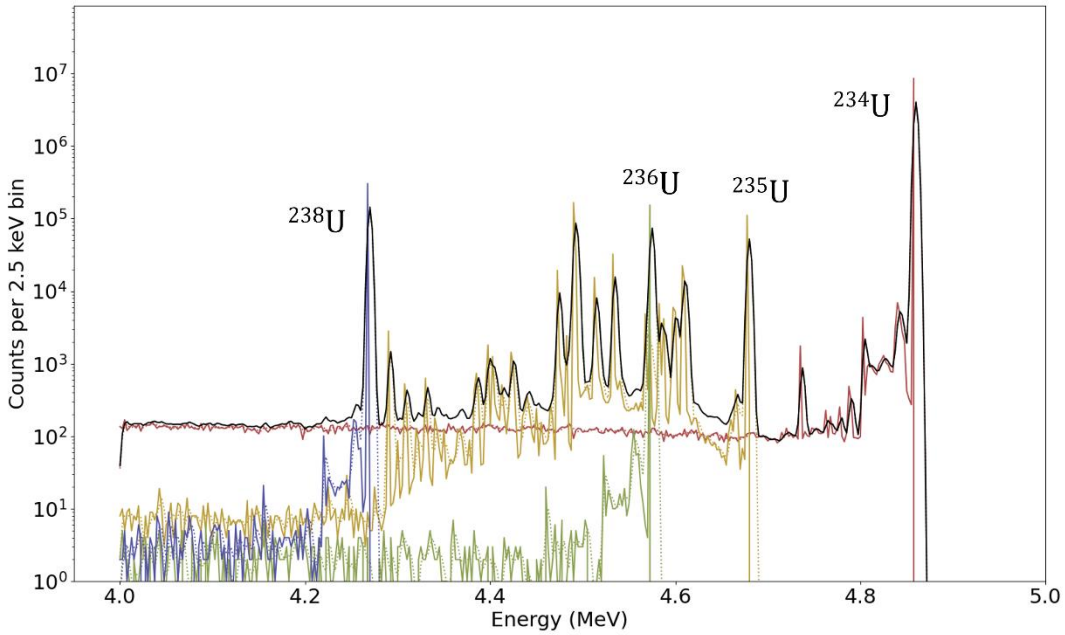


Figure 17. A spectrum simulated with no absorber cap. The high Compton background present in the ^{234}U trace interferes with analysis.

4.3.2. Introducing Systematic Uncertainty

Because of variability in absorber dimension and source distribution, another tested quality of the modified ROI method was the sensitivity to physical setup. Instead of analyzing spectra with correction spectra simulated with an identical physical situation, spectra were simulated for identical enrichments but different absorber configurations than the spectra to be analyzed. Once again, the spectra to be analyzed were simulated with 50,000 total counts for enrichments of 0.02%, 1%, 1.5%, 20%, and 97%. However, the correction spectra were simulated with 10,000,000 total counts each, which can be done because the correction spectra need not have an experimentally plausible number of counts. In one case, the correction spectra were simulated with 50 μm absorber cap and in the other, they were simulated with 0 μm absorber cap. The results of the 1.5% and 20% enriched samples are shown in Figure 18 and Figure 19 and the full results are shown in *Appendix E*. The accuracies of these results are shown in Figure 20.

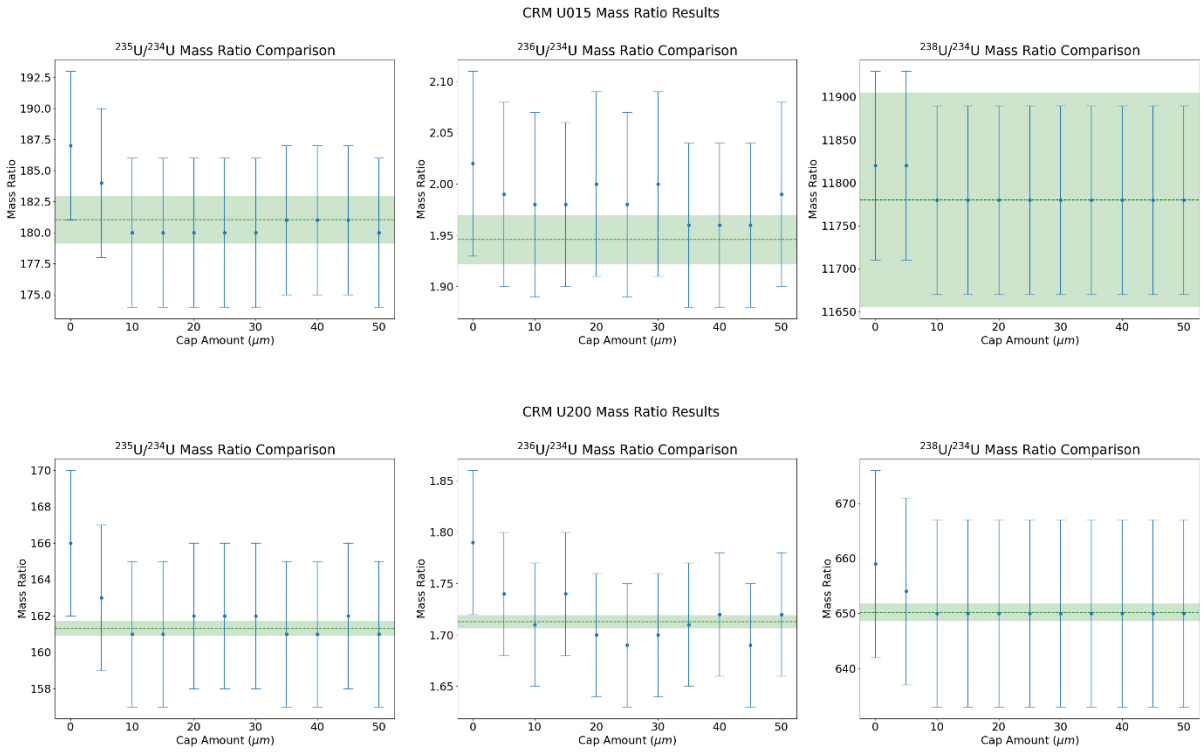


Figure 18. Results of analysis method assuming minimal escape. Correction spectra for this test used absorbers with 50 µm of absorber cap. Simulated experimental spectra varied in cap amounts, from 0 µm to 50 µm.

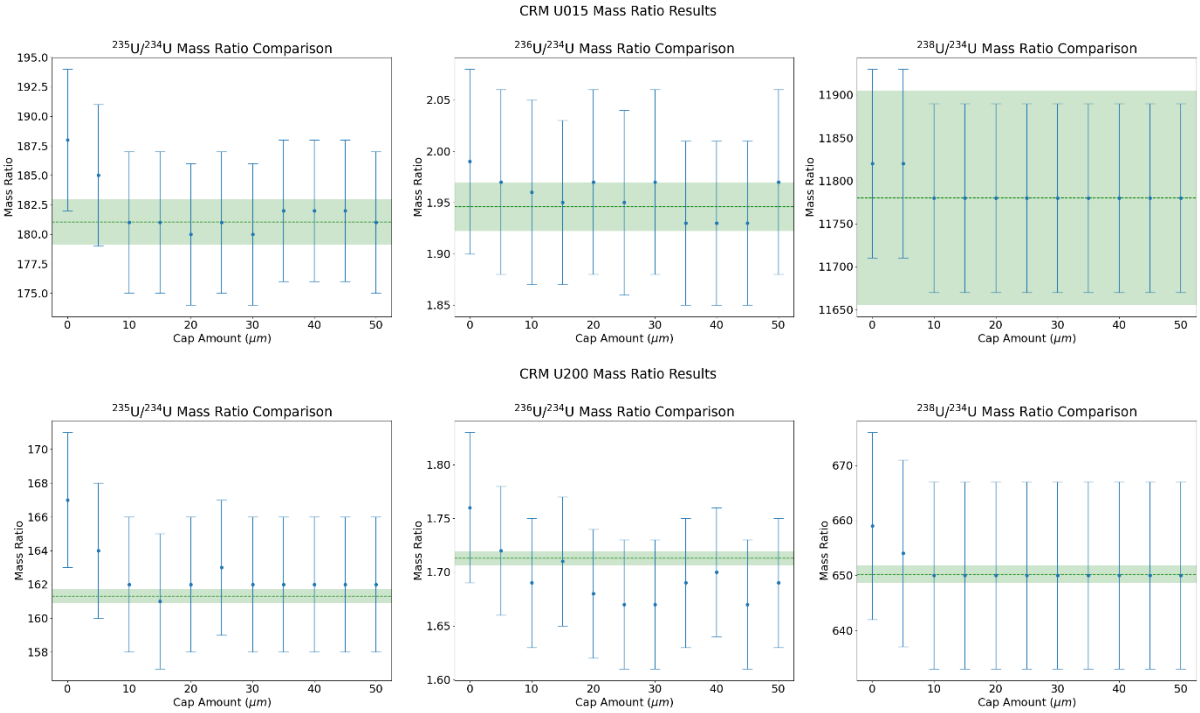
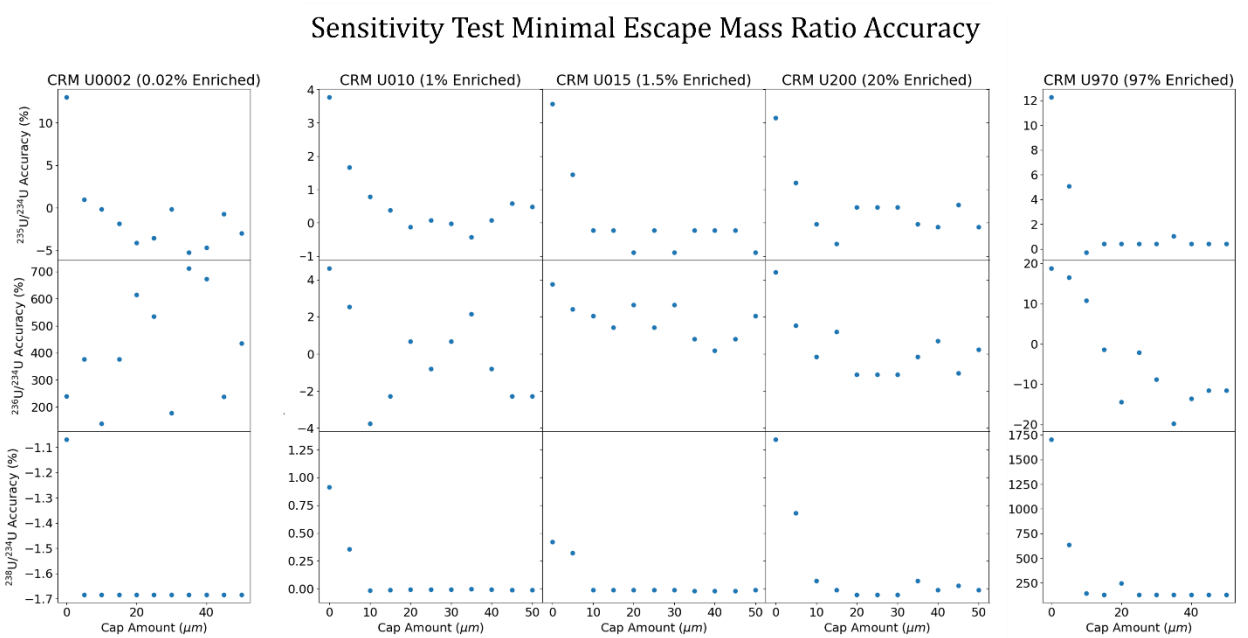


Figure 19. Results of analysis method assuming maximal escape. Correction spectra for this test used absorbers with 0 μm of absorber cap. Simulated experimental spectra varied in cap amounts, from 0 μm to 50 μm.



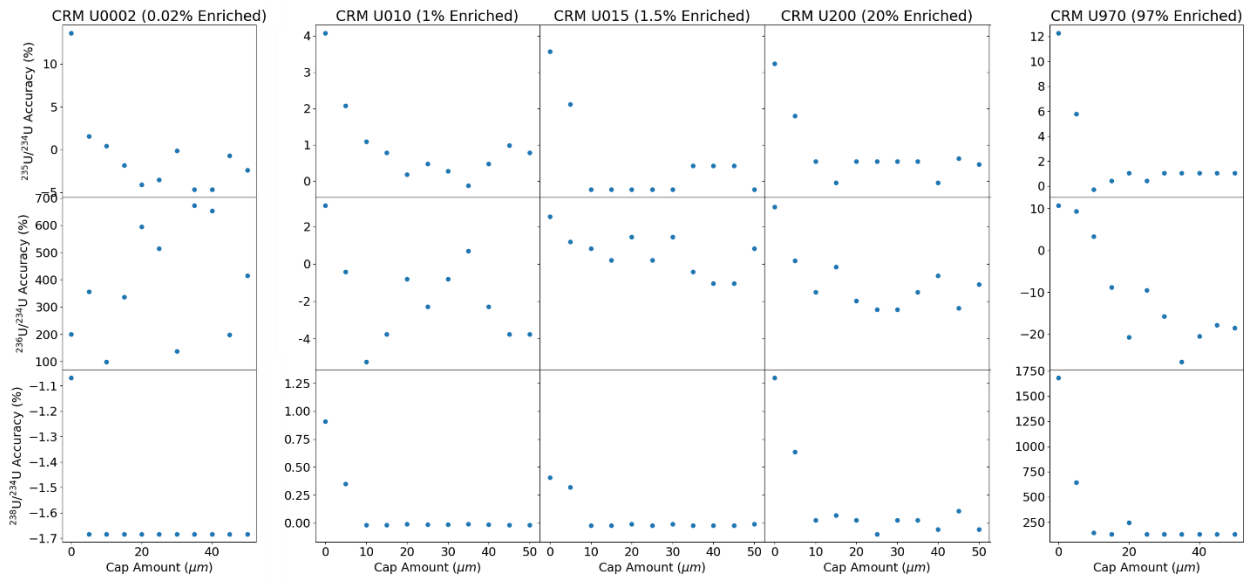


Figure 20. Accuracy and precision of sensitivity test results. Isotopes whose percentage by activity are below 0.002% are removed as trace isotopes, as their accuracy and precision are worse than those isotopes that are present in higher quantities.

Chapter 5

CONCLUSION

5.1. How Can We Use These Results?

The results of this study indicate that the modified ROI method of analyzing DES uranium data can accurately determine the isotopic compositions of spectra simulated using Geant4 to an average accuracy of <1% and an average precision of ~5%. This assumes absorbers have at least a 10-μm cap. Without this cap, the accuracy of the modified ROI method is above 10% error for some spectra. As a result, the recommendation is to increase capping to 10 μm. In addition, to improve precision, the recommendation is to increase count rate, since the current uncertainties are dominated by statistical contributions.

5.2. What About Future Plans?

5.2.1. Aluminum Absorbers

At Los Alamos National Laboratory, aluminum absorbers are currently being explored due to aluminum's much lower heat capacity than gold, allowing for potential improvements to energy resolution and mass loads. However, the lower Z number of aluminum would also result in a lower stopping power, potentially increasing particle escapes. As a result, simulation of how problematic these escapes are can be used to optimally design this absorber type in two ways. One, this approach can be used to optimize the capping thickness necessary for aluminum. Two, simulations could be used to determine whether a gold or aluminum cap is preferable, dependent on whether the stopping power of gold or the lower heat capacity of aluminum is more beneficial.

5.2.2. Detector Aspect Ratio Simulation Variety

Only one detector shape and size has been simulated using Geant4, but no real detector has exactly that geometry. Simulating various detector dimensions could help to determine further how sensitive to detector geometry the modified ROI method is.

5.2.3. Randomization in Geant4 Simulation

For all simulations presented in this work, the random seed was identical, resulting in no variation between simulations beyond geometry. These simulations should be recreated with different random seeds for the Geant4 simulations.

5.2.4. Alternate Analysis Method

As an alternate method for analyzing DES data, a library of Geant4 simulated spectra could be created, encompassing a wide range of detector and absorber geometries. These resulting spectra could then be used in a fitting process to determine the isotopic composition of experimental spectra.

5.2.5. Plutonium Simulation

This process of analyzing the effects of escape on spectral shape can also be used for plutonium sources. In order to analyze plutonium spectra, peak fitting techniques or a modified ROI method similar to the uranium method described here could be used for analyzing plutonium measurements. Alternatively, if the method of fitting spectra using a library of simulated spectra is successful, this method could also be employed to analyze plutonium measurements.

Appendix A

COMPLETE GEANT4 INPUT FILE

```
/control/verbose 0
/run/verbose 1
/event/verbose 0
/tracking/verbose 0

/TrueBq01/det/setAbsorberMate G4_Au
/TrueBq01/det/setAbsorberThickness 100 um
/TrueBq01/det/setAbsorberSide 1000 um

/TrueBq01/det/setActivityThickness 0 um
/TrueBq01/det/setActivitySide 900 um
/TrueBq01/det/setActivityZOffset 0 mm

/TrueBq01/det/setThetaMin 0 deg
/TrueBq01/det/setThetaMax 180 deg

/TrueBq01/det/setChipThickness 275 um
/TrueBq01/det/setChipLength 5.0 mm
/TrueBq01/det/setChipWidth 3.0 mm

/process/em/lowestElectronEnergy 1 keV
/run/initialize

/gun/particle ion
/gun/energy 0 MeV
/TrueBq01/det/ParentOnly TRUE
/process/had/rdm/setPhotoEvaporationFile 92 235 UserEvapData_z92.a235
/process/had/rdm/setPhotoEvaporationFile 90 229 UserEvapData_z90.a229

/gun/ion 94 240
/analysis/setFileName Pu-240
/analysis/h1/set 2 4000 5.200 5.280 MeV
/run/beamOn 10000000
```

Appendix B

DESSIMATE FULL CODE

B.1. DESSIMATE.py

```
import sys
from PyQt5 import QtWidgets, QtCore
from PyQt5.uic import loadUi

import matplotlib
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 22})

import numpy as np
import pandas as pd
import uncertainties
from uncertainties.core import ufloat, ufloat_fromstr
import seaborn as sns
from sympy import sqrt, symbols

# Local Functions and Classes
from util import read_two_col_text_file, create_gaussian_output, create_isotopic_data_frame, get_counts_in_regions, get_isotopic_information_tables, get_mass_from_activity, get_activity_from_mass, upload_sim_spectrum
from help import Ui_DESSIMATE_Help



class Ui(QtWidgets.QMainWindow):
    def __init__(self):
        super(Ui, self).__init__() # Call the inherited classes __init__ method

        # Load the .ui file and perform basic setup
        loadUi('DESSIMATE.ui', self)
        self.setWindowTitle("DESSIMATE")
        self.addToolBar(NavigationToolbar(self.graph.canvas,self))

        # Connect functions to triggers in the UI
        self.action_open.triggered.connect(self.open_file)
        self.set_logarithmic.triggered.connect(lambda: self.set_axis_scale('log'))
        self.set_linear.triggered.connect(lambda: self.set_axis_scale('linear'))
        self.go_to_spectrum_tab.triggered.connect(lambda: self.switch_tabs(0))
        self.go_to_experimental_tab.triggered.connect(lambda: self.switch_tabs(1))
        self.update_detector_response_button.pressed.connect(self.update_graph)
        self.scaling_toggle.triggered.connect(self.toggle_spectrum_scaling)
        self.action_help.triggered.connect(self.load_help_file)
        self.normalize_by_activity_button.pressed.connect(self.update_table_by_activity)
        self.normalize_by_mass_button.pressed.connect(self.update_table_by_mass)
```

```
        self.get_counts_in_region.triggered.connect(self.determine_counts_in_region_by_isotope)
        self.uranium_analysis_button.triggered.connect(self.uranium_analysis)
        self.actionSave_Graph_Figure.triggered.connect(self.savefig_graph)

        self.spectrum_one_field.textChanged.connect(self.upload_spectrum_one)
        self.spectrum_two_field.textChanged.connect(self.upload_spectrum_two)
        self.spectrum_three_field.textChanged.connect(self.upload_spectrum_three)
        self.spectrum_four_field.textChanged.connect(self.upload_spectrum_four)
        self.spectrum_five_field.textChanged.connect(self.upload_spectrum_five)
        self.spectrum_six_field.textChanged.connect(self.upload_spectrum_six)

        self.spectrum_one_clear.pressed.connect(self.clear_spectrum_one)
        self.spectrum_two_clear.pressed.connect(self.clear_spectrum_two)
        self.spectrum_three_clear.pressed.connect(self.clear_spectrum_three)
        self.spectrum_four_clear.pressed.connect(self.clear_spectrum_four)
        self.spectrum_five_clear.pressed.connect(self.clear_spectrum_five)
        self.spectrum_six_clear.pressed.connect(self.clear_spectrum_six)

        # Initialize global variables
        self.energies = []
        self.total_raw_spectrum = []
        self.convolved_spectrum = []
        self.isotope_names = ['','','','','','']
        self.simulated_count_numbers = [0,0,0,0,0,0]
        self.scale_spectra = False
        self.plotting_palette = sns.color_palette(palette = 'tab20b')
        self.plotting_colors = {0: np.array(self.plotting_palette[13]), 1: self.plotting_palette[9], 2:
self.plotting_palette[5], 3: self.plotting_palette[1], 4: self.plotting_palette[17], 5: 'orange'}
        self.isotopic_df = create_isotopic_data_frame()
        self.length_of_measurement = 0.
        self.graph.axes.set_yscale('log')
        self.max_scaling_factor = 10
        self.uranium_mode = False


        ### TEMPORARY DICTIONARIES OF GENERIC ISOTOPIC INFORMATION
        self.half_life_dict, self.mass_dict = get_isotopic_information_tables()

        # Show the GUI
        self.show()



    def load_help_file(self):
        # Opens a separate UI window containing information that guides a new user into using DESSIMATE
        help_app = QtWidgets.QMainWindow()
        help_ui = Ui_DESSIMATE_Help()
        help_ui.setupUi(help_app)
        help_app.show()
        app.hide()



    # Graph formatting functions
    def set_axis_scale(self, scale_type):
```

```
        self.graph.axes.set_yscale(scale_type)
        if scale_type == 'linear':
            self.graph.axes.set_ylim(bottom=0)
            self.max_scaling_factor = 1.35
        if scale_type == 'log':
            self.graph.axes.set_ylim(bottom=1)
            self.max_scaling_factor = 10
        self.update_graph()

    def toggle_spectrum_scaling(self):
        self.scale_spectra = not self.scale_spectra
        self.update_graph()

    def switch_tabs(self, tab_num):
        self.tab_widget.setCurrentIndex(tab_num)


    # Spectrum loading functions

    def open_file(self):
        # Determine which file should be opened and read its contents in as text
        name = QtWidgets.QFileDialog.getOpenFileName(self, 'Open File')
        self.upload_spectrum(event = name[0], slot_number = 0)

    def upload_spectrum_one   (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 0)
            self.update_graph()
            self.spectrum_one_field.setText(str(self.isotopic_df['Maximum   Counts'].iloc[-1])  +  '  counts  of  '  +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()

    def upload_spectrum_two   (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 1)
            self.update_graph()
            self.spectrum_two_field.setText(str(self.isotopic_df['Maximum   Counts'].iloc[-1])  +  '  counts  of  '  +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()

    def upload_spectrum_three (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 2)
            self.update_graph()
            self.spectrum_three_field.setText(str(self.isotopic_df['Maximum   Counts'].iloc[-1])  +  '  counts  of  '  +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()

    def upload_spectrum_four  (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 3)
            self.update_graph()
            self.spectrum_four_field.setText(str(self.isotopic_df['Maximum   Counts'].iloc[-1])  +  '  counts  of  '  +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()
```

```
    def upload_spectrum_five  (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 4)
            self.update_graph()
            self.spectrum_five_field.setText(str(self.isotopic_df['Maximum  Counts'].iloc[-1]) + ' counts of ' +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()

    def upload_spectrum_six   (self, event):
        if event[-4:] == '.out':
            self.upload_spectrum(event = event, slot_number = 5)
            self.update_graph()
            self.spectrum_six_field.setText(str(self.isotopic_df['Maximum  Counts'].iloc[-1]) + ' counts of ' +
self.isotopic_df['Isotope'].iloc[-1])
            self.populate_activity_table()



    def upload_spectrum(self, event, slot_number):
        """
        This function is called each time a new spectrum is uploaded into DESSIMATE. It reads the contents of the
file and parses
        several pieces of information: the isotope name, number of simulated counts, and actual energy spectrum
of those simulated counts.
        It then pads this spectrum with 75 bins above the maximum energy of the simulation to allow for future
detector broadening and
        stores the total raw spectrum in a global array of each raw spectrum.

        Parameters
        ----------
        event : string
            The filepath of the new spectrum that has been input into DESSIMATE.
        slot_number : int
            The field number that the spectrum was input into.

        Returns
        -------
        None.

        """
        if event[-4:] == '.out':

            # Read the dropped file's contents in as text
            with open(event, 'r') as file:
                file_text = file.readlines()

            # Parse the text by line, looking for keywords describing the spectrum
            for line_num, line in enumerate(file_text):
                if line[0] == '#':
                    # Found a keyword
                    var_name = line.split('\t')[0].split('#')[1]
                    var_value = line.split('\t')[1].split('\n')[0]
                    if var_name.split(' ')[0] == 'E_l':
                        # Read in spectrum file
```

```
                spectrum = read_two_col_text_file(file_text[line_num+1:])
                spectrum = [spectrum[0][1:], spectrum[1][1:]]

                if len(self.energies) != 0:

                    # Check to make sure new energy array is aligned with old energy array
                    new_intercept, old_intercept = self.check_if_valid_binning(spectrum)

                    # Pad end of spectrum with zeroes
                    highest_old_energy = spectrum[0][-1]
                    padding_index = np.arange(1,75)
                    new_energies = spectrum[0]
                    new_energies_padded = np.append(new_energies, highest_old_energy+(padding_index*self.energy_step_size))

                    # Create new energies array for all of the spectra together
                    spectrum = self.pad_spectra(spectrum, new_intercept, old_intercept, new_energies_padded[-1], self.energies[-1])

                else:

                    self.energy_step_size = spectrum[0][1] - spectrum[0][0]
                    padding_index = np.arange(1, 75)
                    highest_old_energy = spectrum[0][-1]
                    self.energies = np.append(spectrum[0], highest_old_energy + (padding_index * self.energy_step_size))

            elif var_name == 'partle':
                isotope_name = var_value
                self.isotope_names[slot_number] = isotope_name

            elif var_name == 'n_done':
                num_max_counts = int(float(var_value))

        # Create new row for data frame and append to the full isotopic data frame
        new_row = pd.DataFrame(data = {'Isotope': isotope_name, 'Maximum Counts': num_max_counts, 'Time Taken for Maximum Counts (in seconds)': 0., 'Measured Counts': num_max_counts, 'Total Atoms': 0., 'Activity %': 0., 'Activity (Bq)': 0., \
                        'Mass (pg)': 0., 'Mass %': 0., 'Percentage Counts Measured': 1, 'Raw Spectrum': [np.append(spectrum[1], np.zeros(74))], 'Unconvolved Spectrum': [[]], 'Convolved Spectrum': [[]], \
                        'Plotting Color': [self.plotting_colors[slot_number]]}, index = [len(self.isotopic_df['Isotope'])], dtype = object)

        self.isotopic_df = pd.concat([self.isotopic_df, new_row])



    def check_if_valid_binning(self, spectrum):

        new_slope = round(spectrum[0][1] - spectrum[0][0], 10)
        old_slope = round(self.energies[1] - self.energies[0], 10)

        if new_slope != old_slope:
            raise Exception(f'The bin width of this spectrum is not the same as that of the other spectra. The bin width of this spectrum was {new_slope} and the bin width of the other spectra is {old_slope}.')
```

```
            return 0

        self.energy_step_size = new_slope

        new_intercept = spectrum[0][0]
        old_intercept = self.energies[0]

        slope_differences = round(abs(new_intercept - old_intercept), 10) % self.energy_step_size # Determines if the intercepts are separated by an integer multiple of the slope

        if slope_differences > 0.0000001 and abs(slope_differences - self.energy_step_size) > 0.0000001:
            # The energy bins are not compatible; please input a new spectrum instead
            raise Exception('The bins of this spectrum are offset from the other spectra by a non-integer multiple of the bin width and are therefore incompatible. This spectrum was not added to the plotted total. Please try a different spectrum.')
            return 0

        return new_intercept, old_intercept



    def pad_spectra(self, spectrum, new_intercept, old_intercept, highest_new_energy, highest_old_energy):

        low_energy = min(old_intercept, new_intercept)
        high_energy = max(highest_old_energy, highest_new_energy)

        if old_intercept > new_intercept:
            num_paddings = (old_intercept - new_intercept) / self.energy_step_size
            padding_index = np.zeros(round(num_paddings))
            for row_num in range(len(self.isotopic_df['Isotope'])):
                self.isotopic_df.loc[row_num, 'Raw Spectrum'] = np.append(padding_index, self.isotopic_df.loc[row_num, 'Raw Spectrum'])

        if new_intercept > old_intercept:
            num_paddings = (new_intercept - old_intercept) / self.energy_step_size
            padding_index = np.zeros(round(num_paddings))
            spectrum[1] = np.append(padding_index, spectrum[1])

        if round(highest_new_energy, 10) > round(highest_old_energy, 10):
            num_paddings = (highest_new_energy - highest_old_energy) / self.energy_step_size
            padding_index = np.zeros(round(num_paddings))
            for row_num in range(len(self.isotopic_df['Isotope'])):
                appended = np.append(self.isotopic_df.loc[row_num, 'Raw Spectrum'], padding_index)
                self.isotopic_df.loc[[row_num], 'Raw Spectrum'] = pd.Series([appended], index = [row_num])

        if round(highest_old_energy, 10) > round(highest_new_energy, 10):
            num_paddings = (highest_old_energy - highest_new_energy) / self.energy_step_size
            padding_index = np.zeros(round(num_paddings))
            spectrum[1] = np.append(spectrum[1], padding_index)

        self.energies = np.arange(low_energy, round(round(high_energy, 10) + round(self.energy_step_size, 10), 10), self.energy_step_size)

        return spectrum
```

```
def update_graph(self):
    """
    This function is called whenever something changes what should be graphed in DESSIMATE, most notably each time a new spectrum is uploaded or an existing
    spectrum is cleared from DESSIMATE. It begins by resetting the graph in order to prevent leaving previous removed lines in the plot. It then retrieves
    pertinent information such as the detector response size and energy step size of the spectra which will be used to label the plot and accurately portray
    the spectra. It then plots each spectrum separately and the total raw spectrum (with the ability to scale it to a level that is the same as the eventual
    convolved spectrum) as well.

    Returns
    -------
    None.

    """
    # Remove old plots of the total spectrum before convolution
    for line in self.graph.axes.lines[::-1]: # Loops backwards to avoid removing an index and changing an index while looping
        if 'Raw Spectrum' in line.get_label(): line.remove()

    # Read in detector FWHM and energy step size
    self.detector_response_size = float(self.detector_response_size_user_input_field.text())
    self.energy_step_size = self.energies[2] - self.energies[1]

    # Graph each spectrum before convolution, scaling if necessary
    if isinstance(self.isotopic_df.loc[0, 'Measured Counts'], uncertainties.UFloat):
        self.isotopic_df['Percentage Counts Measured'] = [self.isotopic_df.loc[row_num, 'Measured Counts'].nominal_value / self.isotopic_df.loc[row_num, 'Maximum Counts'] for row_num in range(len(self.isotopic_df['Isotope']))]
    else:
        self.isotopic_df['Percentage Counts Measured'] = self.isotopic_df['Measured Counts'] / self.isotopic_df['Maximum Counts']

    self.isotopic_df['Unconvolved Spectrum'] = self.isotopic_df['Raw Spectrum'] * self.isotopic_df['Percentage Counts Measured']


    # Draw uranium regions if applicable
    if self.uranium_mode:
        self.graph.axes.axvspan(self.measured_counts_df['ROI Bounds'][0][0], self.measured_counts_df['ROI Bounds'][0][1], color = self.plotting_palette[14], alpha = 0.5)
        self.graph.axes.axvspan(self.measured_counts_df['ROI Bounds'][1][0], self.measured_counts_df['ROI Bounds'][1][1], color = self.plotting_palette[10], alpha = 0.5)
        self.graph.axes.axvspan(self.measured_counts_df['ROI Bounds'][2][0], self.measured_counts_df['ROI Bounds'][2][1], color = self.plotting_palette[6], alpha = 0.5)
        self.graph.axes.axvspan(self.measured_counts_df['ROI Bounds'][3][0], self.measured_counts_df['ROI Bounds'][3][1], color = self.plotting_palette[2], alpha = 0.5)
        self.graph.axes.axvspan(self.measured_counts_df['ROI Bounds'][4][0], self.measured_counts_df['ROI Bounds'][4][1], color = self.plotting_palette[9], alpha = 0.5)

    # Format graph
```

```
    self.graph.axes.set_xlabel('Energy (MeV)')
    self.graph.axes.set_ylabel(f'Counts per {np.round((self.energy_step_size)*1000, decimals = 4)} keV bin')
    self.graph.canvas.draw()

    self.include_detector_response()



  def include_detector_response(self):
    """
    This function accounts for the effect of a detector and how its spectra appear differently from purely simulated data due to a broadening effect that results from
    its imperfections. The way this effect is mimicked is through a convolution with a Gaussian curve, which is the reason for the padding of each raw spectrum previously
    in the process of inputting a spectrum. In order to do this, a Gaussian curve is formed using an external function, given amplitude, sigma, and center (each determined
    primarily from the value of the detector response size, which can be manually changed by the user; changing this will automatically replot this convolved spectrum).

    Returns
    -------
    None.

    """

    # Sum each input spectrum and define as total raw spectrum
    self.total_raw_spectrum = sum(self.isotopic_df['Unconvolved Spectrum'])

    # Check whether there is a spectrum imported yet
    if sum(self.total_raw_spectrum) == 0:
      raise Exception('There is no spectrum to correct. Please input a file before adjusting the contents of this text box.')
      return 0

    # Remove any previous broadened spectra lines and labels from the graph and legend
    for line in self.graph.axes.lines[::-1]: # Loops backwards to avoid removing an index and changing all future indices
      if line.get_label() in ['Convolved Spectrum']: line.remove()

    if self.detector_response_size == 0:
      # There is no detector response included; return unconvolved spectrum
      self.isotopic_df['Convolved Spectrum'] = self.isotopic_df['Unconvolved Spectrum']

    else:
      # Create the inputs for the Gaussian response curve
      gaussian_inputs = np.arange(0, (len(self.total_raw_spectrum)-1)*self.energy_step_size, self.energy_step_size) - ((len(self.total_raw_spectrum)-1)/2)*self.energy_step_size

      # Define parameters for outputs and create outputs for the Gaussian response curve
      sigma = self.detector_response_size/2.355
      amplitude = 1./(sigma*np.sqrt(2*np.pi))
      center = 0
      gaussian_func = create_gaussian_output(amplitude, center, sigma, gaussian_inputs) * self.energy_step_size
```

```
        # Convolve and plot convolved spectrum
        convolved_spectra = []
        for spectrum_number in range(len(self.isotopic_df['Isotope'])):
            convolved_spectrum = np.convolve(self.isotopic_df.loc[spectrum_number, 'Unconvolved Spectrum'], gaussian_func, mode = 'same')
            convolved_spectra.append(convolved_spectrum)
        self.isotopic_df['Convolved Spectrum'] = convolved_spectra

    self.convolved_spectrum = sum(self.isotopic_df['Convolved Spectrum'])

    for row_num in range(len(self.isotopic_df['Isotope'])):
        if self.scale_spectra and not self.detector_response_size == 0:
            self.graph.axes.plot(self.energies, self.isotopic_df.loc[row_num, 'Unconvolved Spectrum']*(self.energy_step_size/self.detector_response_size),\
                    label=f'{self.isotopic_df.loc[row_num, "Isotope"]} Raw Spectrum',color=self.isotopic_df.loc[row_num, 'Plotting Color'])
        else:
            self.graph.axes.plot(self.energies, self.isotopic_df.loc[row_num, 'Unconvolved Spectrum'], label = f'{self.isotopic_df.loc[row_num, "Isotope"]} Raw Spectrum', color = self.isotopic_df.loc[row_num, 'Plotting Color'])

        self.graph.axes.plot(self.energies, self.isotopic_df.loc[row_num, 'Convolved Spectrum'], label = f'{self.isotopic_df.loc[row_num, "Isotope"]} Raw Spectrum', color = self.isotopic_df.loc[row_num, 'Plotting Color'], linestyle = ':')

    self.graph.axes.plot(self.energies, self.convolved_spectrum, label = 'Convolved Spectrum', color = 'black')
    if self.scale_spectra and not self.detector_response_size == 0:
        # Important note: self.max_scaling_factor is set to 1.25 if the graph is linear and 10 if the graph is logarithmic
        self.graph.axes.set_ylim(1, max(self.total_raw_spectrum)*self.energy_step_size/self.detector_response_size*self.max_scaling_factor)
    else:
        self.graph.axes.set_ylim(1, max(self.total_raw_spectrum)*self.max_scaling_factor)
    self.graph.axes.legend(fontsize = 15)
    self.graph.canvas.draw()



  def clear_spectrum_one(self):
    field_text = self.spectrum_one_field.text()
    row_number = self.find_row_number(field_text)
    self.spectrum_one_field.setText(' ')
    self.clear_spectrum(row_num = row_number, spectrum_num = 0)

  def clear_spectrum_two(self):
    field_text = self.spectrum_two_field.text()
    row_number = self.find_row_number(field_text)
    self.spectrum_two_field.setText(' ')
    self.clear_spectrum(row_num = row_number, spectrum_num = 1)

  def clear_spectrum_three(self):
    field_text = self.spectrum_three_field.text()
    row_number = self.find_row_number(field_text)
    self.spectrum_three_field.setText(' ')
    self.clear_spectrum(row_num = row_number, spectrum_num = 2)
```

```
    def clear_spectrum_four(self):
        field_text = self.spectrum_four_field.text()
        row_number = self.find_row_number(field_text)
        self.spectrum_four_field.setText(' ')
        self.clear_spectrum(row_num = row_number, spectrum_num = 3)

    def clear_spectrum_five(self):
        field_text = self.spectrum_five_field.text()
        row_number = self.find_row_number(field_text)
        self.spectrum_five_field.setText(' ')
        self.clear_spectrum(row_num = row_number, spectrum_num = 4)

    def clear_spectrum_six(self):
        field_text = self.spectrum_six_field.text()
        row_number = self.find_row_number(field_text)
        self.spectrum_six_field.setText(' ')
        self.clear_spectrum(row_num = row_number, spectrum_num = 5)

    def find_row_number(self, field_text):
        isotope_name = field_text.split()[3]
        row_number = np.where(self.isotopic_df['Isotope'] == isotope_name)[0][0]
        return row_number

    def clear_spectrum(self, row_num, spectrum_num):
        # Removes all global variables connected to a given spectrum when it is removed from DESSIMATE
        self.isotopic_df = self.isotopic_df.drop(labels = row_num, axis = 0)
        self.isotopic_df.reset_index(drop = True, inplace = True)
        self.populate_activity_table()

        self.update_graph()



    def populate_activity_table(self):
        """
        This function calculates the isotopic composition of a given sample and displays the results of this
calculation in a table. The two main things it calculates
        are the percentage composition of each isotope by activity and the percentage composition of each isotope
by mass. In addition to automatically calculating both
        of these things each time a spectrum is uploaded or removed, the user will be able to manually change the
activity percentage of any isotope in the sample and
        see the effect of such a change on the mass percentage of the sample by isotope.

        Returns
        -------
        None.

        """
        if sum(self.total_raw_spectrum) == 0:
            raise Exception('There is no spectrum to analyze. Please input a file before adjusting the contents of this
table.')
            return 0

        self.isotopic_composition_table.clear()
```

```
    total_num_counts = sum(self.isotopic_df['Maximum Counts'])

    self.isotopic_df['Activity %'] = self.isotopic_df['Maximum Counts']/total_num_counts
    self.isotopic_df['Activity (Bq)'] = self.isotopic_df['Activity %'] * float(self.total_activity_number.text())

    self.isotopic_df['Time Taken for Maximum Counts (in seconds)'] = total_num_counts/float(self.total_activity_number.text())

    self.isotopic_df['Total Atoms'] = self.isotopic_df['Activity (Bq)']*[self.half_life_dict[isotope_name] for isotope_name in self.isotopic_df['Isotope']]/np.log(2)
    self.isotopic_df['Mass (pg)'] = self.isotopic_df['Total Atoms']*[self.mass_dict[isotope_name]*1000000000 for isotope_name in self.isotopic_df['Isotope']]
    total_mass = sum(self.isotopic_df['Mass (pg)'])

    self.isotopic_df['Mass %'] = self.isotopic_df['Mass (pg)']/total_mass

    column_names = ['Isotope', 'Activity %', 'Activity (Bq)', 'Mass %', 'Mass (pg)']
    self.isotopic_composition_table.setHorizontalHeaderLabels(column_names)

    for column_num, column_name in enumerate(column_names):
      for row_num, cell_value in enumerate(self.isotopic_df[column_name]):

        table_item = QtWidgets.QTableWidgetItem()
        if type(cell_value) == str:
          table_item.setText(cell_value)
        else:
          if column_name == 'Mass %' or column_name == 'Activity %':
            table_item.setText(f'{100*cell_value:.3f}%')
          else:
            table_item.setText(f'{cell_value:.3f}')
        self.isotopic_composition_table.setItem(row_num, column_num, table_item)

    self.total_mass_number.setText(f'{total_mass:.3f}')



  def determine_counts_in_region_by_isotope(self, energy_region_unconvolved_min = 4.5727, energy_region_unconvolved_max = 4.573, energy_region_convolved_min = 4.55, energy_region_convolved_max = 4.59):
    """
    This function, given an energy region and several Geant output files, determines the number of counts due to each isotope in that region.
    From there, it prints each of these numbers and also the percentages of the total for each isotope.

    Returns
    -------
    None.

    """

    if len(self.isotopic_df['Isotope']) == 0:
      raise Exception('No spectra found; please input a spectrum first.')
      return 0
```

```
    self.counts_dict = {'Isotope': [], 'Variable': [], 'Expression': [], 'Convolved Counts': [], 'Convolved Uncertainty': [], 'Convolved Percentage': [], \
                'Unconvolved Counts': [], 'Unconvolved Uncertainty': [], 'Unconvolved Percentage': []}
    self.counts_df = pd.DataFrame(data = self.counts_dict)

    self.counts_df['Isotope'] = self.isotopic_df['Isotope']

    self.counts_df['Unconvolved Counts'], self.counts_df['Convolved Counts'] = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], \
                                        [energy_region_unconvolved_min, energy_region_unconvolved_max], [energy_region_convolved_min, energy_region_convolved_max])

    self.counts_df['Unconvolved Uncertainty'] = [np.sqrt(cts) for cts in self.counts_df['Unconvolved Counts']]
    self.counts_df['Convolved Uncertainty'] = [np.sqrt(cts) for cts in self.counts_df['Convolved Counts']]

    # Prepare a string of each isotope name, separated by spaces, for creating variables
    symbols_string = ''
    for isotope_num, isotope_name in enumerate(self.isotopic_df['Isotope']):
      symbols_string += isotope_name
      if isotope_num < (len(self.isotopic_df['Isotope']) - 1): symbols_string += ' '


    self.counts_df['Variable'] = symbols(symbols_string) # Define variables for each isotope

    # Create lambda function to create an expression for the percentage of counts
    create_expression = lambda x: x / (sum(self.counts_df['Variable']))
    self.counts_df['Expression'] = create_expression(self.counts_df['Variable'])

    self.propagate_uncertainties()

    num_counts_convolved = [ufloat(i, np.sqrt(i)) for i in self.counts_df['Convolved Counts']]

    #percentage_counts_convolved = [i / sum(num_counts_convolved) for i in num_counts_convolved]

    #for i in range(len(self.isotopic_df['Isotope'])):
     #       print(f'Number of counts of {self.isotopic_df.loc[i, "Isotope"]} in convolved region: {num_counts_convolved[i]}.')
      #      print(f'Percentage of counts of {self.isotopic_df.loc[i, "Isotope"]} in convolved region: {percentage_counts_convolved[i]:0.3f}%.')
       # print()



  def propagate_uncertainties(self):

    convolved_uncertainties = []
    unconvolved_uncertainties = []

    for expression in self.counts_df['Expression']:

      # Determine the uncertainty in the expression for the given values
      squared_total_convolved_unc = 0
      squared_total_unconvolved_unc = 0
```

```
        # Loop through each variable to add the uncertainty in that variable squared times the derivative of the function squared
        for variable, unconvolved_individual_unc, convolved_individual_unc in zip(self.counts_df['Variable'], self.counts_df['Unconvolved Uncertainty'], self.counts_df['Convolved Uncertainty']):
            derivative_convolved = expression.diff(variable)
            derivative_unconvolved = expression.diff(variable)

            expression_convolved = expression
            expression_unconvolved = expression

            # Loop through each variable to substitute each value into the derivative
            for variable, unconvolved_counts, convolved_counts in zip(self.counts_df['Variable'], self.counts_df['Unconvolved Counts'], self.counts_df['Convolved Counts']):
                derivative_convolved = derivative_convolved.subs(variable, convolved_counts)
                derivative_unconvolved = derivative_unconvolved.subs(variable, unconvolved_counts)

                expression_convolved = expression_convolved.subs(variable, convolved_counts)
                expression_unconvolved = expression_unconvolved.subs(variable, unconvolved_counts)

            # Add the value squared times the derivative squared
            squared_total_convolved_unc += ((convolved_individual_unc ** 2) * (derivative_convolved ** 2))
            squared_total_unconvolved_unc += ((unconvolved_individual_unc ** 2) * (derivative_unconvolved ** 2))

        total_convolved_uncertainty = sqrt(squared_total_convolved_unc)
        total_unconvolved_uncertainty = sqrt(squared_total_unconvolved_unc)

        convolved_uncertainties.append(ufloat(expression_convolved, total_convolved_uncertainty))
        unconvolved_uncertainties.append(ufloat(expression_unconvolved, total_unconvolved_uncertainty))

    self.counts_df['Convolved Percentage'] = convolved_uncertainties
    self.counts_df['Unconvolved Percentage'] = unconvolved_uncertainties



  def update_table_by_activity(self, read_table = True):
    """
    This function reads the user-input activity percentages for the isotopes present in the current sample
    and determines from those the mass percentages of the sample by isotope, then updates the isotopic
    information table in DESSIMATE and scales the simulated spectrum accordingly.

    Returns
    -------
    None.

    """

    if sum(self.total_raw_spectrum) == 0:
        raise Exception('There is no spectrum to analyze. Please input a file before attempting this function.')
        return 0

    # Current known things are isotopic nuclear data, maximum counts possible for each isotope,
    # and activities of each isotope. Read these in from the table now.
```

```
    if read_table: self.isotopic_df['Activity %'] = [float(self.isotopic_composition_table.item(row_num, 1).text().replace('%', '')) for row_num in range(len(self.isotopic_df['Isotope']))]

    half_lives = []
    atomic_masses = []
    for isotope_name in self.isotopic_df['Isotope']:

        half_life = self.half_life_dict[isotope_name]
        atomic_mass = self.mass_dict[isotope_name]

        half_lives.append(half_life)
        atomic_masses.append(atomic_mass)

    self.isotopic_df['Activity (Bq)'], self.isotopic_df['Time Taken for Maximum Counts (in seconds)'], self.length_of_measurement, self.isotopic_df['Measured Counts'], \
        self.isotopic_df['Total Atoms'], self.isotopic_df['Mass (pg)'], total_mass, self.isotopic_df['Mass %'] = get_mass_from_activity(self.isotopic_df['Activity %'], \
        float(self.total_activity_number.text()), self.isotopic_df['Maximum Counts'], half_lives, atomic_masses)

    # Set the length of measurement to the nearest lower integer, then update the text box displaying this
    self.length_of_measurement_field.setText(str(int(self.length_of_measurement.nominal_value)))

    # Fill out table with newly calculated values
    column_names = ['Isotope', 'Activity %', 'Activity (Bq)', 'Mass %', 'Mass (pg)']
    self.isotopic_composition_table.setHorizontalHeaderLabels(column_names)

    for column_num, column_name in enumerate(column_names):
        for row_num, cell_value in enumerate(self.isotopic_df[column_name]):

            table_item = QtWidgets.QTableWidgetItem()
            if type(cell_value) == str:
                table_item.setText(cell_value)
            else:
                if column_name == 'Mass %' or column_name == 'Activity %':
                    table_item.setText(f'{100*cell_value:.3f}%')
                else:
                    table_item.setText(f'{cell_value:.3f}')
            self.isotopic_composition_table.setItem(row_num, column_num, table_item)

    self.total_mass_number.setText(f'{total_mass:.3f}')

    # Update spectrum with new isotopic information
    self.update_graph()



def update_table_by_mass(self):
    """
    This function reads the user-input mass percentages for the isotopes present in the current sample
    and determines from those the activity percentages of the sample by isotope, then update the isotopic
    information table in DESSIMATE and scales the simulated spectrum accordingly.

    Returns
    -------
    None.
```

```
    """

    if sum(self.total_raw_spectrum) == 0:
        raise Exception('There is no spectrum to analyze. Please input a file before attempting this function.')
        return 0

    # Current known things are isotopic nuclear data, maximum counts possible for each isotope,
    # and masses of each isotope. Read thes in from the table now.

    self.isotopic_df['Mass %'] = [float(self.isotopic_composition_table.item(row_num, 3).text().replace('%', ''))
for row_num in range(len(self.isotopic_df['Isotope']))]

    half_lives = []
    atomic_masses = []
    for isotope_name in self.isotopic_df['Isotope']:

        half_life = self.half_life_dict[isotope_name]
        atomic_mass = self.mass_dict[isotope_name]

        half_lives.append(half_life)
        atomic_masses.append(atomic_mass)

    self.isotopic_df['Mass (pg)'], self.isotopic_df['Total Atoms'], self.isotopic_df['Activity (Bq)'], total_activity,
self.isotopic_df['Activity %'], self.isotopic_df['Time Taken for Maximum Counts (in seconds)'], \
    self.length_of_measurement, self.isotopic_df['Measured Counts'] =
get_activity_from_mass(self.isotopic_df['Mass %'], float(self.total_mass_number.text()),
self.isotopic_df['Maximum Counts'], half_lives, atomic_masses)

    # Set the length of measurement to the nearest lower integer, then update the text box displaying this
    self.length_of_measurement_field.setText(str(int(self.length_of_measurement.nominal_value)))

    # Fill out table with newly calculated values
    column_names = ['Isotope', 'Activity %', 'Activity (Bq)', 'Mass %', 'Mass (pg)']
    self.isotopic_composition_table.setHorizontalHeaderLabels(column_names)

    for column_num, column_name in enumerate(column_names):
        for row_num, cell_value in enumerate(self.isotopic_df[column_name]):

            table_item = QtWidgets.QTableWidgetItem()
            if type(cell_value) == str:
                table_item.setText(cell_value)
            else:
                if column_name == 'Mass %' or column_name == 'Activity %':
                    table_item.setText(f'{100*cell_value:.3f}%')
                else:
                    table_item.setText(f'{cell_value:.3f}')
            self.isotopic_composition_table.setItem(row_num, column_num, table_item)

    self.total_activity_number.setText(f'{total_activity:.3f}')

    # Update spectrum with new isotopic information
    self.update_graph()
```

```
    def uranium_analysis(self):

        self.default_uranium_rois()
        self.determine_uranium_counts()
        self.perform_correction_factor()
        self.determine_mass_ratios()
        self.switch_tabs(1)
        #self.save_results()
        self.update_graph()



    def default_uranium_rois(self):

        if sum(self.total_raw_spectrum) == 0:
            raise Exception('There is no spectrum to analyze. Please input a file before attempting to analyze something.')
            return 0

        self.uranium_mode = True

        measured_counts_dict = {'Isotope': ['U234', 'U235', 'U236', 'U238', 'U235inf'], 'ROI Bounds': [[4.8, 4.9], [4.285, 4.7], [4.56, 4.585], [4.255, 4.285], [4.46, 4.55]],\
                                'Counts in ROI': [0, 0, 0, 0, 0], 'Total Counts': [0, 0, 0, 0, 0], 'Mass Percent': [0, 0, 0, 0, 0]}

        self.measured_counts_df = pd.DataFrame(data = measured_counts_dict, dtype = object)



    def determine_uranium_counts(self):

        for row_num in range(len(self.measured_counts_df['Isotope'])):

            x, counts = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[row_num, 'ROI Bounds'], self.measured_counts_df.loc[row_num, 'ROI Bounds'])
            x, test_cts = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], [4, 4.2], [4, 4.2])
            #print(test_cts)

            #for row_num_2 in range(len(self.measured_counts_df['Isotope'][:-1])):
                #print(f'Number of {self.measured_counts_df.loc[row_num_2, "Isotope"]} counts in {self.measured_counts_df.loc[row_num, "Isotope"]} ROI: {counts[row_num_2]}.')
            #print()

            self.measured_counts_df.loc[row_num, 'Counts in ROI'] = ufloat(sum(counts), np.sqrt(sum(counts)))



    def perform_correction_factor(self):

        if not self.isotopic_df['Isotope'].str.contains('U235').any():
            print('A correction factor cannot be performed without a U235 spectrum present. Please input a U235 file before attempting to perform a correction factor.')
```

```
        self.measured_counts_df['Total Counts'] = self.measured_counts_df['Counts in ROI']
        return 0

    self.measured_counts_df.loc[0, 'Total Counts'] = self.measured_counts_df.loc[0, 'Counts in ROI']
    self.measured_counts_df.loc[3, 'Total Counts'] = self.measured_counts_df.loc[3, 'Counts in ROI']

    # We need to determine three things:
      # The number of simulated counts of U235 in the U236 region
      # The number of simulated counts of U235 in the inf. region
      # The number of "experimental" counts of U235 in the inf. region

    crm_num = '970'
    cap_amt = '00'
    spec_234 = f'capping_dependence/simulations_to_analyze_with/{cap_amt}_um_CRMU{crm_num}_U234_h1_0.out'
    spec_235 = f'capping_dependence/simulations_to_analyze_with/{cap_amt}_um_CRMU{crm_num}_U235_h1_0.out'
    spec_236 = f'capping_dependence/simulations_to_analyze_with/{cap_amt}_um_CRMU{crm_num}_U236_h1_0.out'
    spec_238 = f'capping_dependence/simulations_to_analyze_with/{cap_amt}_um_CRMU{crm_num}_U238_h1_0.out'
    sim_isotopic_df, sim_energies = upload_sim_spectrum(self.detector_response_size, spec_234, spec_235, spec_236, spec_238)

    x, counts_in_inf = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[4, 'ROI Bounds'], self.measured_counts_df.loc[4, 'ROI Bounds'])
    x, counts_in_236 = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[2, 'ROI Bounds'], self.measured_counts_df.loc[2, 'ROI Bounds'])
    x, sim_counts_in_inf = get_counts_in_regions(sim_energies, sim_isotopic_df['Unconvolved Spectrum'], sim_isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[4, 'ROI Bounds'], self.measured_counts_df.loc[4, 'ROI Bounds'])
    x, sim_counts_in_236 = get_counts_in_regions(sim_energies, sim_isotopic_df['Unconvolved Spectrum'], sim_isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[2, 'ROI Bounds'], self.measured_counts_df.loc[2, 'ROI Bounds'])
    for num in range(5):
      x, counts_total = get_counts_in_regions(self.energies, self.isotopic_df['Unconvolved Spectrum'], self.isotopic_df['Convolved Spectrum'], self.measured_counts_df.loc[num, 'ROI Bounds'], self.measured_counts_df.loc[num, 'ROI Bounds'])
      print(counts_total)
    for row_num in range(len(self.isotopic_df['Isotope'])):
      if self.isotopic_df.loc[row_num, 'Isotope'] == 'U235':
        num_sim_235_cts_in_inf = ufloat(sim_counts_in_inf[row_num], np.sqrt(sim_counts_in_inf[row_num]))
        num_sim_235_cts_in_236 = ufloat(sim_counts_in_236[row_num], np.sqrt(sim_counts_in_236[row_num]))

    # The ratio of the simulated counts of U235 in the regions is equal to the ratio of the "experimental" counts of U235 in the regions

    num_exp_235_cts_in_236 = num_sim_235_cts_in_236 * ufloat(sum(counts_in_inf), np.sqrt(sum(counts_in_inf))) / num_sim_235_cts_in_inf

    self.measured_counts_df.loc[2, 'Total Counts'] = self.measured_counts_df.loc[2, 'Counts in ROI'] - num_exp_235_cts_in_236
```

```
    self.measured_counts_df.loc[1, 'Total Counts'] = self.measured_counts_df.loc[1, 'Counts in ROI'] - self.measured_counts_df.loc[2, 'Total Counts']



  def determine_mass_ratios(self):

    # The required steps for this function are:
      # Hijack the simulated activity percentages and replace them with what was "experimentally" determined
      # Call the function to convert these percentages to mass percentages
      # Convert these mass percentages to mass ratios

    half_lives = []
    atomic_masses = []
    for isotope_name in self.measured_counts_df['Isotope']:
      if isotope_name == 'U235inf': continue
      half_life = self.half_life_dict[isotope_name]
      atomic_mass = self.mass_dict[isotope_name]

      half_lives.append(half_life)
      atomic_masses.append(atomic_mass)

    for row_num in range(len(self.isotopic_df['Isotope'])):
      for row_num2 in range(len(self.measured_counts_df['Isotope'])):
        if self.isotopic_df.loc[row_num, 'Isotope'] == self.measured_counts_df.loc[row_num2, 'Isotope']:
          self.isotopic_df.loc[row_num, 'Activity %'] = self.measured_counts_df.loc[row_num2, 'Total Counts']

    self.update_table_by_activity(read_table = False)

    for row_num in range(len(self.isotopic_df['Isotope'])):
      for row_num2 in range(len(self.measured_counts_df['Isotope'])):
        if self.isotopic_df.loc[row_num, 'Isotope'] == self.measured_counts_df.loc[row_num2, 'Isotope']:
          self.measured_counts_df.loc[row_num2, 'Mass Percent'] = self.isotopic_df.loc[row_num, 'Mass %']
          #raw_text = self.isotopic_composition_table.item(row_num, 3).text().replace('%', '')
          #self.measured_counts_df.loc[row_num2, 'Mass Percent'] = ufloat_fromstr(raw_text)

    denominator = ['U234', 0]

    print(f'The chosen denominator is {denominator[0]}.')
    for row_num in range(len(self.measured_counts_df['Isotope'][:-1])):
      print(f'The mass ratio {self.measured_counts_df.loc[row_num, "Isotope"]}/{self.measured_counts_df.loc[denominator[1], "Isotope"]} ' + \
        f'is equal to {self.measured_counts_df.loc[row_num, "Mass Percent"]/self.measured_counts_df.loc[denominator[1], "Mass Percent"]}')

    #self.isotopic_df['Mass %'] = [float(self.isotopic_composition_table.item(row_num, 3).text().replace('%', '')) for row_num in range(len(self.isotopic_df['Isotope']))]



  def save_results(self):
    denominator = ['U234', 0]
```

```
        df = pd.read_csv('capping_dependence.csv', dtype = object)
        df_row_num = 0

        df.loc[df_row_num, 'U234 Mass'] = self.measured_counts_df.loc[0, 'Mass Percent']
        df.loc[df_row_num, 'U235 Mass'] = self.measured_counts_df.loc[1, 'Mass Percent']
        df.loc[df_row_num, 'U236 Mass'] = self.measured_counts_df.loc[2, 'Mass Percent']
        df.loc[df_row_num, 'U238 Mass'] = self.measured_counts_df.loc[3, 'Mass Percent']
        df.loc[df_row_num, 'U234 Ratio'] = df.loc[df_row_num, 'U234 Mass'] / df.loc[df_row_num,
f'{denominator[0]} Mass']
        df.loc[df_row_num, 'U235 Ratio'] = df.loc[df_row_num, 'U235 Mass'] / df.loc[df_row_num,
f'{denominator[0]} Mass']
        df.loc[df_row_num, 'U236 Ratio'] = df.loc[df_row_num, 'U236 Mass'] / df.loc[df_row_num,
f'{denominator[0]} Mass']
        df.loc[df_row_num, 'U238 Ratio'] = df.loc[df_row_num, 'U238 Mass'] / df.loc[df_row_num,
f'{denominator[0]} Mass']

        df.to_csv('capping_dependence.csv', index = False)



    def savefig_graph(self):
        self.graph.axes.get_legend().set_visible(False)
        self.graph.canvas.figure.savefig('output.png')
        self.graph.axes.get_legend().set_visible(True)




app = QtWidgets.QApplication(sys.argv)
window = Ui()
app.exec_()
```

B.2. util.py

```
import pandas as pd
import numpy as np
import math
from uncertainties import ufloat



def read_two_col_text_file(text_lines, return_components = False):
    energies = []
    counts = []
    plottable = []

    for line in text_lines:
        energy = line.split('\t')[0]
        count = line.split('\t')[1]

        energies.append(float(energy))
        counts.append(float(count))

    energies = np.array(energies)
    counts = np.array(counts)
```

```
    plottable = np.array([energies, counts])

    if return_components: return plottable, energies, counts
    else: return plottable



def create_gaussian_output(amplitude, center, sigma, inputs):
    gaussianOutput = amplitude*np.exp((-1*(inputs-center)**2)/(2*(sigma**2)))
    return gaussianOutput



def create_isotopic_data_frame():
    preliminary_dict = {'Isotope': [],
                        'Maximum Counts': [],
                        'Time Taken for Maximum Counts (in seconds)': [],
                        'Measured Counts': [],
                        'Total Atoms': [],
                        'Activity %': [],
                        'Activity (Bq)': [],
                        'Mass %': [],
                        'Mass (pg)': [],
                        'Percentage Counts Measured': [],
                        'Raw Spectrum': [],
                        'Unconvolved Spectrum': [],
                        'Convolved Spectrum': [],
                        'Plotting Color': [],
                        }

    isotopic_df = pd.DataFrame(data = preliminary_dict, dtype = object)

    return isotopic_df



def get_counts_in_region(lower_bound, upper_bound, spectrum, energies):
    energies_yint = energies[0]
    energies_slope = energies[1] - energies[0]

    energy_lower_idx = max(math.floor((lower_bound - energies_yint) / energies_slope), 0)
    energy_upper_idx = min(math.ceil((upper_bound - energies_yint) / energies_slope), len(spectrum)-1)

    num_counts_in_region = sum(spectrum[energy_lower_idx : energy_upper_idx+1])
    return num_counts_in_region



def get_counts_in_regions(energies, unconvolved_spectra, convolved_spectra,
energy_region_unconvolved_bounds, energy_region_convolved_bounds):
    """

    Parameters
    ----------
    energies : array
```

```
    A list of the energies at which the specta are relevant.
  unconvolved_spectra : dataframe row of arrays
    A column of a data frame containing several unconvolved spectra at the energies in energies.
  convolved_spectra : dataframe row of arrays
    A column of a data frame containing several convolved spectra at the energies in energies.
  energy_region_unconvolved_min : float
    The lower energy of the unconvolved energy region in question.
  energy_region_unconvolved_max : float
    The upper energy of the unconvolved energy region in question.
  energy_region_convolved_min : float
    The lower energy of the convolved energy region in question.
  energy_region_convolved_max : float
    The upper energy of the convolved energy region in question.

  Returns
  -------
  The counts in the convolved and unconvolved energy regions by isotope.

  """

  # Find yint and slope of energies array
  energies_yint = energies[0]
  energies_slope = energies[1] - energies[0]

  # Define bounds explicitly
  energy_region_unconvolved_min = energy_region_unconvolved_bounds[0]
  energy_region_unconvolved_max = energy_region_unconvolved_bounds[1]
  energy_region_convolved_min = energy_region_convolved_bounds[0]
  energy_region_convolved_max = energy_region_convolved_bounds[1]

  # Find location in energies array that correspond to desired energy limits
  energy_unconvolved_lower_idx = math.floor((energy_region_unconvolved_min - energies_yint) /
energies_slope)
  energy_unconvolved_upper_idx = math.ceil((energy_region_unconvolved_max - energies_yint) /
energies_slope)
  energy_convolved_lower_idx = math.floor((energy_region_convolved_min - energies_yint) / energies_slope)
  energy_convolved_upper_idx = math.ceil((energy_region_convolved_max - energies_yint) / energies_slope)

  # Integrate output spectra from Geant in these regions to get counts in these regions by isotope
  num_counts_unconvolved = np.zeros_like(unconvolved_spectra)
  for spectrum_idx, spectrum in enumerate(unconvolved_spectra):
    num_counts_unconvolved[spectrum_idx]                                                    =
sum(spectrum[energy_unconvolved_lower_idx:energy_unconvolved_upper_idx+1])

  num_counts_convolved = np.zeros_like(convolved_spectra)
  for spectrum_idx, spectrum in enumerate(convolved_spectra):
    num_counts_convolved[spectrum_idx]                                                    =
sum(spectrum[energy_convolved_lower_idx:energy_convolved_upper_idx+1])

  return num_counts_unconvolved, num_counts_convolved



def get_mass_from_activity(activity_percents, total_activity, maximum_counts, half_lives, atomic_masses):
```

```
    # Normalize the activity percentages to sum to 100
    activity_percents /= sum(activity_percents)

    # Determine the activity of each isotope in the sample as percentage of total activity times total activity
    abs_activity = activity_percents * total_activity

    # Calculate the total amount of time required for each isotope to reach the maximum number of counts and
set the length of measurement to the shortest of these
    time_for_max_counts = maximum_counts / abs_activity
    actual_time = min(time_for_max_counts)

    # Determine the number of counts that will occur during this measurement length for each isotope
    measured_counts = actual_time * abs_activity

    # Determine the total number of atoms of each isotope present in the source
    total_atoms = abs_activity * half_lives / np.log(2)

    # Find the total mass of each isotope in the source by value and percentage of the full source
    abs_mass = total_atoms * atomic_masses

    total_mass = sum(abs_mass)

    mass_percents = abs_mass / total_mass

    return abs_activity, time_for_max_counts, actual_time, measured_counts, total_atoms, abs_mass, total_mass,
mass_percents



def get_activity_from_mass(mass_percents, total_mass, maximum_counts, half_lives, atomic_masses):

    # Normalize the mass percentages to sum to 100
    mass_percents /= sum(mass_percents)

    # Determine the mass of each isotope in the sample as percentage of total mass times total mass
    abs_mass = mass_percents * total_mass

    # Calculate the total number of atoms of each isotope present in the source
    total_atoms = abs_mass / atomic_masses

    # Determine the activity of each isotope in the sample from the number of atoms and the half life of each
isotope
    abs_activity = total_atoms * np.log(2) / half_lives

    # Find the total activity of the source and the percentage of the activity from each isotope
    total_activity = sum(abs_activity)
    activity_percents = abs_activity / total_activity

    # Calculate the total amount of time required for each isotope to reach the maximum number of counts and
set the length of measurement to the shortest of these
    time_for_max_counts = maximum_counts / abs_activity
    actual_time = min(time_for_max_counts)

    # Determine the number of counts that will occur during this measurement length for each isotope
    measured_counts = actual_time.nominal_value * abs_activity.apply(lambda x: x.nominal_value)
```

```
    return abs_mass, total_atoms, abs_activity, total_activity, activity_percents, time_for_max_counts, actual_time, measured_counts



def get_isotopic_information_tables(decay_modes = ['A']):

    with open('nuclides.txt', 'r') as f:
        lines = f.readlines()

    half_life_dict = {}
    mass_dict = {}
    u = 1.66054 * (10 ** (-27))
    eV_mass = 1.78266192 * (10 ** (-36))

    for idx, line in enumerate(lines[1:]):
        a_val = int(line[:4].strip())
        element_name = line[4:12].strip()
        z_val = int(line[12:16].strip())
        n_val = int(line[16:20].strip())
        energy = line[20:29].strip()
        spin_val = line[29:44].strip()
        mass_excess = float(line[44:65].strip())
        mass_excess_unc = float(line[65:76].strip())
        half_life_text = line[76:102].strip()
        half_life = float(line[102:128].strip())
        abundance = line[128:139].strip()
        abundance_unc = line[139:148].strip()
        decay_mode = line[148:157].strip()
        branching_ratio = line[157:].strip()

        isotope_name = element_name + str(a_val)
        mass_excess_with_unc = ufloat(mass_excess, mass_excess_unc)
        atomic_mass = a_val * u + (mass_excess_with_unc * 1000000 * eV_mass)

        if decay_mode in decay_modes:
            half_life_dict[isotope_name] = half_life
            mass_dict[isotope_name] = atomic_mass

    return half_life_dict, mass_dict

def upload_sim_spectrum(detector_response_size, spec_234, spec_235, spec_236, spec_238):
    isotopic_df = create_isotopic_data_frame()
    energies = []
    for spectrum_number, event in zip([0, 1, 2, 3], [spec_234, spec_235, spec_236, spec_238]):
        with open(event, 'r') as file:
            file_text = file.readlines()

        # Parse the text by line, looking for keywords describing the spectrum
        for line_num, line in enumerate(file_text):
            if line[0] == '#':
                # Found a keyword
                var_name = line.split('\t')[0].split('#')[1]
                var_value = line.split('\t')[1].split('\n')[0]
```

```
        if var_name.split(' ')[0] == 'E_l':
            # Read in spectrum file
            spectrum = read_two_col_text_file(file_text[line_num+1:])
            spectrum = [spectrum[0][1:], spectrum[1][1:]]

            if len(energies) != 0:

                # Check to make sure new energy array is aligned with old energy array
                new_intercept, old_intercept, energy_step_size = check_if_valid_binning(energies, spectrum)

                # Pad end of spectrum with zeroes
                highest_old_energy = spectrum[0][-1]
                padding_index = np.arange(1,75)
                new_energies = spectrum[0]
                new_energies_padded = np.append(new_energies, highest_old_energy+(padding_index*energy_step_size))

                # Create new energies array for all of the spectra together
                spectrum, energies = pad_spectra(energy_step_size, isotopic_df, spectrum, new_intercept, old_intercept, new_energies_padded[-1], energies[-1])

            else:

                energy_step_size = spectrum[0][1] - spectrum[0][0]
                padding_index = np.arange(1, 75)
                highest_old_energy = spectrum[0][-1]
                energies = np.append(spectrum[0], highest_old_energy + (padding_index * energy_step_size))

        elif var_name == 'partle':
            isotope_name = var_value

        elif var_name == 'n_done':
            num_max_counts = int(float(var_value))

    # Create new row for data frame and append to the full isotopic data frame
    new_row = pd.DataFrame(data = {'Isotope': isotope_name, 'Maximum Counts': num_max_counts, 'Time Taken for Maximum Counts (in seconds)': 0., 'Measured Counts': num_max_counts, 'Total Atoms': 0., 'Activity %': 0., 'Activity (Bq)': 0., \
                    'Mass (pg)': 0., 'Mass %': 0., 'Percentage Counts Measured': 1, 'Raw Spectrum': [np.append(spectrum[1], np.zeros(74))], 'Unconvolved Spectrum': [[]], 'Convolved Spectrum': [[]], \
                    'Plotting Color': 'orange'}, index = [len(isotopic_df['Isotope'])], dtype = object)
    isotopic_df = pd.concat([isotopic_df, new_row])

  isotopic_df['Unconvolved Spectrum'] = isotopic_df['Raw Spectrum']
  total_raw_spectrum = sum(isotopic_df['Unconvolved Spectrum'])
  gaussian_inputs = np.arange(0, (len(total_raw_spectrum)-1)*energy_step_size, energy_step_size) - ((len(total_raw_spectrum)-1)/2)*energy_step_size

  # Define parameters for outputs and create outputs for the Gaussian response curve
  sigma = detector_response_size/2.355
  amplitude = 1./(sigma*np.sqrt(2*np.pi))
  center = 0
  gaussian_func = create_gaussian_output(amplitude, center, sigma, gaussian_inputs) * energy_step_size

  # Convolve and plot convolved spectrum
```

```
    convolved_spectra = []
    for spectrum_number in range(len(isotopic_df['Isotope'])):
        convolved_spectrum = np.convolve(isotopic_df.loc[spectrum_number, 'Unconvolved Spectrum'], gaussian_func, mode = 'same')
        convolved_spectra.append(convolved_spectrum)
    isotopic_df['Convolved Spectrum'] = convolved_spectra

    convolved_spectrum = sum(isotopic_df['Convolved Spectrum'])

    return isotopic_df, energies



def check_if_valid_binning(energies, spectrum):

    new_slope = round(spectrum[0][1] - spectrum[0][0], 10)
    old_slope = round(energies[1] - energies[0], 10)

    if new_slope != old_slope:
        raise Exception(f'The bin width of this spectrum is not the same as that of the other spectra. The bin width of this spectrum was {new_slope} and the bin width of the other spectra is {old_slope}.')
        return 0

    energy_step_size = new_slope

    new_intercept = spectrum[0][0]
    old_intercept = energies[0]

    slope_differences = round(abs(new_intercept - old_intercept), 10) % energy_step_size # Determines if the intercepts are separated by an integer multiple of the slope

    if slope_differences > 0.0000001 and abs(slope_differences - energy_step_size) > 0.0000001:
        # The energy bins are not compatible; please input a new spectrum instead
        raise Exception('The bins of this spectrum are offset from the other spectra by a non-integer multiple of the bin width and are therefore incompatible. This spectrum was not added to the plotted total. Please try a different spectrum.')
        return 0

    return new_intercept, old_intercept, energy_step_size

def pad_spectra(energy_step_size, isotopic_df, spectrum, new_intercept, old_intercept, highest_new_energy, highest_old_energy):

    low_energy = min(old_intercept, new_intercept)
    high_energy = max(highest_old_energy, highest_new_energy)

    if old_intercept > new_intercept:
        num_paddings = (old_intercept - new_intercept) / energy_step_size
        padding_index = np.zeros(round(num_paddings))
        for row_num in range(len(isotopic_df['Isotope'])):
            isotopic_df.loc[row_num, 'Raw Spectrum'] = np.append(padding_index, isotopic_df.loc[row_num, 'Raw Spectrum'])

    if new_intercept > old_intercept:
        num_paddings = (new_intercept - old_intercept) / energy_step_size
```

```
        padding_index = np.zeros(round(num_paddings))
        spectrum[1] = np.append(padding_index, spectrum[1])

    if round(highest_new_energy, 10) > round(highest_old_energy, 10):
        num_paddings = (highest_new_energy - highest_old_energy) / energy_step_size
        padding_index = np.zeros(round(num_paddings))
        for row_num in range(len(isotopic_df['Isotope'])):
            appended = np.append(isotopic_df.loc[row_num, 'Raw Spectrum'], padding_index)
            isotopic_df.loc[[row_num], 'Raw Spectrum'] = pd.Series([appended], index = [row_num])

    if round(highest_old_energy, 10) > round(highest_new_energy, 10):
        num_paddings = (highest_old_energy - highest_new_energy) / energy_step_size
        padding_index = np.zeros(round(num_paddings))
        spectrum[1] = np.append(spectrum[1], padding_index)

    #print(low_energy)
    #print(high_energy)
    #print(self.energy_step_size)

    energies = np.arange(low_energy, round(round(high_energy, 10) + round(energy_step_size, 10), 10),
energy_step_size)

    return spectrum, energies
```

Appendix C

ADDITIONAL SPECTRAL COUNT BREAKDOWNS

C.1. 20% Enriched, 0 µm Cap

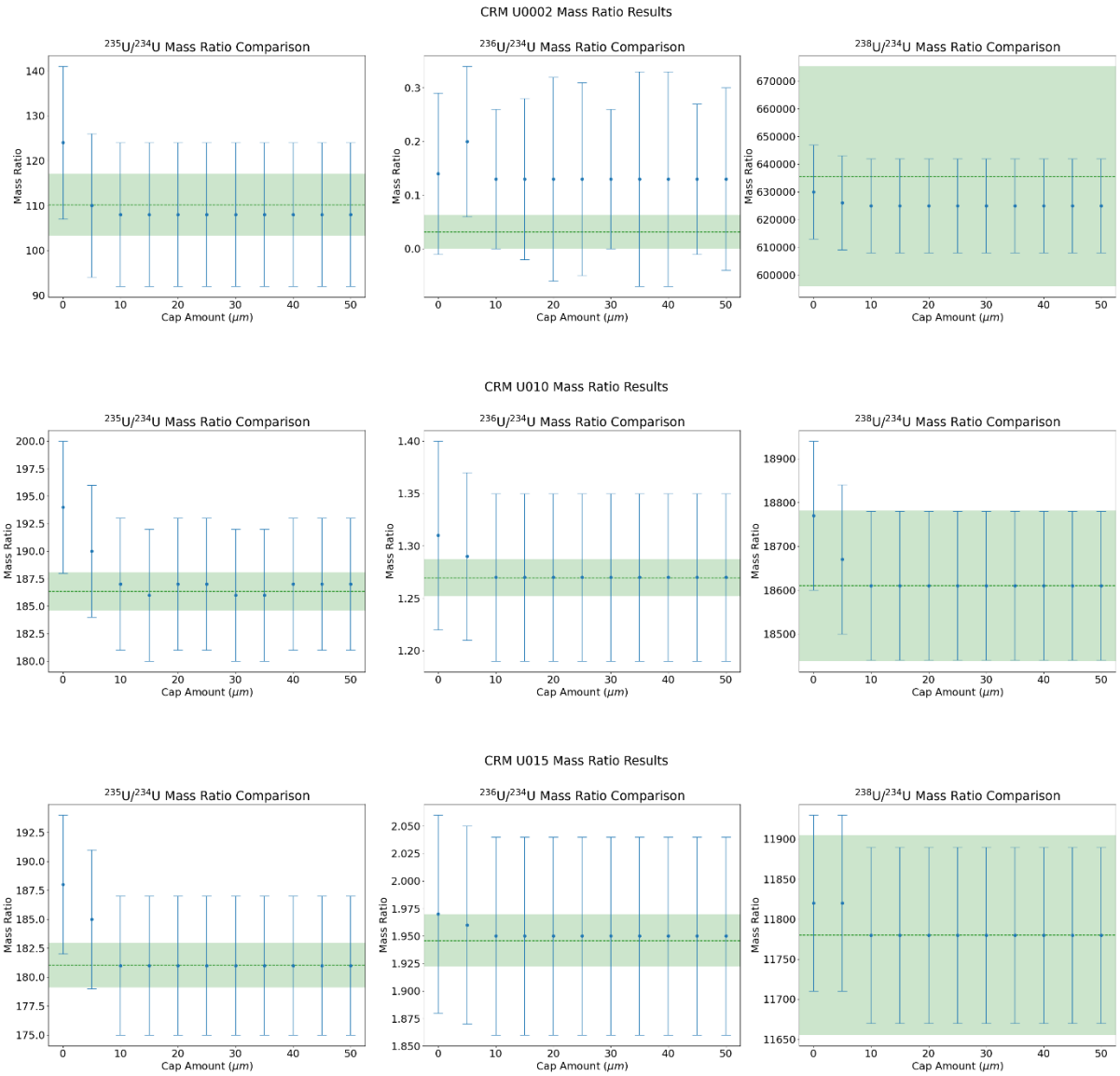
	Isotopic Makeup of Region					Percent of Isotope's Counts in Region			
ROI	**Total**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**
^{234}U	8628787	8628787 (100%)	0 (0%)	0 (0%)	0 (0%)	95.68%	0%	0%	0%
^{235}U	659806	19880 (3.01%)	485736 (73.62%)	154190 (23.37%)	0 (0%)	0.22%	96.23%	96.14%	0%
^{236}U	170460	1362 (0.8%)	15398 (9.03%)	153700 (90.17%)	0 (0%)	0.02%	3.05%	95.83%	0%
^{238}U	306734	1813 (0.59%)	111 (0.04%)	25 (0.01%)	304785 (99.36%)	0.02%	0.02%	0.02%	96.28%

C.2. 1.5% Enriched, 0 µm Cap

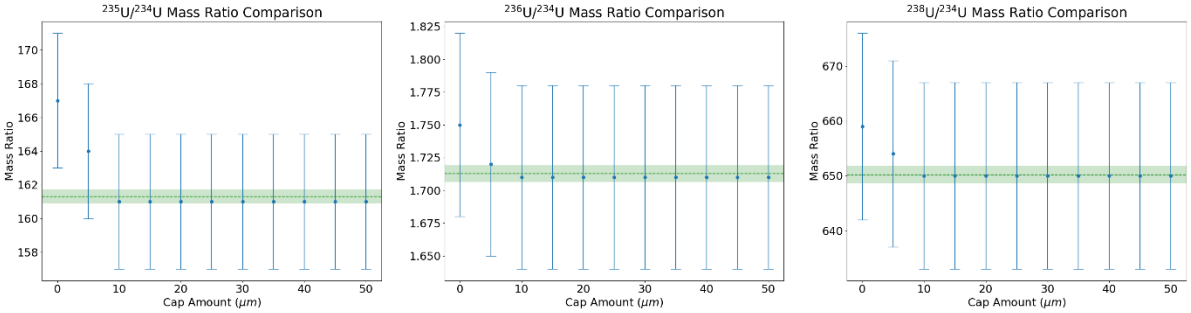
	Isotopic Makeup of Region					Percent of Isotope's Counts in Region			
ROI	**Total**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**	**^{234}U**	**^{235}U**	**^{236}U**	**^{238}U**
^{234}U	5565298	5565298 (100%)	0 (0%)	0 (0%)	0 (0%)	95.69%	0%	0%	0%
^{235}U	477781	12777 (2.67%)	351778 (73.63%)	113226 (23.70%)	0 (0%)	0.22%	96.23%	96.07%	0%
^{236}U	124931	861 (0.69%)	11191 (8.96%)	112879 (90.35%)	0 (0%)	0.01%	3.06%	95.77%	0%
^{238}U	3562731	1186 (0.03%)	82 (0%)	26 (0%)	3561437 (99.96%)	0.02%	0.02%	0.02%	96.24%

Appendix D

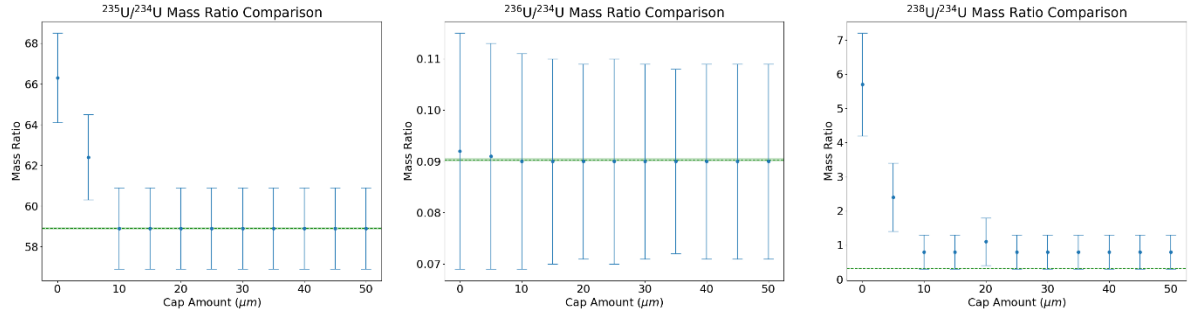
SELF-CONSISTENCY TEST RESULTS



CRM U200 Mass Ratio Results



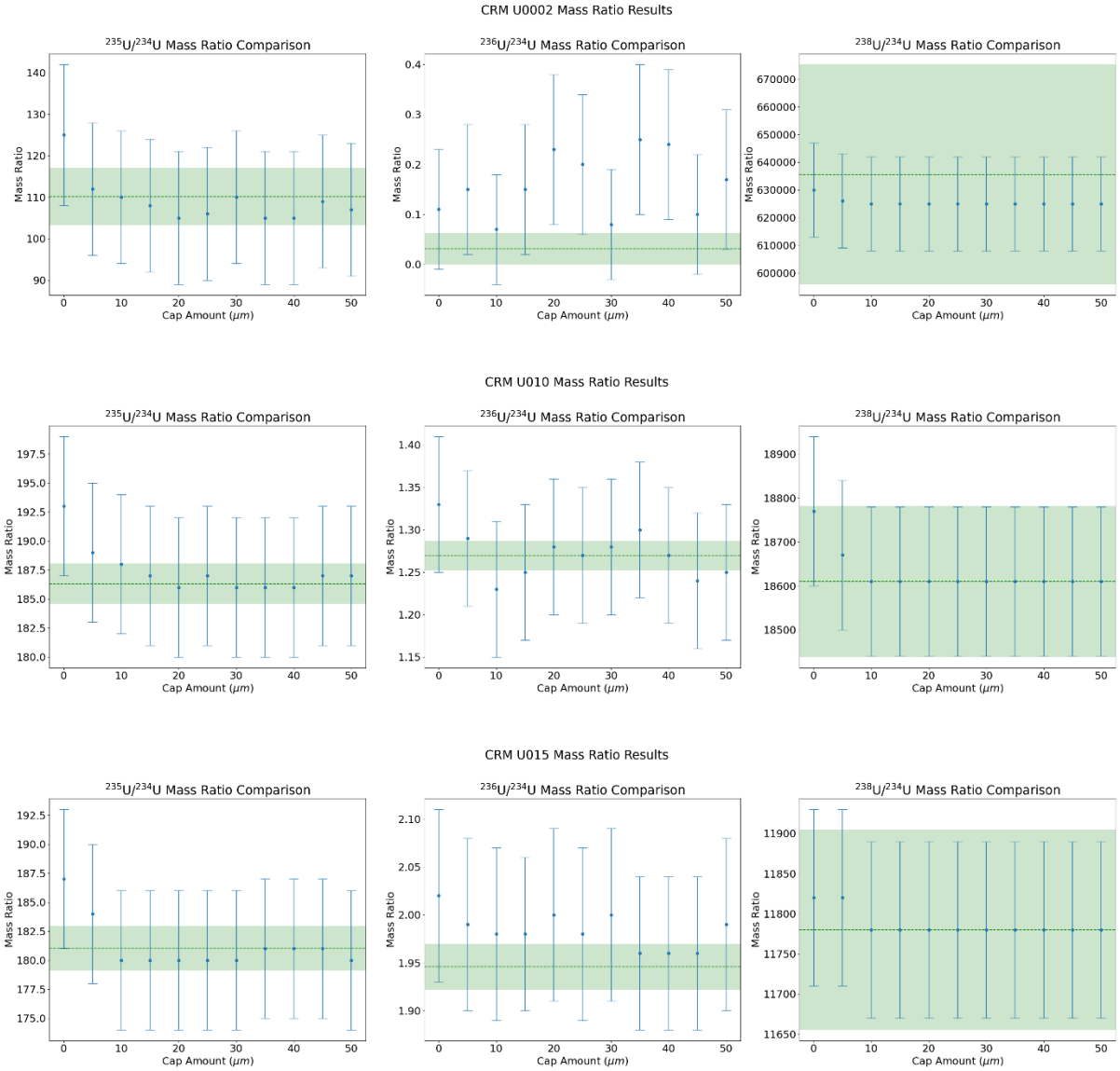
CRM U970 Mass Ratio Results

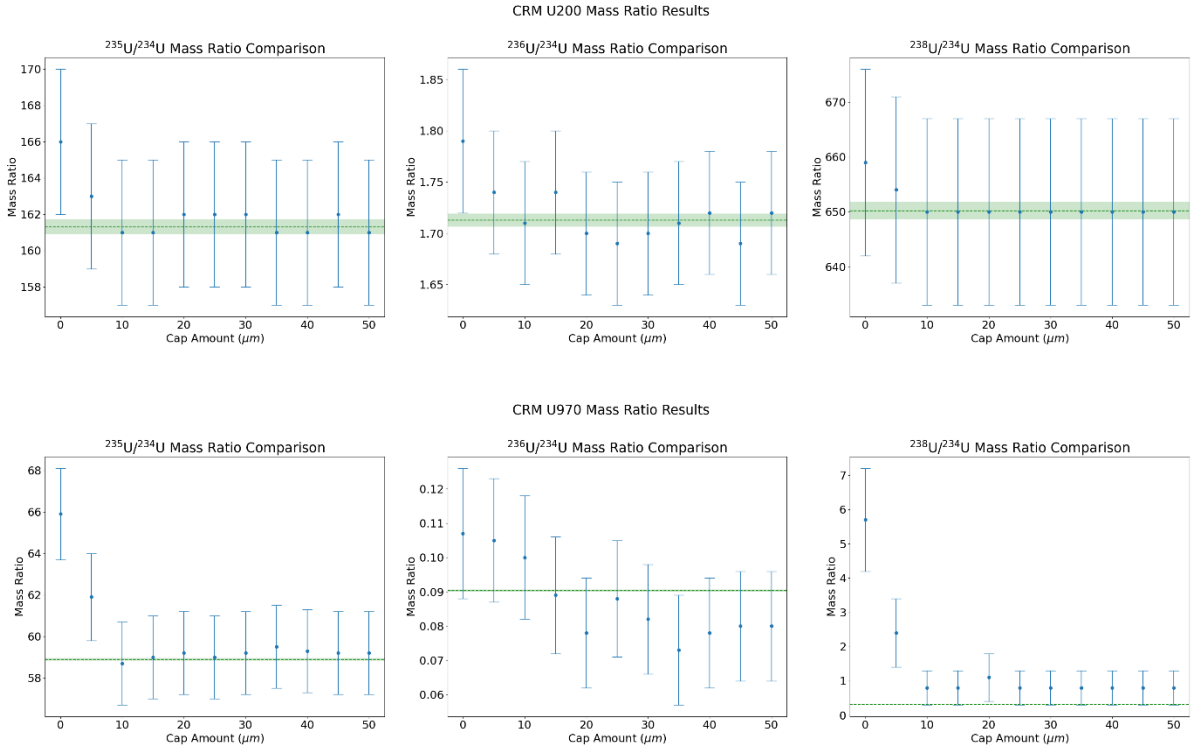


Appendix E

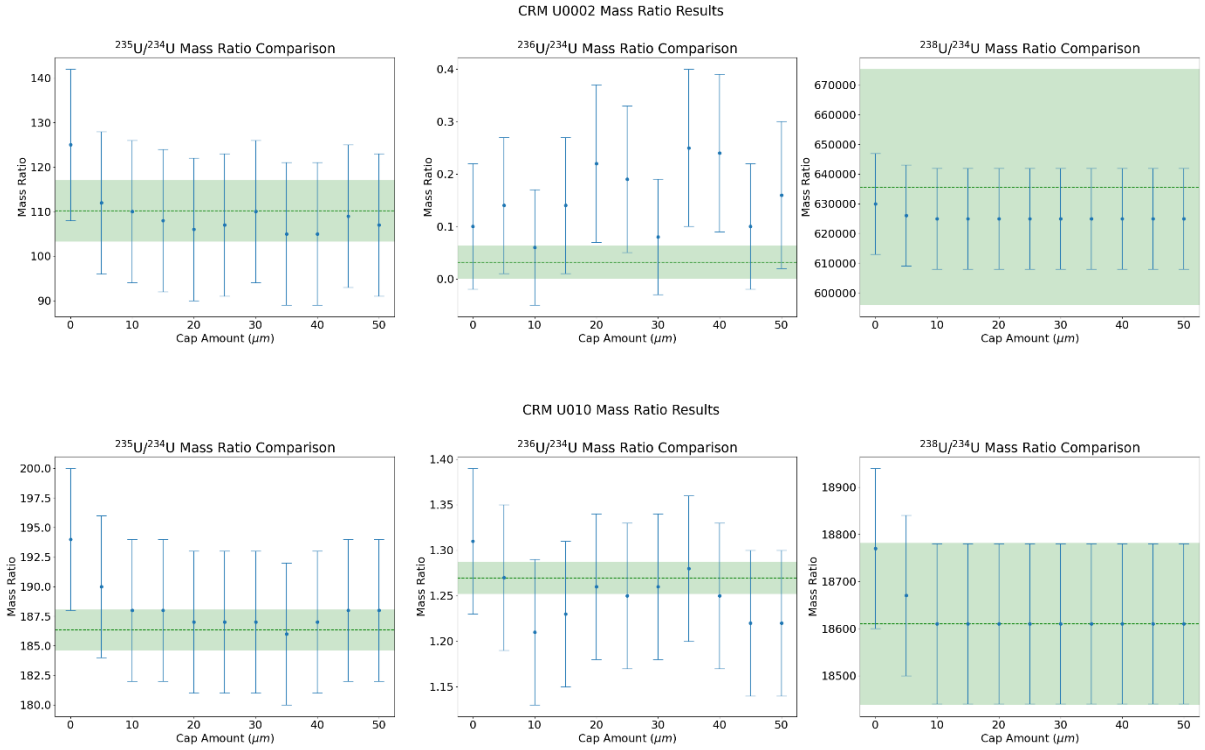
SENSITIVITY TEST RESULTS

E.1. Analysis of Simulated Experimental Spectra with a 50-µm Cap Simulated Correction Spectrum

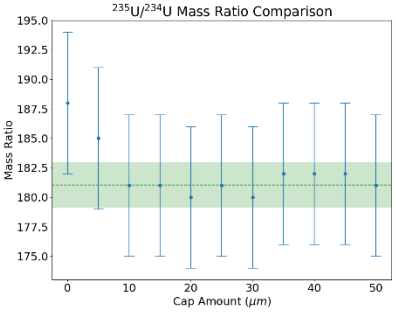


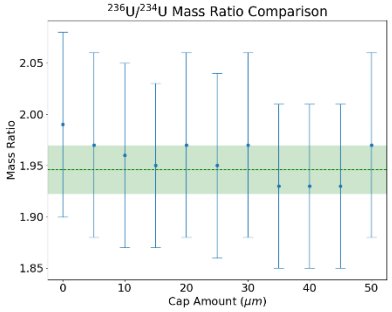


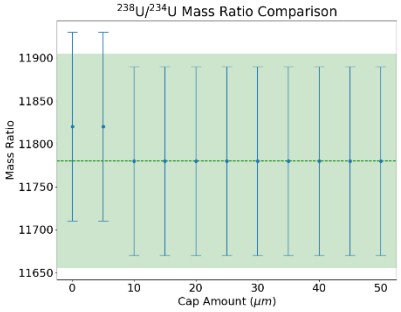
E.2. Analysis of Simulated Experimental Spectra with a 0-µm Cap Simulated Correction Spectrum



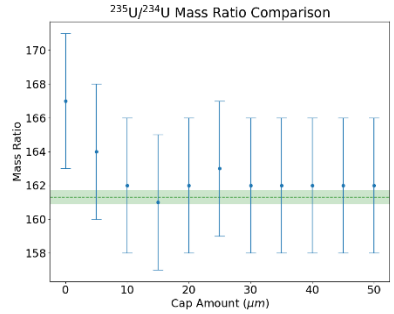
CRM U015 Mass Ratio Results

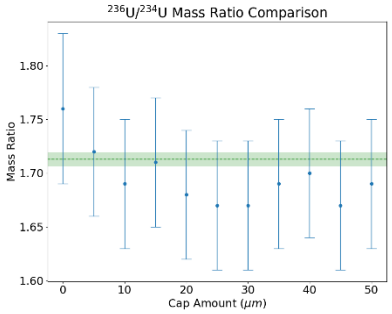


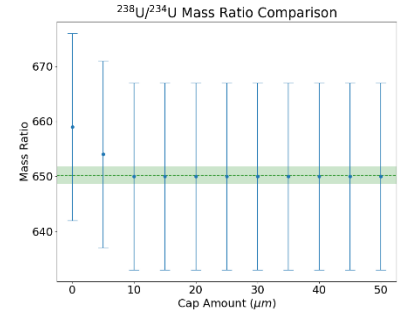




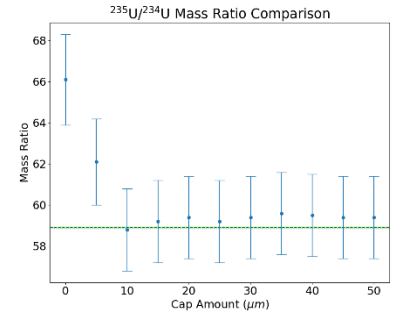
CRM U200 Mass Ratio Results

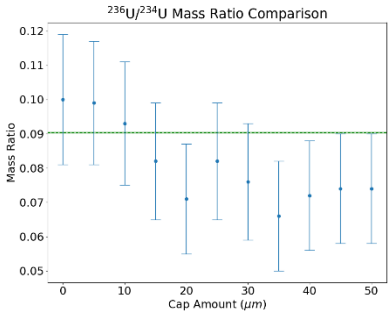


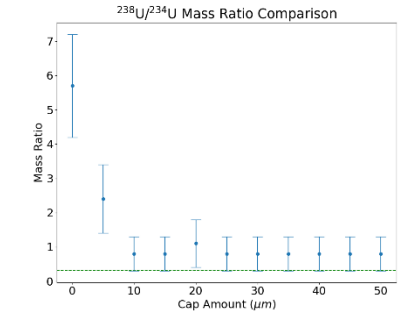




CRM U970 Mass Ratio Results







R e f e r e n c e s

[1] N. Vajda *et al.*, Handbook Rad. Anal. Fourth Ed. **1**, 493-573 (2020).
[2] G. Jia, J. Jia, J. Env. Rad. **106**, 98-119 (2012).
[3] K. Koehler *et. al.*, Appl. Sci. **11** (9), 4044 (2021).
[4] Y. Jang *et. al.*, Appl. Rad. Iso. **70** (9), 2255-2259 (2012).
[5] D. Mercer *et al.*, Nucl. Tech. **207**, S309-S320 (2021).
[6] A. Tollefson *et al.*, App. Rad. Iso. **172**, 109693 (2021).
[7] A. Fleischmann, C. Enss, G. Seidel, Topics App. Phys. **99**, 151-216 (2005).
[8] J. Ziegler *et al*., Nucl. Instrum. Methods Phys. Res. B. **268**, 1818-1823 (2010).
[9] A. Hoover *et al.*, Anal. Chem. **87** (7), 3996-4000 (2015).
[10] D. Reilly, N. Ensslin, in Nuclear Safeguards, Security, and Nonproliferation, edited by J. Doyle (Butterworth-Heinemann, Burlington, MA, 2008), pp. 31-62.
[11] R. Fitzgerald *et al.*, J. Res. Natl. Inst. Stan. Tech. **126**, 126048 (2021).
[12] R. Davis, Rev. Mod. Phys. **75**, 985 (2003).
[13] M. Aker *et al.*, Phys. Rev. Lett. **123**, 221802 (2019).
[14] M. Aker *et al.*, Nat. Phys. **18**, 160-166 (2022).
[15] M. Croce *et al.*, J. Low Temp. Phys. **184**, 958-968 (2016).
[16] K. Koehler, Ph.D. Thesis, Western Michigan University, 2019.
[17] J. Allison *et al.*, Nucl. Instrum. Methods Phys. Res. A. **835**, 186-225 (2016).
[18] J. Allison *et al.*, IEEE Trans. Nucl. Sci. **53** (1), 270-278 (2006).
[19] S. Agostinelli *et al*., Nucl. Instrum. Methods Phys. Res. A. **506** (3), 250-303 (2003).
[20] E. Rutherford, Philosophical Magazine **5** (47), 109-163 (1899).
[21] R. Horansky *et al.*, IEEE Trans. App. Supercon. **23** (3), 2101104 (2013).
[22] T. Sampson, J. Parker, "Equations for Plutonium and Americium-241 Decay Corrections" (unpublished).
[23] M. S. Basunia, Nuclear Data Sheets **107**, 3323 (2006).
[24] M. S. Basunia, Nuclear Data Sheets **107**, 2323 (2006).