Fall 2022

Search for Triple-Proton Decay Using Machine Learning With CUORE

Douglas Adams

Search for Triple-Proton Decay using Machine Learning with CUORE

by

Douglas Adams

Bachelor of Science
University of Michigan 2012

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Physics

College of Arts and Sciences

University of South Carolina

2022

Abstract

A framework to search for a triple-proton decay of ^{130}Te in the CUORE detector against a background of muons is presented. We use machine learning to classify different kinds of energy depositing events. We use the classification information to improve our detection or non-detection limits of a triple-proton decay process. We derive and use a methodology of combining Poisson counting statistics with supervised classification machine learning tools. Additionally, a sensitivity calculation is provided which uses the classification counting likelihood. Using our analysis technique, we achieve an lower 2σ half-life bound of $7.43 \times 10^{24} yrs$ for triple-proton decay of ^{130}Te.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation for Triple-Proton-Decay Search

The instability of nucleons is theoretically well motivated target to look for physics beyond the standard model. The standard model Lagrangian was designed with $SU(3)\times SU(2)\times U(1)$ local symmetries, and with Poincare global symmetries. Baryon number conservation, Lepton number conservation, and Baryon-Lepton (B-L) number conservation are so-called "Accidental Symmetries" in the standard model physical processes. Because these properties are "accidental" they are good targets for theorists to violate with new models [12].

Single proton decay (p-decay) is the most simple way to violate Baryon number. Single proton decay is also highly motivated from the perspective of Grand Unified Theories (GUTs). One of the most anticipated GUT models for testing was the Georgi-Glashow model [13] which was proposed in 1974 and described a SU(5) unified group that can be spontaneously broken into the standard model. The model was elegant, reduced to the standard model in low energy regimes, and implied a single proton would decay. Thus searches for single p-decay have been done quite exhaustively and SU(5) is widely regarded as ruled out. [14].

However, triple nucleon decay searches have not been as widely performed, and are thus good candidates for a new search of B-L number violation [15]. The Standard Model has an anomaly-free discrete Z_6 symmetry [15, 16]. The Z_6 symmetry can stabilize single nucleons, while allowing groups of 3 nucleons to decay. The most dominant decay processes associated with the Z_6 symmetry are:

$$\begin{aligned} ppp &\rightarrow e^+ + \pi^+ + \pi^+ \\ ppn &\rightarrow e^+ + \pi^+ \\ pnn &\rightarrow e^+ + \pi^0 \\ nnn &\rightarrow \bar{\nu}\pi^0 \end{aligned} \tag{1.1}$$

Of the four processes shown in Eqn 1.1 we will focus on the first and most dominant tri-nucleon decay process which is the triple proton decay. We will also limit the scope of our search to three protons decaying within the nucleus of a Tellurium 130 atom.

$$^{130}Te \rightarrow ^{127} In + e^+ + \pi^+ + \pi^+ \tag{1.2}$$

The triple proton decay channel from ^{130}Te is the focus of this research. After three protons decay within a large atom, one would see a positron, and two pions emitted. If the protons decayed from within the nucleus of a ^{130}Te atom, the resulting outgoing particles would have 2.71 GeV of kinetic energy shared between them.

Searches for Tri-Nucleon decay have been done recently with two previous experiments. The Majorana demonstrator [17] did a search in 2018 and obtained a 90% confidence lower bound on the life-time of Tri-Nucleon decay in ^{76}Ge of $10^{25}yr$ using $35(kg)(yr)$ of data exposure. The Majorana demonstrator analysis approach involved searching for decay products. The demonstrator searched for γ-rays from ^{73}Ge, β^- from ^{73}Ga, γ-rays from ^{73}Zn and β^- from ^{73}Cu The corresponding spectroscopy of the daughter nuclei was analyzed. They use plain cuts to reduce their candidates, and achieve their result.

The EXO200 collaboration [18] did their own search in 2018, to get a final 90% confidence lower bound of $10^{23}yr$ and used $223(kg)(yr)$ of exposure. The EXO analysis technique was to compare the energy spectrum of every event they observed with and without Tri-Nucleon decay of ^{136}Xe as a source. They concluded that their best fit model without Tri-Nucleon decay fit the spectrum better than their best fit model including Tri-Nucleon decay.

1.2 Neutrino Properties

The neutrino is a spin 1/2 particle in the standard model. There are three neutrinos $[\nu_e, \nu_\tau, \nu_\mu]$ which have left-handed chirality. There are three anti-neutrinos $[\bar{\nu}_e, \bar{\nu}_\tau, \bar{\nu}_\mu]$ which have right-handed chirality. The original neutrino models assumed them to be massless, however more recent experiments have shown that they are not massless. Neutrinos oscillate between their 3 flavor states as they travel. At any given time a single neutrino is a linear combination of the different flavor states.

Separately from the neutrino flavors, we denote three neutrino-mass eigen-states as $[\nu_1, \nu_2, \nu_3]$, and their masses as $[m_1, m_2, m_3]$. The mass eigen-states and the flavor eigen-states are each complete bases that can represent the state of a single neutrino. Because they are each complete, any mass eigen-state can be represented as a linear combination of flavor states, and any flavor state can be represented as a linear combination of mass eigen-states. It is more common in recent literature to represent the state of a neutrino with mass eigen-states. The combination of mass states is described by the Pontecorvo-Maki-Naka-Sakata (PMNS) mixing matrix (U).

$$|\nu_\alpha >= \sum_i^3 U^*_{\alpha i} |\nu_i > \tag{1.3}$$

Neutrino oscillation experiments have measured the difference between the squares of two pairs of observable state masses. We also know from cosmological data analysis that the total sum of all three neutrino masses is less than a millionth of that of the electron.

$$\begin{aligned} m_1^2 - m_2^2 &\approx 7 \times 10^{-5} eV^2 \qquad \text{Solar} \\ |m_1^2 - m_3^2| &\approx 2 \times 10^{-3} eV^2 \qquad \text{Atmospheric} \end{aligned} \tag{1.4}$$

$$m_1 + m_2 + m_3 < \frac{m_e}{10^6} \quad \text{Cosmology} \tag{1.5}$$

The information of the mass differences, leads to 2 possible cases for the absolute masses. There are two possible hierarchies we are denoted as "normal" and "inverted" hierarchies. In the normal hierarchy $m_1 < m_2 < m_3$ and in the inverted hierarchy $m_3 < m_1 < m_2$. The mass difference information cannot distinguish which hierarchy is correct, and cannot state the exact values of the masses.

1.3 Neutrinoless Double Beta Decay

Neutrinoless double beta decay ($0\nu\beta\beta$) is a well motivated theoretical decay which is important for probing neutrino physics models. In $0\nu\beta\beta$ decay there are two neutrons which simultaneously decay into two protons, and release only two outgoing electrons (Fig 1.1).

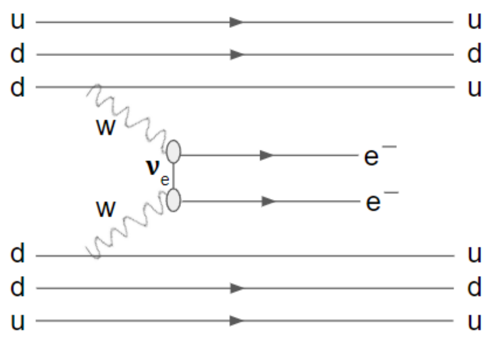


Figure 1.1 An neutrinoless double beta decay feynman diagram. There are a large number of particles involved in a double beta decay, which require calculation of nuclear matrix elements using nuclear structure theory.

Bounds on the $0\nu\beta\beta$ decay rate constrain properties of neutrinos in multiple ways. The $0\nu\beta\beta$ rate constrains normal vs inverted hierarchy. If the decay exists, then we can say that neutrinos are Majorana particles (they are their own antiparticle). If we observe the decay we can also say that lepton number violation exists. All these properties can be further pinned down with experiments highly sensitive to the $0\nu\beta\beta$ decay rate.

The $0\nu\beta\beta$ decay rate can be written as follows:

$$\frac{1}{T_{1/2}^{0\nu\beta\beta}} = G_{0\nu\beta\beta}(Q,Z)|M_{0\nu\beta\beta}|^2| < m_{\beta\beta} > |^2 \tag{1.6}$$

Where $T_{1/2}^{0\nu\beta\beta}$ is the half-life, $G_{0\nu\beta\beta}(Q,Z)$ is a phase-space factor, where Q is the energy released, Z is the number of protons, $M_{0\nu\beta\beta}$ is a nuclear matrix element, and $< m_{\beta\beta} >$ is the "effective" Majorana mass of the electron-neutrino. The effective Majorana mass is further defined as a linear combination of neutrino eigen-state masses:

$$< m_{\beta\beta} > \equiv |\sum U_{\alpha i}^2 m_i| \tag{1.7}$$

Where U is the PMNS mixing matrix (mentioned in previous section), and m_i is a neutrino mass. Because the effective majorana mass can be expressed in terms of the neutrino masses, bounding it will directly constrain the neutrino masses. The effective majorana mass cannot be measured directly, but the $0\nu\beta\beta$ half-life $T_{1/2}^{0\nu\beta\beta}$ can be measured. Thus constraining $T_{1/2}^{0\nu\beta\beta}$ will constrain the neutrino eigen-state masses m_i.

In experimental searches for double beta decay, one possible background to contend with is single beta decay. Double beta decay occurs from a nucleus (A,Z) to (A, Z+2) and single beta decay occurs from nucleus (A, Z) to (A,Z+1). In both single and double beta decays the parent nucleus (A, Z) must be heavier than the daughter nucleus for the decay to occur. If the single beta decay is allowed, the measurable detector signal from double beta decay will be overwhelmed by single beta decay. Thus, experiments are designed with isotopes that energetically forbid single beta decay. The easiest method to forbid single beta decay is to design the experiment with an isotope which has two properties: 1) Has an even number of protons as neutrons (even-even) and 2) is lighter than its (A, Z+1) beta decay result nucleus. In practice double beta decay is searched for using large, even-even isotopes, which have a highly suppressed or energy-forbidden single beta decay rate.

1.4 The CUORE Experiment

The Cryogenic Underground Observatory for Rare Events (CUORE) is an experiment specifically designed to search for neutrinoless double beta decay $0\nu\beta\beta$. CUORE is an underground detector located at the Gran Sasso National Laboratory in Italy. The ton scale experiment consists of 988 TeO_2 crystal bolometers. The primary goal of CUORE is to search for neutrinoless double beta decay, by looking at energy deposits in each of the single $0.75kg$ crystals. Recently CUORE has begun to look for more kinds of signals, and has begun to look at simultaneous energy deposits in multiple crystals. We will be using CUORE to probe the stability of nucleons using data in the regime of high numbers of crystals per single event (high multiplicity events), and at events with GeV scale energies.

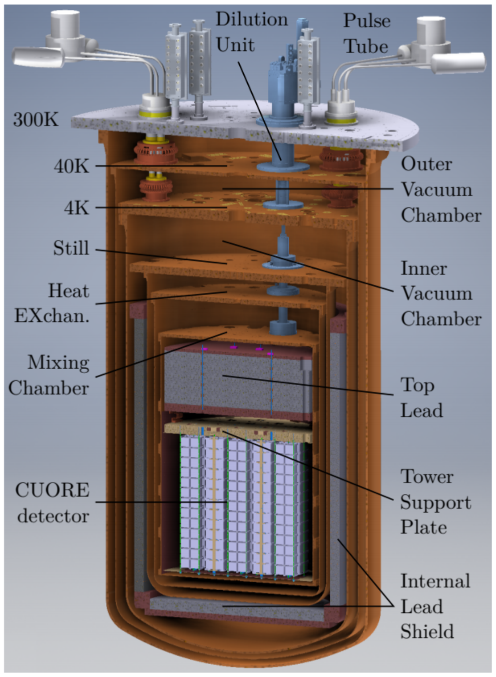


Figure 1.2 A rendering of the CUORE detector. The image was copied directly from Ref. [1, 2]

The CUORE detector is designed to keep and hold the 988 crystal cubes at a low temperature of 15 millikelvin (mk). [2]. To keep the crystals at such a low temperature, 6 nested vessels are required, two of which are separate vacuum chambers. The coldest inner temperature is achieved using an external dilution refrigerator connected to a mixing chamber. The low temperature is maintained with pulse tubes. The mixing chamber is thermally connected to a copper frame hanging below the mixing chamber. The copper frame houses the crystals. Surrounding the entire detector is a lead shield. The lead shield contains a lower than usual count of internal radioactive elements to further reduce contaminant background radiation.

Each crystal within CUORE is composed of tellurium dioxide (TeO_2). Using tellurium dioxide has several advantages in the search for nutrinoless double beta decay. Tellurium 130 (^{130}Te) has a large natural isotopic abundance of 34% so it is low cost compared to some other nuclei. ^{130}Te is an "even-even" nucleus which is lighter than ^{130}I. Thus ^{130}Te is disallowed to single beta decay into ^{130}I. Instead ^{130}Te only allows double beta decay into ^{130}Xe.

Each tellurium oxide crystal has an attached germanium thermister mounted with a thermal paste epoxy such that they are strongly-thermally coupled. A crystal and it's attached thermister are assumed to share the same temperature at any given time. If a decay occurs from within a tellurium oxide crystal we expect to observe a corresponding change temperature using the thermister. The combination of a tellurium crystal and it's attached thermister is denoted in literature as a "bolometer". Each crystal is also weakly-thermally coupled and mounted to the 15mk copper frame using Polytetrafluoroethylene (PTFE or Teflon) brackets. The PTFE brackets sit between the crystal cube and the copper frame.

The thermister consists of a small germanium block, and two gold wires circulating a measurable current through the germanium block within a simple circuit. CUORE uses germanium thermisters which have been doped with neutrons from a reactor so they have temperature sensitive resistance properties. At the low temperatures in CUORE, any temperature fluctuations will change the resistance provided by the germanium thermister connected to the crystal bolometer. The germanium thermister, is kept in a simple circuit of gold wires and two other resistors each chosen to have higher resistance than the germanium to keep the current in the circuit near constant.

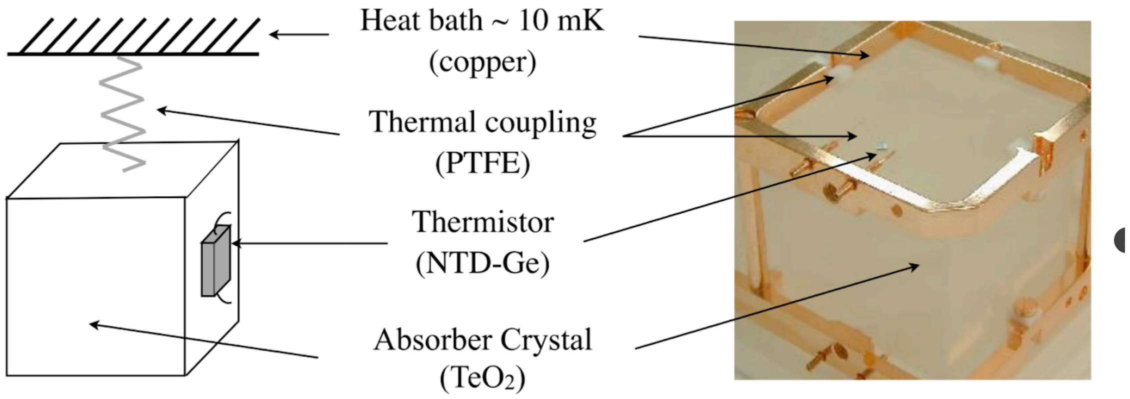


Figure 1.3 Single Crystal Within CUORE. Image copied from Ref. [3]

When a particle deposits energy in a tellurium dioxide crystal the temperature of both the TeO_2 crystal and its attached germanium thermister will rise. There will be an associated measurable quick increase in voltage and corresponding decrease in resistance of the germanium thermister. After energy is deposited, the thermistor-bolometer pair will both slowly cool back down to the temperature of the copper frame. The cooling time is of order 1-5 seconds depending on the amount of energy deposited and the PTFE mount properties. These voltage changes for single energy deposits in CUORE are referred to as "pulses".

1.5 CUORE Neutrinoless Double Beta Decay Results

The CUORE experiment has achieved a lower half-life bound for neutrinoless double beta decay $0\nu\beta\beta$ of $T^{0\nu}_{1/2} > 2.2 \cdot 10^{25}$ yr at 90% confidence level. To achieve the $0\nu\beta\beta$ result, CUORE applied cuts, a blinding procedure, and a Bayesian parametric unbinned likelihood analysis. Background simulations were performed using the qshields software (described in sec 2.2), for each known source of background.

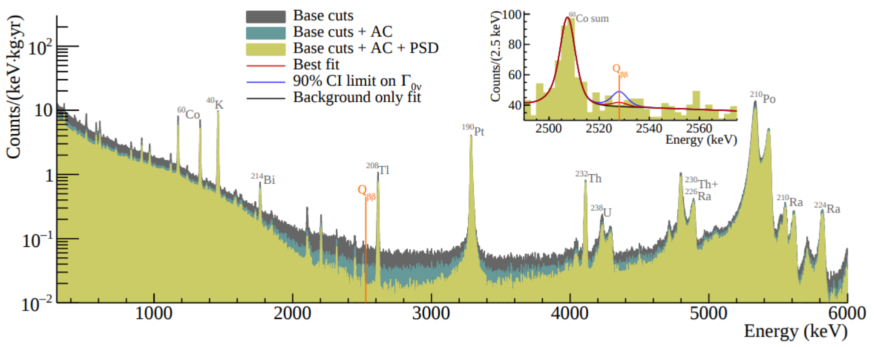


Figure 1.4 The full single multiplicity, single crystal energy spectrum observed by CUORE. All events which have time coincidence to deposit energy simultaneously in more than one crystal has been discarded. Figure copied from [4]

To search for the result a few different cuts were applied upon data in an attempt to remove more background than signal. The first cut applied was to remove periods of time with high noise, assessed by comparing the width of the measured energies at a known energy source calibration line at 2615 keV. In addition to removing those periods of time, events that created energy deposits in more than one crystal were also discarded, knowing that from simulations 88% of the $0\nu\beta\beta$ decays would only deposit energies in single crystals within CUORE. The last major kind of cut was to look at the individual pulses, and discard those which did not appear "physical" in shape.

The analysis was performed using unbinned bayesian parameter fitting allowing for each known source of energy contribution to be modeled as a probability density function in energy, and then summed to produce the full spectrum. CUORE subscribes to the technique of "blinded" analysis which is when the analysis framework is developed without having full access to the real data. One such method of blinding is known as salting, where fake data is mixed in real data. CUORE blinded the real data by salting it with an additional artificial set of single crystal energies distributed about a mean energy value. The entire analysis framework was developed upon the salted data, before removing the salt. CUORE then performed a fit on the real data with and without an additional $0\nu\beta\beta$ energy source. CUORE found no preference for inclusion of the additional source when trying to reproduce the the real data energy spectrum.

The CUORE experiment and our $0\nu\beta\beta$ result has already had impact in two ways. CUORE has shown that large cryogenic dilution refrigeration systems are feasible, and next generation experiments such as CUPID, AMoRE, SuperCDMS, CRESST will benefit from the cryogenic innovations. The $0\nu\beta\beta$ result itself has implications for the standard model, by showing that so far, there is no evidence that the neutrino is a majorana particle.

1.6 Overview of Search Methodology

We will search for triple proton decay with the CUORE detector. The major background we have to contend with is cosmic ray muons. We will simulate both triple proton decays and muons from the atmosphere. We use the simulations to learn what each kind of event looks like in CUORE. We pick some distinguishing properties of each kind of event which we will denote as "features". We train a machine learning classification algorithm upon the simulations, and their associated features. We use combination of counting statistics for expected Muon background, as well as the machine learning classification to place a bound on the triple proton decay observation count within CUORE, and associated half-life for the decay of ^{130}Te by this process.

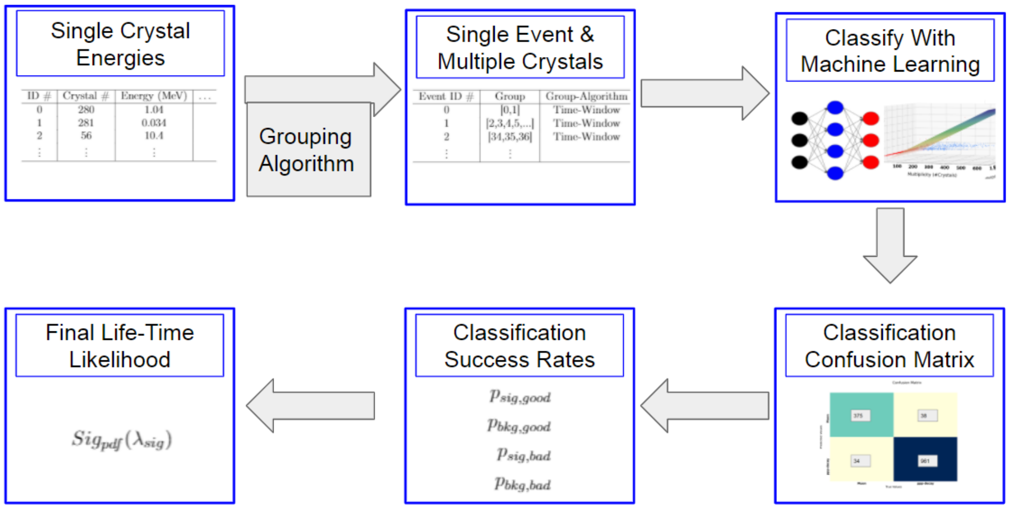

Single Crystal Energies
ID # | Crystal # | Energy (MeV) | ...
0 | 280 | 1.04
1 | 281 | 0.034
2 | 56 | 10.4
Grouping Algorithm
Single Event & Multiple Crystals
Event ID # | Group | Group-Algorithm
0 | [0,1] | Time-Window
1 | [2,3,4,5,...] | Time-Window
2 | [34,35,36] | Time-Window
Classify With Machine Learning
Classification Confusion Matrix
Classification Success Rates
$p_{sig,good}$
$p_{bkg,good}$
$p_{sig,bad}$
$p_{bkg,bad}$
Final Life-Time Likelihood
$Sig_{pdf}(\lambda_{sig})$

Figure 1.5 Workflow Diagram: The figure illustrates the basic approach to determining a final likelihood of the life-time and associated confidence bounds of a Tri-Nucleon decay using a classification algorithm. We start with a table of every crystal that lights up in CUORE (Top-Left). The single crystals are grouped into events (Top-Middle). The events are classified using a version of Machine Learning algorithm (Top-Right). The classification results are stored in a "confusion matrix" (Bottom-Right). Then the confusion matrix is transformed into classification probabilities (Bottom-Middle). The classification probabilities can be used to formulate a final likelihood (Bottom-Left).

Chapter 2

Simulating Events of Interest

We have simulated the events we want to classify: simulations of both signal, and background were required. To perform the simulations CUORE uses Geant4 which is a computational "toolkit for simulating the passage of particles through matter" [19]. We needed to do many iterations of full simulations of Muons from the sky, [20, 21] and of triple proton decays [15] within ^{130}Te.

CUORE has already built software which makes use of Geant4 called "qshields" [22, 6, 7]. The qshields code contains all the relevant geometry within CUORE. By geometry we mean many of CUORE's internal components as described in Sec 1.4 including: lead shield, copper framing, thermal plates, and of course the TeO_2 cubes which are thermally monitored producing CUORE data.

We can ask qshields to spawn any standard model particle at any point in space, with any initial energy and direction. The qshields code will then take that particle, and simulate the standard model physics that would happen subsequently. The qshields code will progress through time, allow the particle to travel through any rendered materials. As the particle travels through a material, it will lose energy by transferring some energy to nearby electrons it travels near (electron excitation or ionization), and it will have a chance to collide with a nucleus, triggering a decay into outgoing particles. The qshields code will then spawn subsequent necessary outgoing particles from reactions of interest, and follow them through time and space. The code will keep propagating all particles until an end condition is triggered. Any energy from a simulated spawned particle is kept track of within the simulation. Ultimately

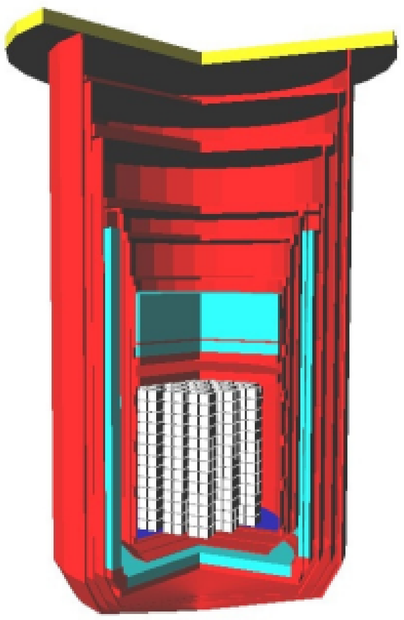
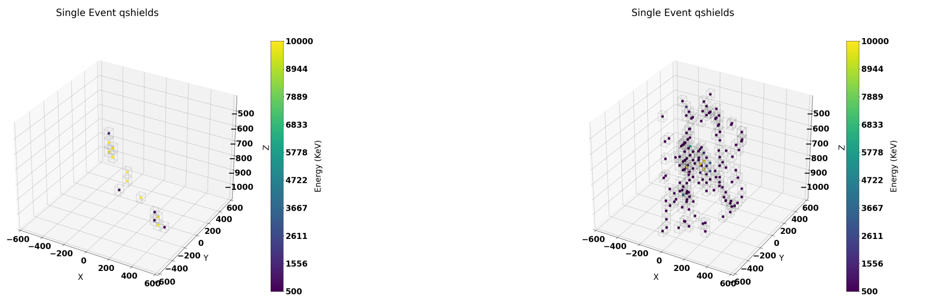


Figure 2.1 A qshields rendering of the CUORE detector. "The copper plates inside the 10 mK shield are in red and blue. The internal lead shields are in cyan, and the TeO_2 array is in white." Image and caption quote taken from the PhD thesis of Barbara Sue Wang. [5] (Page 89)

all the initial energy from an incoming simulated particle will either be deposited in the form of heat into a material, or will be lost in the form of particles traveling out of CUORE. The energy deposited within each of the crystals in CUORE is measured by a small attached germanium thermistor that constantly monitors the crystal temperature. Finally at the end of simulating the full reaction chain of a single particle, the output from the qshields code will contain a table of Tellurium cubes, and the amount of energy deposited within each.

Using the simulation software we can simulate both triple-proton-decays (ppp-decays) (Fig 2.3) and Muons(Fig 2.8). The result from a simulation will show how much energy is deposited in each crystal, for each kind of event. Typical examples simulation results for each kind of event are shown in Fig 2.2.

There are wildly varying possible decay chains for both signal and background events. With real data we will not be able to take an arbitrary single event and know



(a) Background: Typical Atmospheric Muon (b) Signal: Typical Triple Proton Decay

Figure 2.2 Examples of CUORE's simulated response to both a single background event (left), and single signal event (right)

exactly what caused the various energies in each individual crystal (we will not be able to reconstruct decay chain path information illustrated in Fig 2.3). However, we can see that Muons and the Triple-Proton decays have geometric properties that are distinguishable. For example: Muon events have "track-like" (Fig 2.2a) energy deposits and the Triple-Proton decay events have more "blob-like" (Fig 2.2b) energy deposits.

In the next section we will quantitatively define properties such as "track-like" and "blob-like" as well as some other properties of single events. These properties are known as "features". These features can be used distinguish between different kinds of events and are explained in detail in Sec 3.

2.1 Simulations: Triple Proton Decay

To simulate the Triple proton decay signal that we are searching for (Section 1), we use the qshields CUORE software. We know such a decay will result in three outgoing particles: [e^+, π^+, π^+] Therefore to simulate a triple proton decay event, we can use qshields to spawn each of the three outgoing particles at the same start location, at the same time. We will be looking for the decay to happen from within a ^{130}Te atom.

$$^{130}Te \rightarrow ^{127} In + \pi^+ + \pi^+ + e^+ \tag{2.1}$$

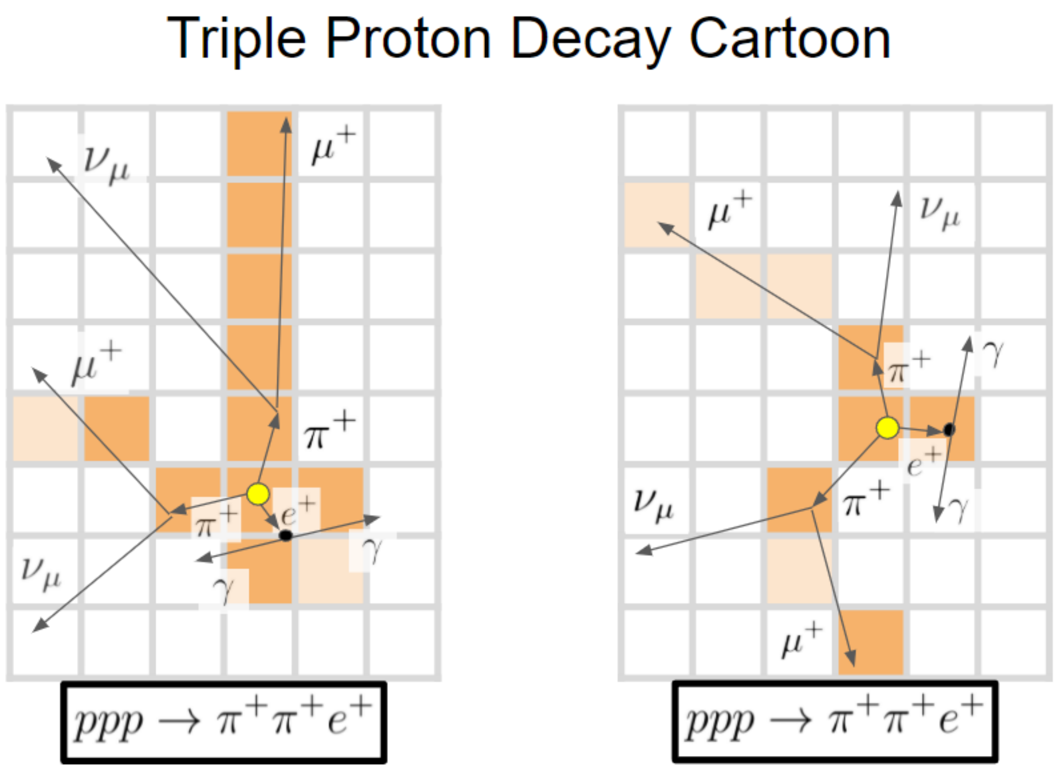


Figure 2.3 Two conceptual illustrative depictions of single ppp decay events in CUORE hitting a grid of crystals. The orange blocks represent crystals with energy deposited by any resulting particles from the decay chain. The darker orange blocks represent when crystals are saturated with energy, and CUORE can only detect that more than 10MeV of energy was deposited in the crystal. The lighter orange blocks represent events where some detectable energy is present, and an accurate energy measurement is observed. The white blocks represent other crystals for which no measurable energy was deposited by the decay chain. Note how in both examples, the events do not look like a single track of energy deposits (a straight line fit would a be poor approximation).

A single triple proton decay simulation is split into three smaller simulations: one for each of [e^+, π^+, π^+]. Such a simulation split is reasonable so long as the decay from each outgoing particle does not have a chance interact with the decay from another outgoing particle. Each single outgoing particle qshields simulation starts off the particle with an initial location, direction, and energy.

2.1.1 PPP-Decay Initial Conditions: Choice and Justification

The initial conditions are chosen in a manner to best reflect the information we have on the triple proton decay model. Initial start locations are chosen to be within a randomly chosen single TeO_2 crystal. Initial directions are chosen to be uniformly random, and allowed to propagate outwards from the start location. We also assume no angular correlation between the outgoing particles. Kinetic Energy of each outgoing particle is chosen as randomly as possible under the condition that the sum of initial kinetic energies of three outgoing particles equals 2.71GeV.

In this section we will show that these assumptions are reasonable. We limit our search to Triple proton decays that would happen within a ^{130}Te nucleus. We assume the nucleus to be at rest within the reference frame of the detector. A ^{130}Te nucleus that undergoes a triple proton decay would leave a ^{127}In nucleus behind, as well as produce the three outgoing particles: [e^+, π^+, π^+].

We need statements of energy (E) and momentum (P) conservation:

$$E_{Te130} = E_{In127} + E_{\pi_1^+} + E_{\pi_2^+} + E_{e^+} \tag{2.2}$$

$$0 = \overrightarrow{P}_{In127} + \overrightarrow{P}_{\pi_1^+} + \overrightarrow{P}_{\pi_2^+} + \overrightarrow{P}_{e^+} \tag{2.3}$$

Table 2.1 presents the masses of relevant particles:

Name	Symbol	Mass
Tellurium 130	^{130}Te	129.906 GeV
Indium 127	^{127}In	126.917 GeV
Pion Plus	π^+	139.570 MeV
Positron	e^+	0.510 MeV
Proton	p^+	0.938 GeV

Table 2.1 Triple Proton Decay Masses

We also need to know how much energy is released through the decay. In a triple proton decay a ^{130}Te nucleus would lose three protons, and turn into an ^{127}In nucleus.

The difference in nucleus mass is turned into the new particle masses as well as kinetic energy split among the resulting particles.

$$m_{Te130} = m_{In127} + 2m_{\pi+} + m_{e+} + E_{Kinetic} \tag{2.4}$$

The kinetic energy is then found to be 2.71 GeV (using known particle mass values) where the kinetic energy is the difference between total energy and rest mass energy.

$$E_{Kinetic} = m_{Te130} - (m_{In127} + 2m_{\pi+} + m_{e+}) \quad \approx 2.709.35GeV \tag{2.5}$$

One can show that ^{127}In nucleus will remain non-relativistic and obtain a negligible amount of energy from the decay. The kinetic energy of the ^{127}In is also negligible, because the mass of the nucleus is very large compared to the other particles, while at the same time the the total momentum must be conserved. The kinetic energy in each of the three outgoing particles [e^+, π^+, π^+] on the other hand, could range anywhere from 0 to 2.71 GeV. (Details in Sec 2.1.2) Thus splitting the full 2.71 GeV of kinetic energy of the decay randomly between the three outgoing particles is a reasonable assumption. The momentum conservation (Eqn 2.3) implies that the ^{127}In's recoil direction can balance out any choice of momentum for the other three outgoing particles, so arbitrary random direction for the outgoing particles is also reasonable assumption.

2.1.2 PPP-Decay Initial Conditions: Energy Choice Constraint

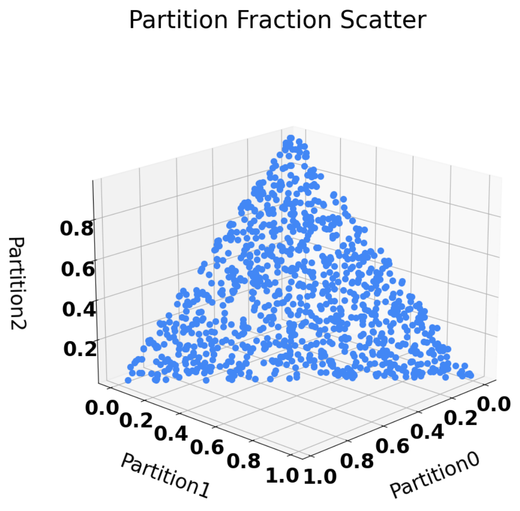


Figure 2.4 Illustration of how a uniform amount of energy is split between three outgoing particles. For illustration purposes the total amount of energy in this plots is scaled to 1. All values can be rescaled to units of 2.71 GeV to obtain real samples. The sum must equal a constant value therefore we are sampling from a plane in 3D space.

To obtain a more correct answer without making any limiting assumptions, we could derive a multi-dimensional probability density function using the maximum entropy principle (Sec 5). We could take our Energy and Momentum conservation equations (Eqn 2.2 2.3) as constraints, and then use Lagrange multipliers to derive a probability density function. Performing the maximum entopy calculation would derive a result that works in both relativistic and non-relativistic cases for all particles involved. The amount of work required to perform the maximum entropy result is beyond the scope of this work. As we highlighted in the previous sections example, the maximum amount of energy that could possibly given to the Indium-127 is 0.03GeV of kinetic energy and the large atom nucleus can absorb random combinations of other momentum directions fairly easily. We have chosen not to obtain the more correct entropy-based energy split.

Instead of performing a maximum entropy calculation, we feel justified in making the following initial condition assumptions: We split the 2.71 GeV of energy between the three outgoing particles. The energy conservation condition (part of Eqn 2.2) implies a constraint for possible energy choices, and takes the form of a plane in three dimensions. The momentum conservation condition (also Eqn 2.3) does not place any constraint on the energy or direction of the the particles. Under the conditions we need to satisfy, we want to divide the energy as randomly as possible.

Our initial conditions amount to choosing energies from the triangle shown in Fig 2.4. We simply need an algorithm to choose points randomly upon the triangle. The triangle describes the positive energy region a plane defined by Eqn 2.2.

2.1.3 PPP-Decay Initial Conditions: Energy Split Algorithm

Splitting a finite energy E between K items is a continuous analog of the "Stars and Bars" problem [23]. We can take a line segment, and choose two un-correlated break points uniformly randomly upon the segment. We can then assign the distance from 0 to the first partition as the first energy, the distance between the two partitions as the second energy, and the distance from the second partition to 2.71 as the third energy. Illustration of choice of "break points" and resulting partitions are illustrated in Fig 2.5.

We will denote the algorithm we used as the "break point algorithm". Despite the fact the algorithm is very simple, and definitely not novel, the break point algorithm is very efficient at producing correct behavior. We can visualize a real world example by imagining picking up a stick from a yard and breaking it at 2 random locations upon it's length. The results of executing "break point" algorithm for a line segment of length 1 are shown in Fig 2.4 and Fig 2.6. The same samples are shown in both figures and were drawn using the "break point algorithm". To obtain energies for our triple proton decay, we need only take the samples generated in the two figures, and re-scale their values to 2.71 GeV scale (multiply a set of samples by 2.71 GeV).

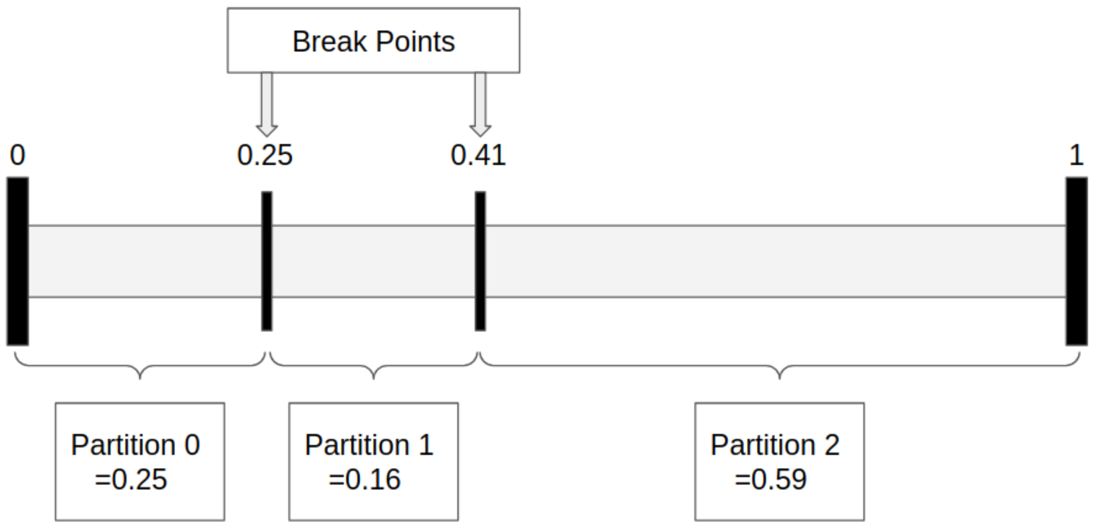


Figure 2.5 An example of a single sample generated by splitting a line segment of length 1, into random partitions of length between 0 and 1. The Break Points are chosen uniformly randomly from 0 to 1, and the resulting partition segment lengths are not uniformly random in length. If each partition length was drawn from an independent uniform random segment, then the sum of partition lengths could not possibly add up to a fixed value. A representative sample of partition length choices can be seen in Fig 2.4. The partition length distributions are illustrated in Fig 2.6.

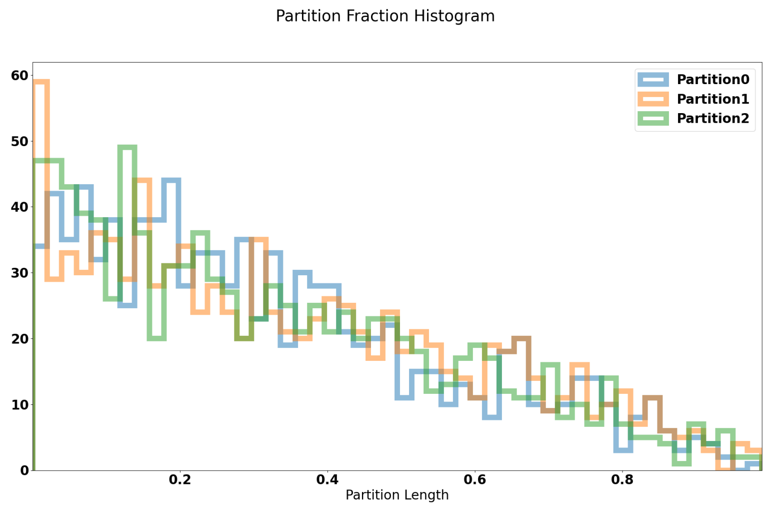


Figure 2.6 The histogram shows the energy distribution of each of the single events. Individually, each outgoing particle shares the same non-uniform probability density function. The samples shown on these distribution histograms are the exact same as the samples shown in fig 2.4

After all each of the three smaller simulations (one for each outgoing particle) have completed, we can ask how much energy was deposited within each TeO_2 crystal. We can sum up any energies deposited in TeO_2 by the 3 outgoing particles. The total energy deposited within each crystal from a group of three outgoing particles is then stored in a data file. We can then repeat the whole process many times to produce as many triple proton decays as we would like. To produce simulations modeling N triple proton decays, we need $3N$ simulations (For 1 triple proton decay we need 3 qshields simulations, for 2 decays we need 6 simulations, etc...).

2.1.4 PPP-Decays: Typical Simulation

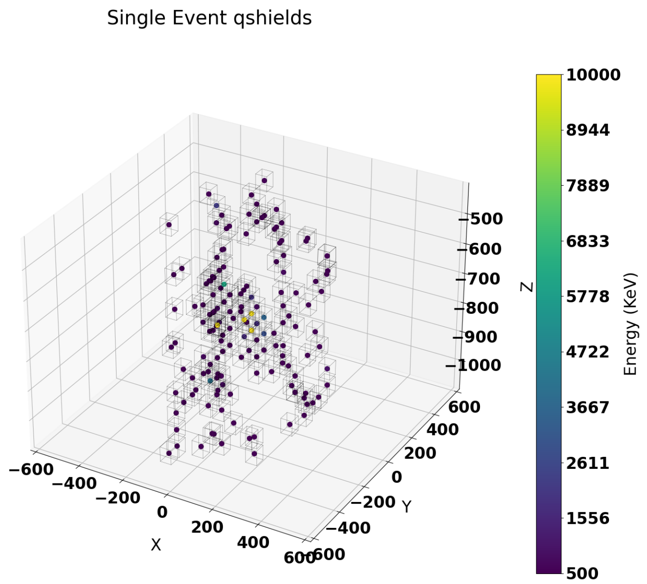


Figure 2.7 Three outgoing particles from a ppp-decay are simulated [e^+, π^+, π^+]. The energies deposited in each crystal by all three particles is simulated and summed up using qshields [6] [7]. Cubes in the figure represent crystals in CUORE. The color of dots represent the amount of energy deposited within each crystal by the event. The energy deposits do not appear single-track-like.

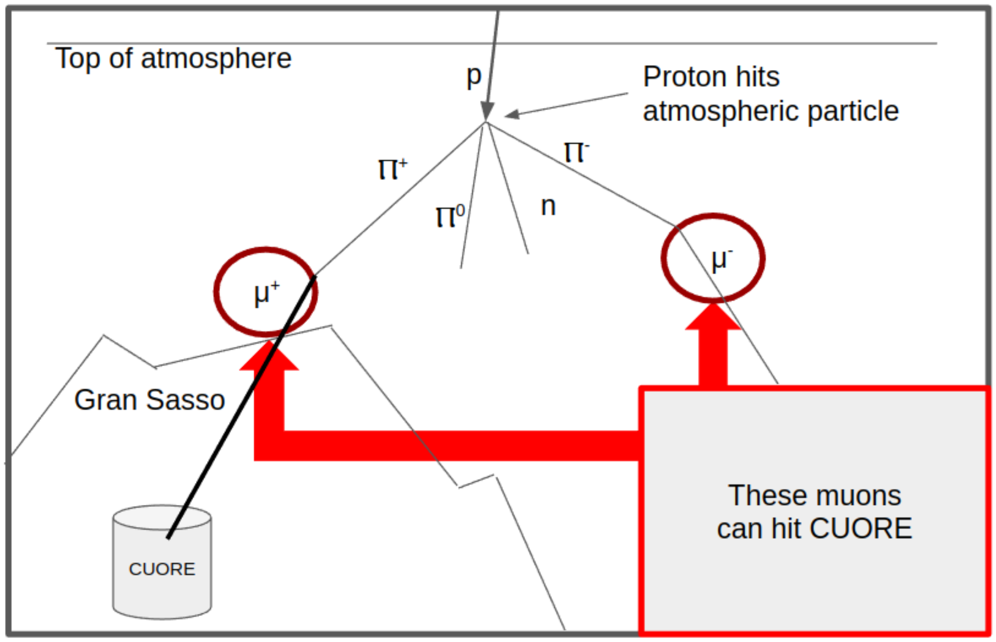


Figure 2.8 An illustration of a cosmic ray creating a muon in the atmosphere that hits the CUORE detector at LNGS.

2.2 Simulations: Muons

In a search for new physics that would yield high multiplicity events, the primary source of background is cosmic ray muons. Muons are a major background to contend with in any detector inside the LNGS. For the case of searching for triple proton decay with CUORE, muons would be the only known source that would both produce a comparable total energy and crystal multiplicity to the signal we are looking for.

Cosmic ray muons are the only type of muons which are observed by detectors at LNGS. Free protons in the galactic medium hit the earth's atmosphere, collide with atmospheric gasses to produce pions, then the pions decay into what we denote "cosmic-ray muons". An observed cosmic-ray muon in the CUORE detector traverses a segment of the Earth's atmosphere, the mountain rock containing the LNGS cave, the lead shield around CUORE, and some crystals within CUORE. An illustation of the typical path of a muon produced by cosmic rays en route to the detector is shown in Figure 2.8.

A small secondary source of muons which CUORE can observe are "neutrino induced muons". Neutrinos can enter some point on the opposite side of the earth from CUORE (e.g. American Somoa) travel through the entire earth, collide with some matter close to the edge of the Earth's crust under the laboratory (below Grand Sasso), decay into a muon, and then muon can continue to travel upwards a small distance through the rest of the earth's crust without colliding again, and then finally go through the lead shield, and hit a detector in LNGS. These neutrino induced muons make up between 1% and 2% of the moun flux at LNGS, and from the perspective of the detector are traveling upwards. While CUORE does not have event time signature precision to tell upwards from downwards muons, the Borexino experiment has measured them [24]. Very few muons in LNGS are neutrino induced. For our analysis we will completely ignore them.

2.2.1 Muons: Initial Conditions, Choice and Justification

With the goal of identifying the background muons and estimating their rates in the detector, we perform simulations of the muons in CUORE. To start, we need a probability distribution for the initial conditions for muons entering the detectors, i.e. for those muons that made it through the mountain and into LNGS. The qshields code calculates an estimated under-the-rock flux for average rock, taking into account the amount of rock traversed by muons coming in from different directions (see Fig 2.9). The muon energy loss as a function of the amount of matter traversed, dE/dx, is obtained from the Particles Properties from Particle Data Group [25]. Thus qshields generates the distributions of energy and direction of motion for muons inside LNGS.

The CUORE collaboration's qshields muon simulation starts the muons upon 5m half-sphere around CUORE (Fig 2.10). A location on the half-sphere is picked randomly, with the muon direction and energy taken from the distribution described above (using the LNGS direction and energy). If the muon's initial direction points

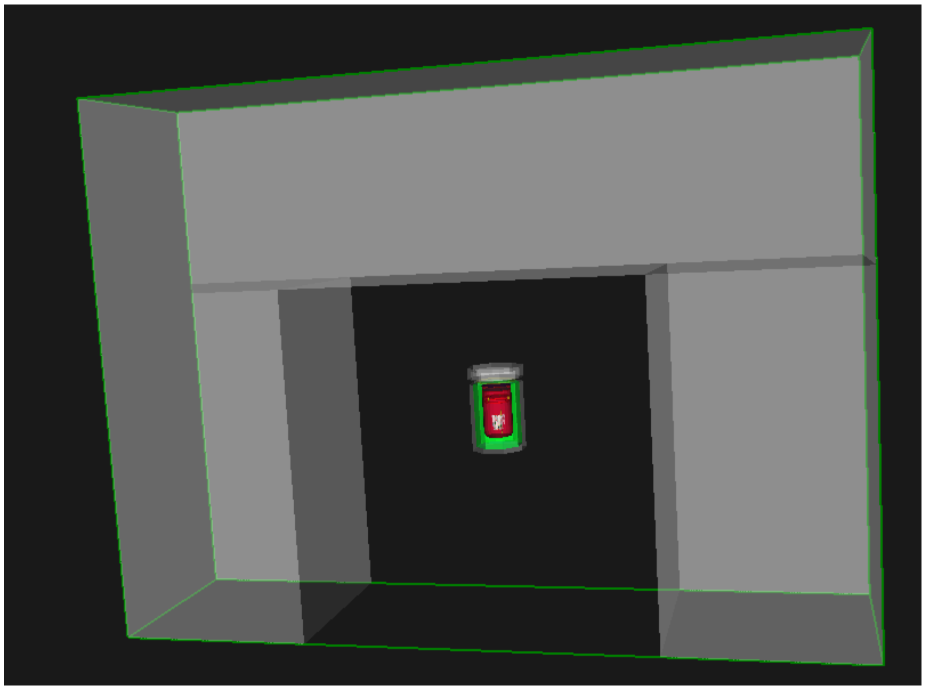


Figure 2.9 Render Image was produced by Stefano Pozzi using qshields software. Rock is included above CUORE for the simulations. Muons are minimum ionizing, so while traveling through the rock lose energy linearly. Additionally they have a small chance to collide, and produce produce particle showers starting within the rock. It turns out that including the rock for simulations does not significantly change the resulting simulated CUORE dataset. Showers produced from the rock above would have to start right before they breach the end surface of rock, and after a shower starts most of its contents will hit the lead shield surrounding CUORE. For practical purposes only the muon makes it through the lead shield to deposit energy in the detector.

inwards upon the half-sphere, the initial location is kept, and the simulation continues. Then we let the muon try to enter the detector (with that energy and direction) and use GEANT to obtain the track and energy deposited by the muon and the particles produced by its interactions in the detector. The GEANT tool will calculate how much energy the muon loses as it travels through materials; the decay or collision of the muon with material (with certain probability); in the case of decay it follows the outgoing particles and their energy deposits, etc.

The typical muon on the initial half-sphere can travel in a direction such that it will miss CUORE and will not deposit any energy in the crystals (See Fig 2.10).

Muon Start Locations

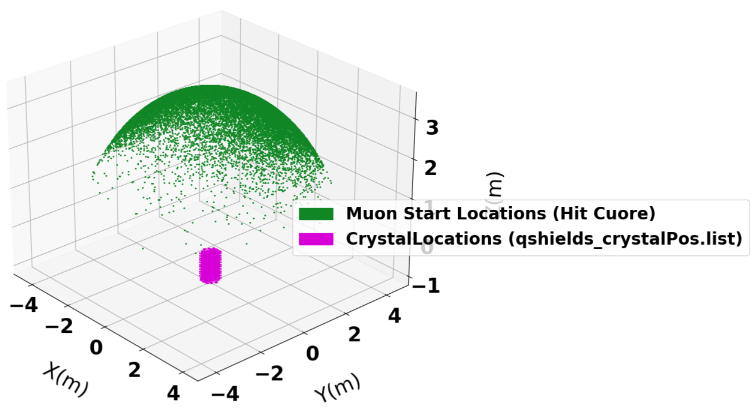


Figure 2.10 Initial muon locations are chosen upon a half sphere. Initial locations are shown (in green) are those for the subset of simulations which happen to deposit energy with CUORE. Note: there exist initialized muons which begin further down on the edge of the half-sphere. Those initialized further down have a much lower chance to hit CUORE and deposit energy, because their initial direction has a good chance to cause them to miss CUORE entirely.

We perform 10 million muon simulations, such that roughly 100,000 of the particles actually pass through the crystals in detector and leave a detectable signal.

2.2.2 Muons: Typical Simulation

2.3 Calculating the Expected Muon Count as a Prior

We will use the muon simulations (Sec 2.2) as a tool to calculate an expected CUORE muon counting rate. We adopt the LNGS muon flux rate of $3.41 \times 10^{-4} m^{-2} s^{-1}$ deduced from the Borexino experiment [24]. A rough approximation of the muon flux at LNGS is approximately one muon per square meter, every three hours.

In this section we will outline a simple approximation procedure to combine the muon flux rate in LNGS with the muon simulations, to calculate an expected CUORE observed muon count rate. A more detailed discussion regarding more careful treatment of the calculation can be seen in Appendix A. We chose to use the simple

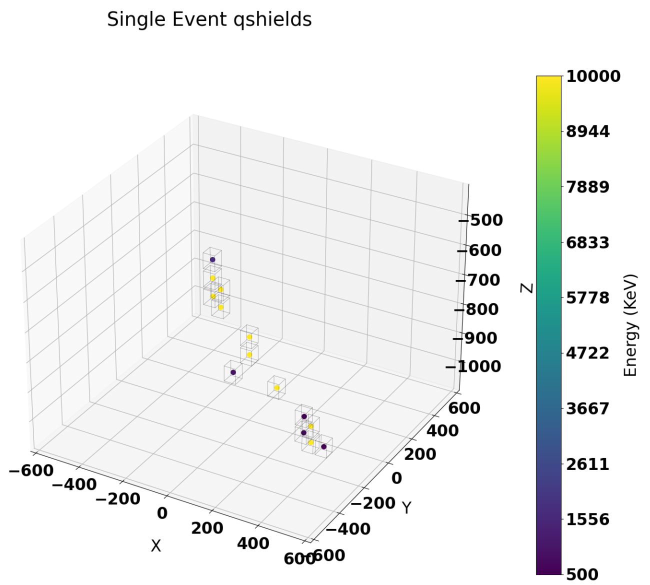


Figure 2.11 For a muon that enters CUORE, qshields computes the amount of energy deposited in each of the detector's crystals. Cubes in the figure represent crystals in CUORE. The color of dots indicates the amount of energy deposited in each crystal by the event. The yellow dots are saturated crystals and show the majority of any energy deposited. The saturated yellow crystals appear track-like in nature.

approximation, in favor of the more careful treatment because the expected number calculated is much closer to the amount observed. The simple approximation equation for exepcted total count is shown below:

$$\text{Expected Count} = (\text{Muon Flux}) \times (\text{Hit Probability}) \times \text{Time} \times (\text{Effective Area}) \tag{2.6}$$

The first two terms are defined below and use simple fixed values. We assume the LNGS **muon flux** upon the surface of the half-sphere of initial muon simulation locations (Fig 2.10). The exposure **time** spanned by the real data-sets we intend to use in the analysis [3612, 3613, 3614, 3615], is approximately 188 days, or half of a year.

We call the probability for a single given muon starting on the half-sphere to produce a detectable muon in CUORE a **hit-probability**. From the many muons we start upon the half-sphere, only a small percentage actually hit CUORE and deposit a meaningful level of observable energy (greater than 500keV). From the many muon simulations initialized upon the half-sphere, we calculate a find a hit probability of 0.0063851. Any given muon started within a muon simulation, has about a 0.63% chance to deposit more than 500 keV in more than ten crystals within the CUORE detector.

To calculate the **effective area**, we need to use information given from the initial half-sphere. The half-sphere has a radius of 5 meters, and has total surface area of $157m^2$. The half-sphere is down-shifted by 1.5 meters, such that the bottom portion is below the location of the internal CUORE crystals. Of the simulated muons which actually hit CUORE, all of them were initialized above the $1.5m$ height. The surface area of the portion of the half-sphere above crystal-height is approximately $71.44m^2$, and we use this value as the effective area.

Using the above values for exposure time, LNGS muon flux rate, effective area, and hit probability, we expect CUORE to observe 2537 muons. We also note that the corresponding the number of muons we would expect from nature to go through that half-sphere to be 397,338 muons given the MACRO muon flux, effective area, and exposure time. We can then approximate a Gaussian prior probability distribution for λ_{bkg} to be that number (2537). We assume that the random variable is truly Poisson distributed, and reduces to the Gaussian limit where the standard deviation is equal to the square root of the mean. For a more careful treatment of the expected count calculation, issues, and possible future improvements see Appendix A.

Chapter 3

Selecting Single Event Features

X	Y	Z	Energy (keV)
103	176.5	-722.8	2549.34
103	117.5	-490.8	510.99
191	176.5	-606.8	3638.59
191	176.5	-548.8	10000.0
191	176.5	-490.8	10000.0
191	176.5	-432.8	10000.0
176.5	264.5	-490.8	1704.67

Table 3.1 Example Single Muon Event Dataset

The resulting data from an event simulation is a list of crystal locations and corresponding energy deposited amounts (Table 3.1). The information in its raw format is not sufficient for a machine learning algorithm. We need to reduce each simulation of a single event into a few numbers that can be compared with different simulations. Such comparable numbers are called a "features".

In this section we define a **Dataset** as a tabular set of data $\overrightarrow{X}$ with any number of rows and columns. Each row represents a single object that we are to observe, and each column represents a different observable property (Example shown in Table 3.1). We will be using consistent notation referring to the same dataset $\overrightarrow{X}$, and designing different ways to map the entire dataset into a single number called a "feature".

The best features we can choose are those that can be used to distinguish between different kinds of events. We determined a list of features which can be used to classify muons and ppp-decays manually (human inferred features). Automatically determining features is an active area of research in computer science at the time of writing this. New algorithms are being built to determine features as part of a larger field of "interpretable AI" research. We relegate algorithmically determining useful features that describe muons and ppp-decays as a body of future work.

The single event features we are considering include: Total Energy, standard deviation of crystal energies for a single event, number of crystals hit (Multiplicity), geometry of the hit crystals represented as "Principal Components", energy weighted principle components, and the number of saturated crystals within a single event. We will describe each of the features in detail in this section. The features can then later be used by a machine learning classification algorithm (Sec 6).

3.1 Features: Total Energy & Multiplicity

The total energy of a given event is simply the sum of the measured energies within each crystal.

$$E_{feature,tot} \equiv Sum(\overrightarrow{X}_{energy,column}) \tag{3.1}$$

The multiplicity of an event is the number of total crystals which have energy deposited within the detector. In our case, the number of such crystals is represented within the dataset as it's length.

$$M_{feature,multiplicity} \equiv length(\overrightarrow{X}) \tag{3.2}$$

3.2 Feature: Saturated Crystal count

Due to current limitations of bolo-metric signal processing techniques, CUORE only has accurate energy resolution for energy regions of E < 10 MeV. The example event shown in Table 3.1 has rows which illustrate the limitation. For the example event, there are exactly 3 saturated crystals.

$$S_{feature} \equiv Count(\overrightarrow{X}_{i,Energy} >= 10MeV) \tag{3.3}$$

3.3 Feature: Principle Components

The technique of principle components is general, and used widely throughout machine learning (6). Because we are specifically only using principle components for defining a feature of a single event, we choose to place the section here. Additionally, because we will use the details of the calculation to define our own new feature in this section, we go into more detail for its calculation than we would otherwise.

Here we outline the mathematics for obtaining the principle components of dataset. In our case we are interested in applying principle components to a list of crystals that have energy deposited within them for a single event.

For our case we want to model the single event as some kind of "blob" of energy. First we need to separate out the spatial position [X,Y,Z] and model it. We can find a covariance matrix (Σ) of only the position columns of the dataset $\overrightarrow{X_{pos}}$. The elements of a covariance matrix ($\Sigma_{i,j}$) are the covariance (Cov) between two columns (i, j) of the positions, divided by the degrees of freedom D_{dof} of the dataset.

$$\begin{aligned} D_{dof} &= length(\overrightarrow{X_{pos}}) - 1 \\ \Sigma &= Cov(\overrightarrow{X_{pos}})/D_{dof} \\ \Sigma_{i,j} &= Cov(X_i, X_j)/D_{dof} \end{aligned} \tag{3.4}$$

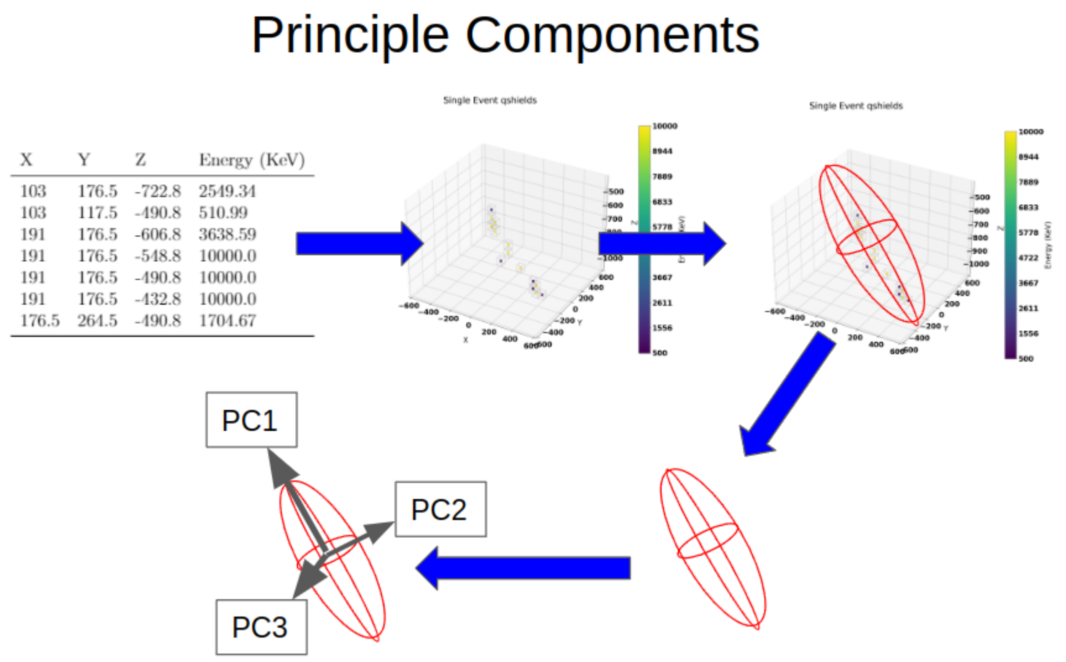
There exists a computationally efficient trick for calculating the covariance matrix. One can re-center the dataset about a mean of zero by subtracting off the true position mean $\mu_{X,Pos} \equiv mean(\overrightarrow{X}_{pos})$, then multiplying the centered dataset by transposed copy of itself, before dividing by the number of degrees of freedom.

$$\begin{aligned} \mu_{X,pos} &= [X_{avg}, Y_{avg}, Z_{avg}] \\ \overrightarrow{X_{pos,ceni}} &= \overrightarrow{X_{posi}} - \mu_{X,pos} \\ \Sigma &= \overrightarrow{X_{pos,cen}}^T \overrightarrow{X_{pos,cen}} \frac{1}{D_{dof}} \end{aligned} \tag{3.5}$$

After we have the covariance matrix, we will use the information within it to describe the shape of the dataset. It is helpful to look at the eigen-vectors $\overrightarrow{v}_k$ and corresponding eigen-values λ_k of the covariance matrix (Σ).

$$\Sigma \overrightarrow{v}_k = \lambda_k \overrightarrow{v}_k \tag{3.6}$$

The eigen vectors and eigen values of the covariance matrix can be interpreted as representations the axis of a triaxial ellipsoid. The vectors represent the direction of the axis, and the eigen values represent the magnitudes of the axes.



X	Y	Z	Energy (KeV)
103	176.5	-722.8	2549.34
103	117.5	-490.8	510.99
191	176.5	-606.8	3638.59
191	176.5	-548.8	10000.0
191	176.5	-490.8	10000.0
191	176.5	-432.8	10000.0
176.5	264.5	-490.8	1704.67

Figure 3.1 An illustration of principle components and their interpretation for a single muon event. Each event has a data table, which can be viewed as a scatter plot of energies in 3D space. Then the scatter plot's positions can be modeled as a 3D multivariate Gaussian. The Gaussian can be interpreted as an ellipsoid that has describing axes. The principle component vectors and values describe those axes.

In our case we chose not to be concerned with direction, and only with shape. Mathematically we are choosing to be agnostic to arbitrary rotations of energy deposits, but we are not agnostic to difference in relationship between axes of energy deposits. As a final step we take the ratio between the principle component eigen-values as a metric for "track-like". One can quickly see that if the longest axis describing the energy deposits is much longer than the other two axes, we have an even that is more track-like. Alternatively if we have an event where all three principle component axes are the same length, we have an event that is much more spherical or "blob-like".

We can define an importance of each axis of the ellipsoid approximation. We can define the importance by the magnitude of each eigen-value of the principle component. Previously we defined the eigen values λ_k of the covariance matrix Σ (Eqn 3.6). Now we sort and re-label the eigen-values, according to their magnitude with a numerical index.

$$\begin{aligned}\overrightarrow{\lambda} &= sorted[\lambda_1, \lambda_2, \lambda_3]\\ \lambda_1 &= Max(\overrightarrow{\lambda})\\ \lambda_2 &= Middle(\overrightarrow{\lambda})\\ \lambda_3 &= Min(\overrightarrow{\lambda})\end{aligned} \tag{3.7}$$

Thus we define the principle component importance values (PC_k) below:

$$\begin{aligned}PC_1 &= \frac{\overrightarrow{\lambda}_1}{\sum_{i=1}^{3} \lambda_i}\\ PC_2 &= \frac{\overrightarrow{\lambda}_2}{\sum_{i=1}^{3} \lambda_i}\\ PC_3 &= \frac{\overrightarrow{\lambda}_3}{\sum_{i=1}^{3} \lambda_i}\end{aligned} \tag{3.8}$$

Note that the PC_k's are normalized and sum to 1:

$$1 = PC_1 + PC_2 + PC_3 \tag{3.9}$$

There are a few consequences of scaling the components such that they add up to one. We need not use the third principle component value because it is informationally redundant to the first two values. So the spatial principle components we use as features are PC_1 and PC_2 as shown in Eqn 3.8. We are also discarding information regarding absolute size of the ellipsoidal energy deposit. A small number of crystals may provide the same spatial information as a large number over a larger volume. These properties should not hinder our ability to perform classification between muon tracks and triple proton decay.

Additionally, we have also disregarded the energy deposited within the crystals. Thus far, we have treated each crystal as a boolean value. Either there is energy within a crystal, or there is none. We will modify our principle components algorithm to make use of the energy information in conjunction with the spatial information in the next section (Sec 3.4).

3.4 Feature: Principle Components - Energy Weighted

We can design a new principle component metric which takes into account the energy as well as position. Here we will denote the technique as weighed principle components. Previously we calculated the covariance matrix by centering the data about the mean (Eqn 3.5). Now we have to center the data about a weighted mean. First we can normalize the weights by making sure they add up to one. Then we construct the weighted centered dataset.

$$\begin{aligned} \overrightarrow{W} &= \overrightarrow{Energies}/Avg(\overrightarrow{Energies}) \\ \mu_{X,pos,weight} &= \frac{1}{len(\overrightarrow{X})}\sum_i W_i \overrightarrow{X_{pos,i}} \\ \overrightarrow{X_{pos,ceni}} &= \overrightarrow{X_{posi}} - \mu_{X,pos} \end{aligned} \tag{3.10}$$

Next we need to use the weighting information within the covariance matrix. We want to weight each value of the matrix by the relative weight of the element. It is then convenient to use the same algebraic technique as before in Eqn: 3.5 but this time add in a diagonal version of the weights applied to each row of the dataset:

$$\Sigma_{weighted} = \overrightarrow{X_{pos,cen}}^T (\mathbb{1} \cdot \overrightarrow{W}) \overrightarrow{X_{pos,cen}} \frac{1}{D_{dof}} \tag{3.11}$$

We can now use the weighted covariance matrix $\Sigma_{weighted}$ to get weighted principle components PC_{1w}, PC_{2w}. The remaining calculation steps are entirely redundant to those in the previous section, and thus are omitted. However, for completeness and reproducability, a python code snippet is included that performs each step of weighted principle components. We use the well accepted standard "numpy" package as a dependency for linear algebra operations as well as efficient eigen-vector and eigen-value calculations.

```
import numpy as np
W = Energies / np.mean(Energies)
DOF = len(W)
WeightedMean = np.sum(X*np.atleast_2d(W).T, axis=0)/DOF
X_cen = X - WeightedMean
Cov_w = np.dot(X_cen.T, np.dot(np.diag(W),X_cen))/(DOF-1)
PC_vals, PC_vecs = np.linalg.eig(Cov_w)
```

3.5 Visualizations of Many Features

After turning each single event into a set of features, it is useful to be able to visualize them. In our work we present two similar methods of visualization. We have a standard scatter plot, of which the highest number of dimensions is 3 (Fig 3.2). We also have a standard corner plot, which can be used to less precisely view higher numbers of dimensions (Figures 3.4, 3.3). The visualizations themselves are not relied upon in our analysis, so any further discussion of them is limited to the figures.

In the next section we will present algorithms used to classify the different event types. First we must visualize an event as a dot living in within a multidimensional feature space, where each dimension corresponds to a single feature that we calculate with regard to the event. The task of classification amounts to creating partitions in that N-dimensional ($N-D$) feature space that can separate different different kinds of dots represented with different colors. While the visualization images are not relied upon mathematically, they are useful sanity checks for any choice of classification algorithm. If the the different event types are not visually separable in some way, we cannot possibly hope to separate them mathematically or algorithmically. By eye, we can see that the features we have chosen have a good tendency to separate out the different kinds of events, so our ability to classify them should be much better than guessing randomly. Thus our final result should be a strong improvement from the counting statistics technique (Sec 7.3), when progressing to the classification and counting technique (Sec 8).

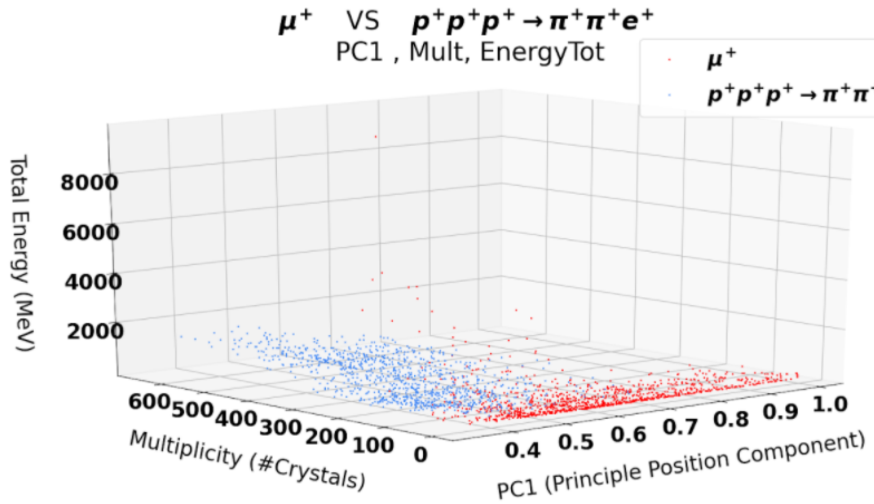


Figure 3.2 Scatter plot of triple proton decay (blue) and Muons (red) single events. Single events are shown in **Feature Space**. Each dot represents a single event, and each axis is a single feature. Each feature approximates all the information about a single event compressed into a single numeric value. For this figure we chose only 3 features for visualization purposes. Our analysis includes additional features corresponding to additional axes in feature space (Figures 3.4, 3.3).

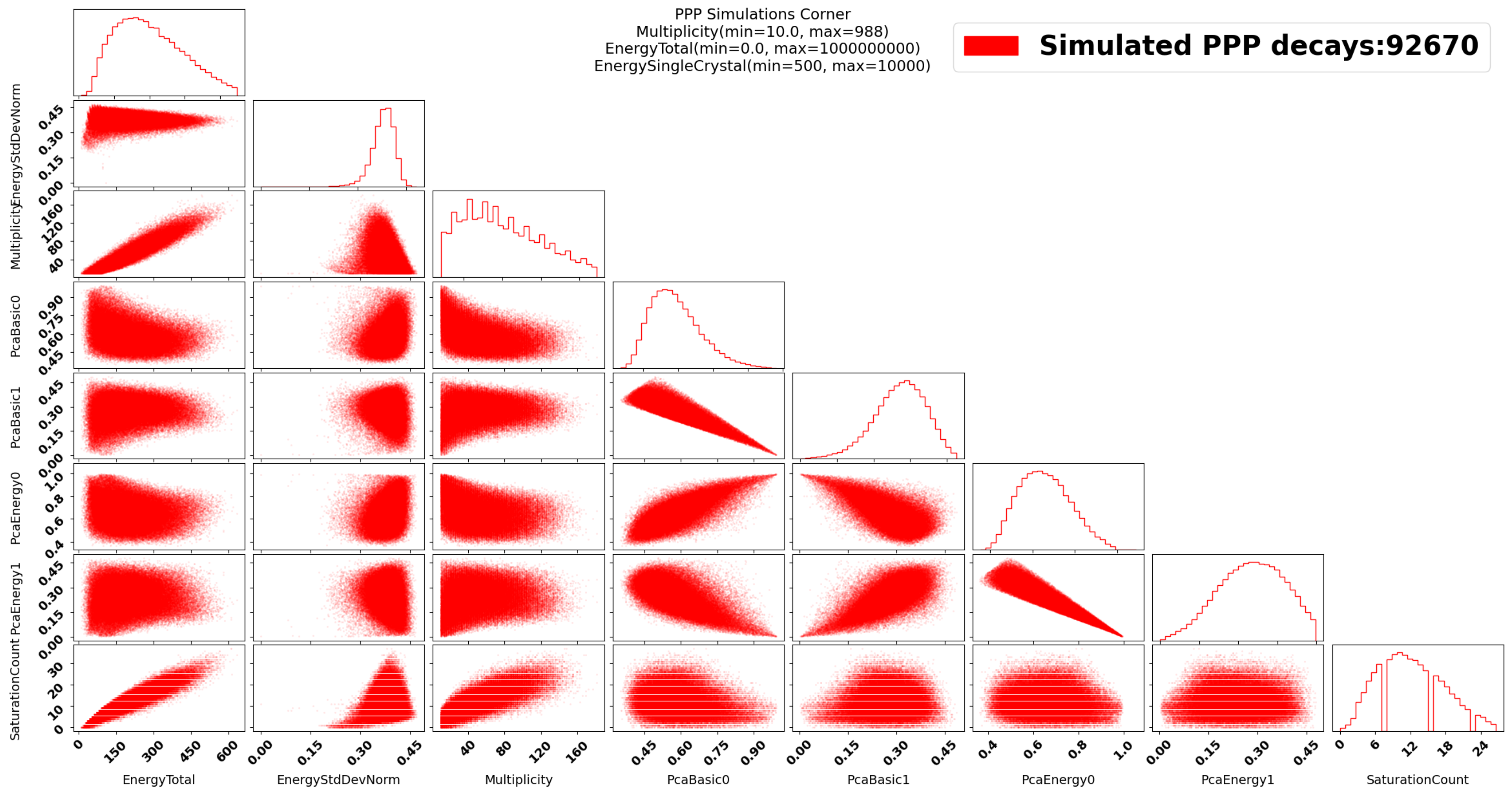


Figure 3.3 Triple-proton-decay simulation features corner plot. The diagonal plots are histograms for each feature. The off-diagonals are each scatter plots corresponding to 2 features. Note: This plot was generated with the corner plot library [8] which has limitations for histograms of discrete values. (Gaps in saturation count, and peaks in multiplicity, are artifacts of the plotting library and are unphysical).

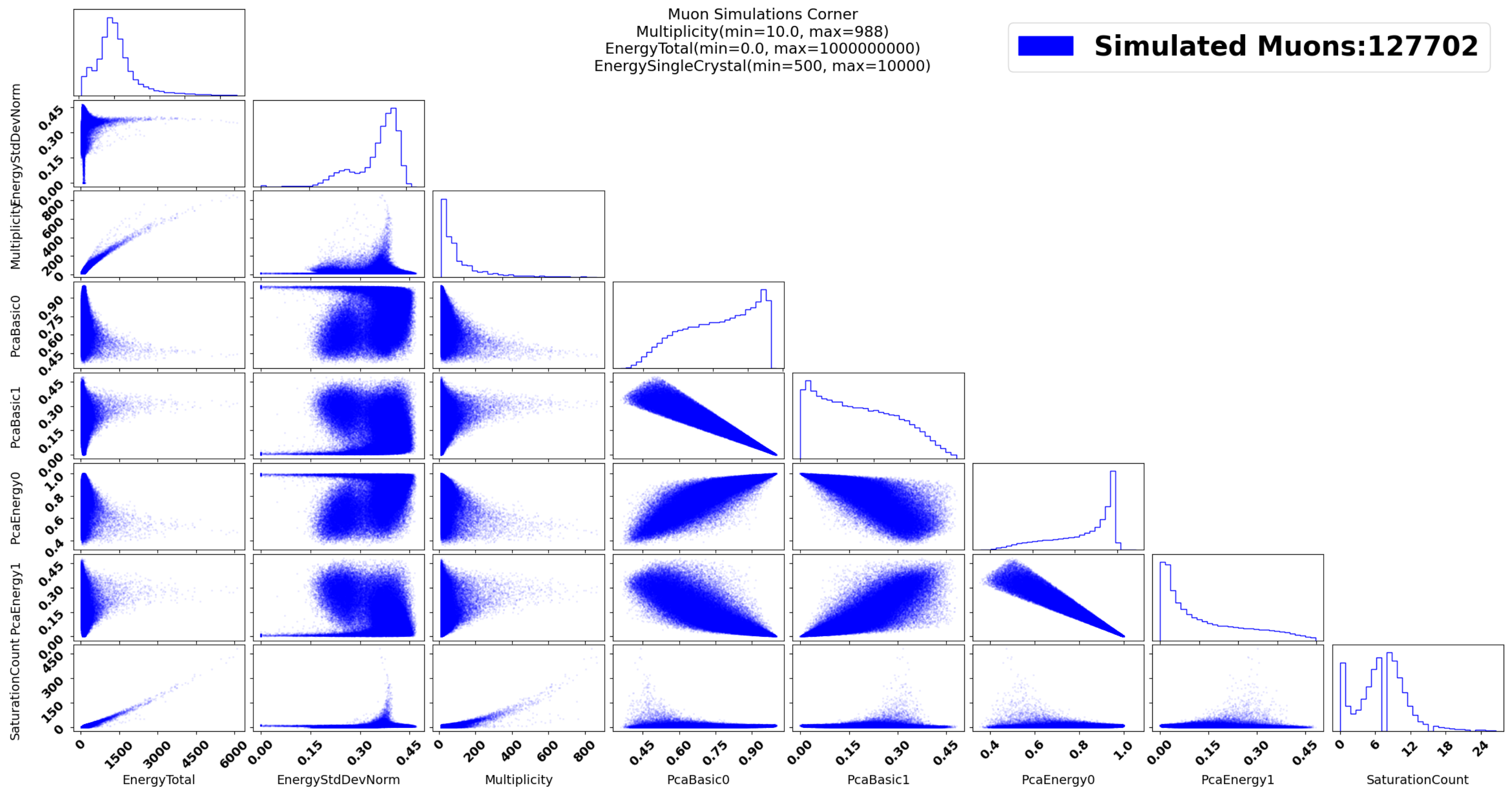


Figure 3.4 Muon simulation features corner plot. The diagonal plots are histograms for each feature. The off-diagonals are each scatter plots corresponding to 2 features. Note: This plot was generated with the corner plot library [8] which has limitations for histograms of discrete values. (Gaps in saturation count, and peaks in multiplicity, are artifacts of the plotting library and are unphysical).

Chapter 4

Extracting Real Data

The CUORE detector is constantly recording information into a complicated database. Extracting the data is a prerequisite to performing any high level analysis. In this section we will describe how CUORE stores it's data at a high level, and how we are extracting the real data from the CUORE database. We review the different data types and storage locations within CUORE. We will explain the way that we group single crystal events together. We show various problems encountered with data extraction. We will explain cleaning procedures that we have introduced to deal with the problems.

4.1 Data Storage Format

CUORE uses "ROOT" to store the bulk of it's data. The "Rapid Object-Oriented Technology" (ROOT) framework was founded by Rene Brun in 1995, which couples together a data file format and a computer-language [26, 27]. The file format is denoted with the ".root" extension. ROOT files are designed to be an efficient storage format for large tables of numbers. The ROOT language is built upon C++, and designed to be able to quickly read the ROOT files to perform some rudimentary analysis and charting/graphing with a few lines of code. At time of writing this thesis, ROOT file storage is standard practice for very large physics data-sets. ROOT is currently used by the Large Hadron Collider to store petabytes of data tables at CERN, which is where the ROOT framework was originally developed. While obviously not as large as the LHC data, CUORE's raw voltage database is on the order of 10's of terabytes in size after relying upon highly optimized ROOT file compression techniques.

CUORE event data in it's most raw format is a list of voltages as a function of time. Each of the 988 crystals within CUORE has its own voltage time series updated every few milliseconds. There exists a logging software which has to store raw voltage time series data to files. CUORE has decided upon the architecture that the logging software is manually stopped then restarted by a human approximately every 24 hours. While the logging software is not running, no voltage information is being stored in the database. Each approximately 24 hour software logging period is denoted by CUORE as a single "run". Each crystal is given a unique number, and each run is given a unique number in the database. For each combination of crystal number and run number, 24 hours worth of raw voltages are stored in a single file in ROOT file format. Many such ROOT files make up CUORE's raw voltage database.

While the voltage time series data is complete for energy depositing events, it is large and unwieldy for general analysis. Simultaneous to logging the voltage time series data, there exists a separate software which is constantly monitoring the voltage for energy deposits in real-time. CUORE has named time series monitoring code "Diana". Energy deposit events in CUORE are commonly referred to as "pulses" in CUORE literature. A pulse exhibits a steep voltage ascent as the corresponding crystal temperature rises, followed by a gradual voltage descent as the crystal cools back down to the steady state temperature of 15mk of CUORE's inner cryogenic chamber [22]. The Diana software constantly searches for energy deposit pulses in real-time as CUORE is logging data. When a pulse is detected, it is analyzed, then deducted analysis parameters are logged in a separate set of root files unique to the run number.

CUORE also has various cryogenic-related sensors (slow-monitor) data time series, which are kept separate from the raw voltage data. The slow-monitor data is stored in a separate ROOT file database, and the the software for monitoring them is kept running constantly. Slow monitor logging is intended to be kept independent of voltage logging "runs". Slow monitor data includes: temperatures of various places within the detector, seismometer readings, coolant tube flow rates, and pulsing flow rate phases between different tubes. The collaboration has a person on shift watching these cryogenic related measurements at all times, as a catastrophic failure could result in serious damage to the detector.

Finally at the highest level CUORE has two web-services called "CORC" and "elog" used by a person with the duty of "remote detector operations shifter". The web-services connect to the the raw voltage database, as well as the cryogenic database, they do various real-time signal processing useful for human visualization, and stores valuable human input about the quality of the data. Different complicated moving averages are taken over the raw voltage time series, other circuitry information (like pulses for calibration), and the slow monitor data. These moving averages are stored in yet another much smaller more easily accessible database, and allow for the web-services to query them. The humans monitoring the cryogenic system and checking the quality of the run data do so through CORC, and manually input time intervals we consider to be "bad intervals" where the quality of data is poor for one reason or another. One common reason that a human would mark a bad interval in the CUORE data is a nearby earthquake, which often shifts the baseline voltage upwards in all the crystals simultaneously. These bad intervals are stored, and subsequently connected to each detected pulse in the database.

4.2 Pulse Energy & Time Estimation

When the start to a pulse is detected, an energy calculation is done. A window of time (usually 10 seconds - 3 before pulse start, and 7s after pulse start) is analyzed to calculate energy deposited. The data for that pulse window is recorded as a detected time, and energy. From a pulse, the energy is estimated using the height of the voltage peak, and the time is estimated using time of the voltage peak. Using the pulse shape analysis software, a separate smaller database is kept with only a list of event energies, and times, for each run. Each Dianna-detected pulse is given an unique ID.

For our analysis we will be restricting our energy range from 500keV to 10 MeV. For high energy events which deposit more than 10MeV, the software has a tendency to under estimate the amount of energy deposited, and it will potentially suggest a delayed event time. High energy events will saturate the crystal, and reach a peak voltage. Saturated events start with a quickly rising voltage, followed by an almost constant plateau at peak voltage, followed many seconds later with a gradual descent towards equilibrium again. Mapping the peak voltage height to an energy in a saturated crystal will then underestimate the energy deposited. Mapping the time of the event to the max of the peak will effectively choose a random time during the plateau. The peak of an expected pulse shape is effectively "decapitated", and of future interest for the CUORE collaboration is to "re-capitate" high energy events, in order to estimate an energy.

4.3 Grouping Many Crystals into a Single Event

From having a list of pulses at various times, we need a procedure to determine if multiple crystals should be grouped together into a single event. We cannot use existing multi-crystal grouping techinqiues used for $0\nu\beta\beta$ decay.

For $0\nu\beta\beta$ decay analysis an existing grouping algorithm is already in use, and being further developed to handle cases from multiplicity 2 to 5. CUORE already has a background model, and associated simulations, where all known sources of energy deposits are considered against our search for neutrinoless double beta decay. From the CUORE background model simulations we can see that non-muon background sources, which produce more than a multiplicity of 5 or more crystals with energy pulse deposits are rare. For multiplicities of [2,3,4,5] various neighboring crystal group orientations in Tetris-like shapes are each considered with different energies and particle-products. Using the Tetris-like neighbor-chain technique quickly becomes infeasible for grouping methodology, as the multiplicity grows beyond about 5 crystals. For events with multiplicity 5 or greater, the only known source of background is muons, and our existing grouping techniques do not work.

For our analysis, we are searching for a ppp-decay which may have 10's or even 100's of crystals with energy deposits simultaneously. If we impose a minimum search of multiplicity of 10 or greater, we lose less than 5% ppp-decay signal efficiency according to our simulations. We also highly reduce any non-muon background which has been thoroughly studied by the CUORE collaboration through the existing background model. To account for the fact that if observed, a triple proton decay would very likely produce an event in CUORE of multiplicity 10 or greater, we need a simple way to group many crystals together that fire pulses at similar times.

With the goal of simplicity, we have chosen find events that happen close together in time, completely ignoring spatial information when grouping them together. CUORE has a resolution on the order of milliseconds for individual pulses. Physics events that take place across the entirety of CUORE will take place at most a 10's of milliseconds apart. CUORE has a maximum distance between any two crystals of one meter. The time required for a full ppp-decay chain, or a Muon (which is relativistic) to traverse the meter entirely will be on the same order as the single pulse time-resolution of the detector.

We will use a grouping algorithm that aggregates any two pulses as long as they are less than 20 milliseconds (ms) apart. To be explicit in illustration let us assume we had a multiplicity 100 event. Our grouping algorithm would allow for us to group a 100 crystal event together, if each crystal had a pulse 19ms one after the next for a maximum duration of 1.9 seconds. In practice however, grouped events have measured pulse times clustered much more closely about some mean. Using this grouping algorithm, we can reasonably say it is unlikely for us to fail to include two crystals which actually happened from the same physics event. We expect to see approximately one multiplicity 10 or greater Muon per hour, so the odds of accidentally grouping two separate Muons into one event are very low. It is possible that our algorithm would group together two spatially uncorrelated multiplicity 5 events together as one multiplicity 10 event, but again that is highly unlikely given the occurrence rate of multiplicity 5 or greater events in CUORE.

4.4 Choosing Cuts

Using knowledge of the existing CUORE data storage infrastructure, and energy estimation limitations, we need to determine a set of requirements for which energy pulse events we wish to look at. We will limit the scope of our analysis to those which pass a certain set of cuts. Because we rely upon the existing CUORE pulse shape analysis software and its limitations, it is natural to apply both a low energy cut and a high energy saturation limit. The low energy cut is applied at 500keV assuming non-detection for energies lower than that amount. A high energy saturation cut is applied at 10MeV assuming that any energy measured to be more than 10MeV should be rounded down to be 10MeV. We also applied a minimum multiplicity cut of 10 crystals, knowing that we are potentially throwing away about 5% of our ppp-decay signal, but significantly reducing contamination from non-Muon background sources in the data. Additionally we apply some data quality requirements on single crystal event extraction from the database: 'NoHeaterInWindow'=True 'RejectBadIntervals'=True, 'IsSignal'=True, and we insist that the energy value extracted from the database is not a 'NaN' value.

CUORE does contain what we denote as 'dead channels'. A 'dead channel' is what we denote any of the 988 CUORE crystals which is not recording data. There are 4 channels which are permanently dead. Also, at various random times, a small subset of channels may be flagged by a CUORE shifter as a 'Bad Interval' for one reason or another. While we reject the data from bad intervals in the extraction of the real data, we do not account for the same effect in simulations. In this analysis, the effect of dead channels is assumed to be negligible in the real data as compared to the simulations with regards to feature calculation, and count rate calculation. To properly account for the effect in simulations, one would have to assign each simulated event a random timestamp within the duration that real data was recorded, and then ignore any simulated energy deposits within the dead channels at the time for that particular event.

4.5 Real Data Features : An Emergent Discrepancy

We now have a set of cuts, a multiple crystal grouping method, and a set of features we can utilize. In lieu of a true bias-free blinded analysis, we planned to use a small portion of real data to inform our extraction procedure, then scale up to a full analysis on all the data after our analysis pipeline was built upon the small portion. CUORE's data are already broken into data-sets so we originally planned to use the first finalized dataset which is labeled 3601, and accounts for approximately 30 days of exposure time. We extracted dataset 3601 and applyed our energy cut from 500keV to 10MeV and applied the multiplicity cut of 10 or greater.

An unexpected discrepency emerged, and we found that that the real dataset 3601 did not match the Muon simulations, when looking at their features. A features comparison plot can be seen in Figure 4.1. A few of the feature histograms did not match between muon simulation and real data. The total energy of a grouped real event had a tendency to have more low energy events than we expected. The normalized standard deviation in single crystal energies seemed to be shifted down from where we expected (see 3 for description).

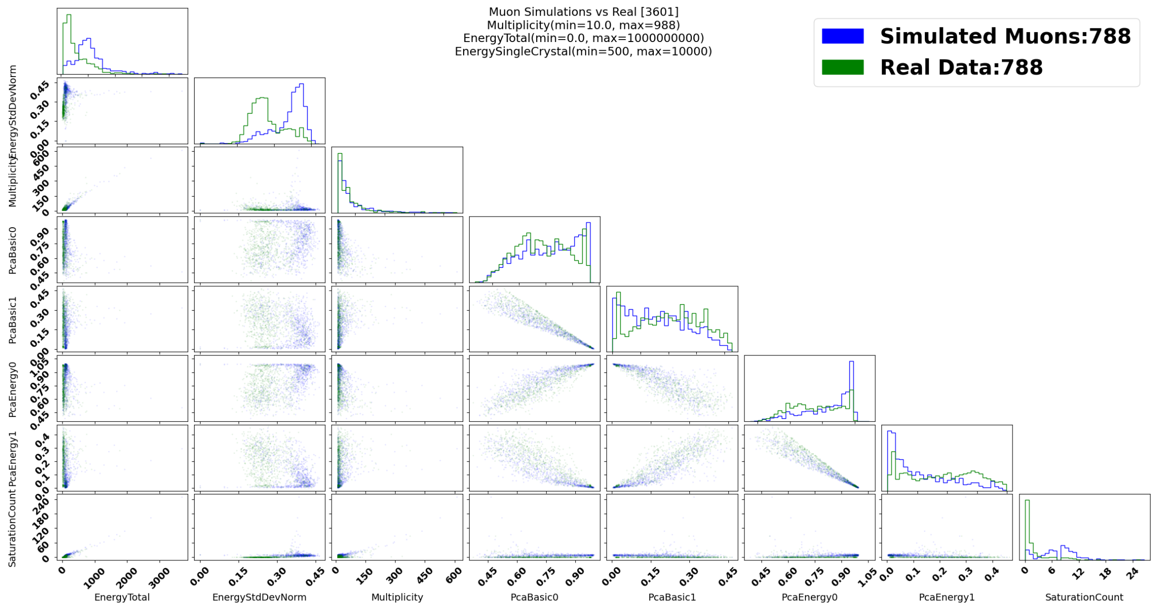


Figure 4.1 Features corner plot for Real Dataset 3601 (green), compared to Muon simulations (blue). The diagonal plots are histograms for each feature. The off-diagonals are each scatter plots corresponding to 2 features. Note the discrepancy between the two left-most features (Total Energy, and Normalized Energy Std-Deviation).

Intuitively we can see that the emergent discrepancies in Figure 4.1 would not be fixed with inclusion with ppp-decay spectra (Figs 3.3 3.4). We needed to investigate the problem more deeply. After discussion within the CUORE physics board, we made the decision to unblind the rest of the real data as compared to muon simulations with respect to their features. At this point we still had not used the real data to perform the rest of our analysis pipeline and obtain half-life result for ppp-decay, but we needed to see if the feature discrepancy was dataset dependent.

We found that the feature discrepancy was indeed dataset dependent. We looked closely at the same features plot for each of the 16 datasets published within CUORE: Datasets 3601 through 3615. Datasets 3601 through 3611 do exhibit a disagreement between muon simulations and real observations and all have feature plots similar to Figure 4.1. Datasets 3612 through 3615 do not exhibit a disagreement between muon simulations and real observations and all have feature plots similar to Figure 4.2. It seems not only is there a dependence upon datasets, but there also exists time of transition from disagreement, to agreement. The time of transition and start date for dataset 3612 is: Mar 12, 2020.

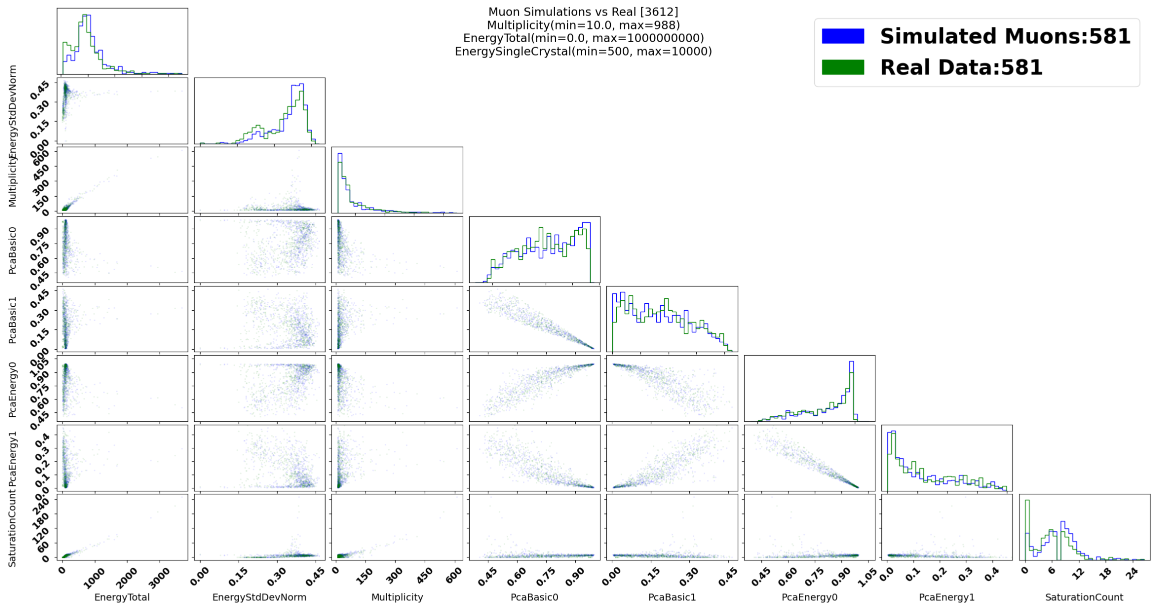


Figure 4.2 Features corner plot for Real Dataset 3612 (green), compared to Muon simulations (blue). The diagonal plots are histograms for each feature. The off-diagonals are each scatter plots corresponding to 2 features. Note there is no discrepancy between the two left-most features (Total Energy, and Normalized Energy Std-Deviation).

With regard to a search for ppp-decay we will simply use data-sets [3612, 3613, 3614, 3615] to obtain our final result. We chose cuts based on knowledge of previous CUORE analysis. We chose features manually that could illustrate differences between high multiplicity events. We viewed the features on dataset 3612, and saw that the data-set's features seem to be in agreement with muon simulations. With regard to a search for ppp-decay the reader can continue to the next section knowing that we we have achieved sufficient pre-requisites to perform an analysis.

4.6 Investigation of the Mar 2020 Real Data Discrepancy

While irrelevant for the search for ppp-decay, here we will begin an investigation into the discrepancy between high multiplicity CUORE data taken before and after a transition time of Mar of 2020. The discrepancy may have more broad importance to the collaboration, and to other physics experiments in LNGS than to the rest of this thesis.

In March of 2020, the Italian COVID-19 lockdown began. Almost all human activity in, and around the Grand Sasso National Laboratory (LNGS) ceased during the lockdown. It is possible that the transition time and the start of full Italian COVID lockdown is a coincidence. It is also possible that the timing is not a coincidence, and other experiments could benefit from studying the difference between their recorded data during the lock down.

To study the timing, we have developed two initial types of plots which may shed light on possible human activity data contamination at LNGS for high-multiplicity CUORE data. One type of plot shows events over time with a histogram of 30 minute bins (Figures 4.3, 4.4, 4.5, 4.6). Another type of plot shows the number of events during each hour of the day (Figures 4.7, 4.8). Both kinds of plots show clear differences in the CUORE data before and after Mar 2020.

Of specific note are the visual differences between dataset 3612, and 3601 on the time histogram. The number of counts in a random 30 minute bin should observe some relationship to the Poisson distribution (Appendix B, Sec 5). With an expected count rate of about 1 Muon per hour, we can see that dataset 3601 once observes 60 Muon-like events in a single 30 minute window. One can infer that the likelihood of this happening is well outside a 5 σ bound.

There must be some contamination in data-sets 3601 through 3611, that is not present in data-sets 3612 through 3615 (while creeping back into one 30 minute bin of dataset 3615). Given the time-correlation, it is possible that the contamination is

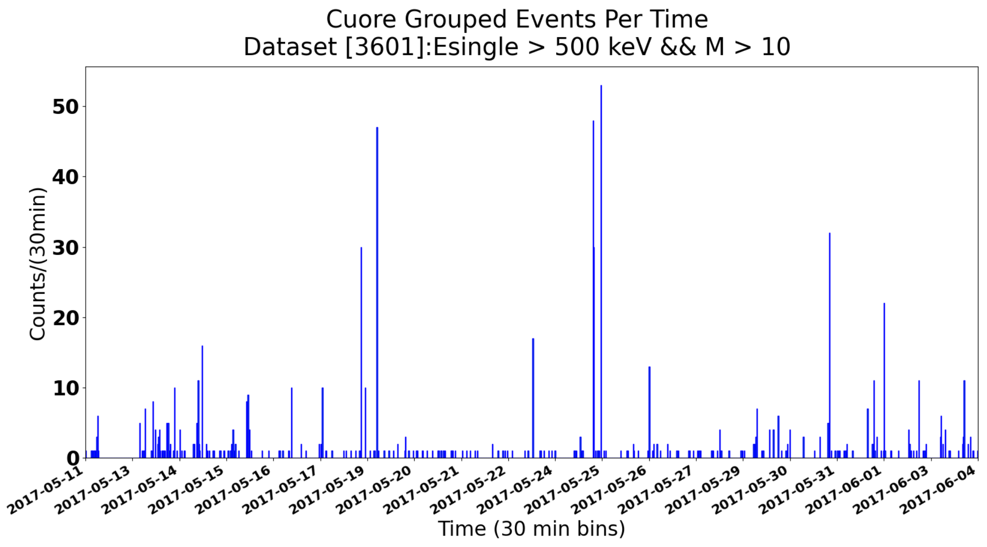


Figure 4.3 Dataset 3601. [M > 10, 500keV < E < 10MeV] Events are shown as a histogram of 30 minute bins allowing time to progress along the x-axis.

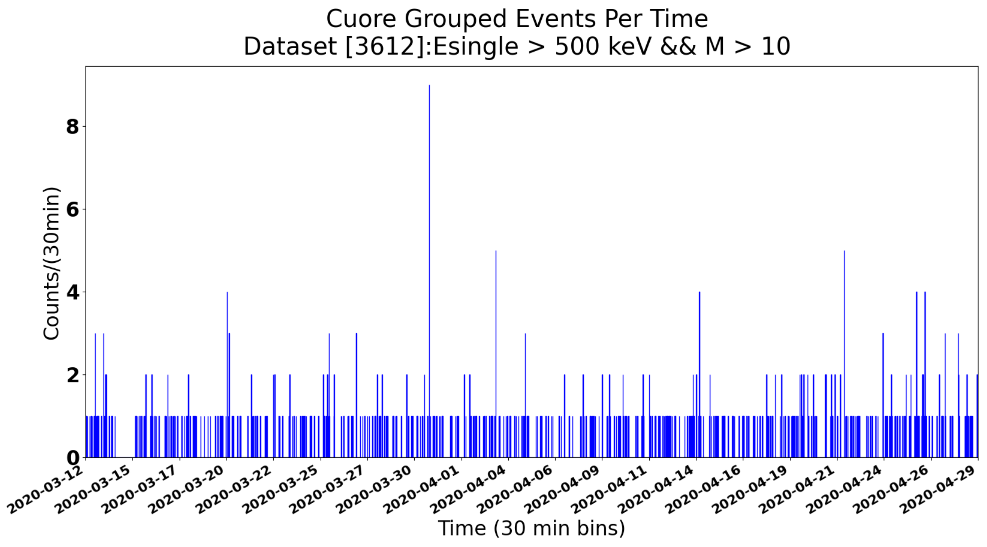


Figure 4.4 Dataset 3612. [M > 10, 500keV < E < 10MeV] Events are shown as a histogram of 30 minute bins allowing time to progress along the x-axis.

due to human activity, which ceased during Covid lockdown. Further investigation of the issue is required, and is actively being pursued by the CUORE collaboration.

If CUORE finds that human activity was responsible for data contamination before covid lockdown, and was almost completely avoided during March 2020 covid lockdown in Italy, we may find that other experiments can use the information to

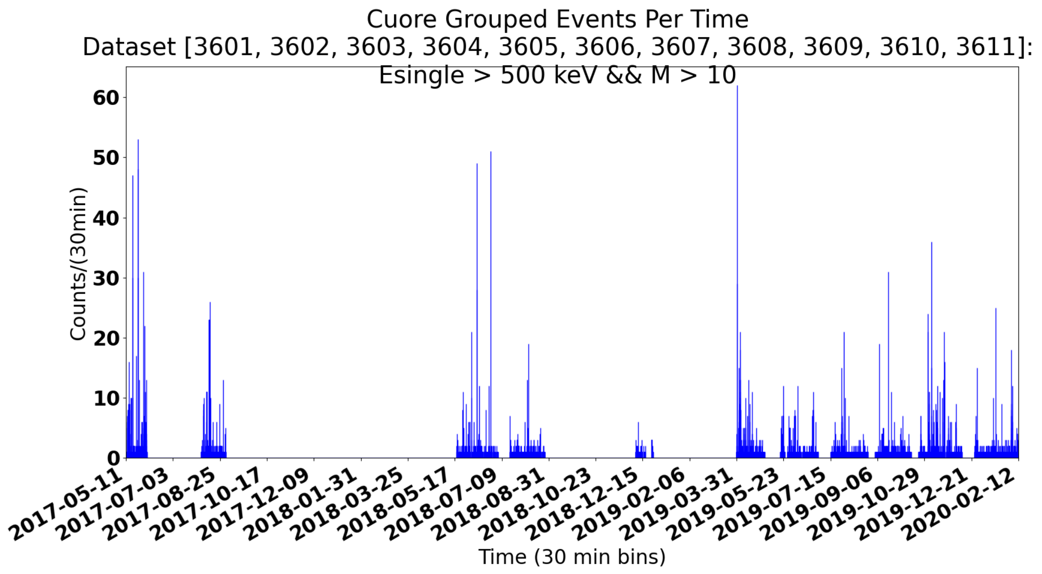


Figure 4.5 Events seen by CUORE before Mar 2020 with [M > 10, 500keV < E < 10MeV]. Events are shown as a histogram of 30 minute bins allowing time to progress along the x-axis.

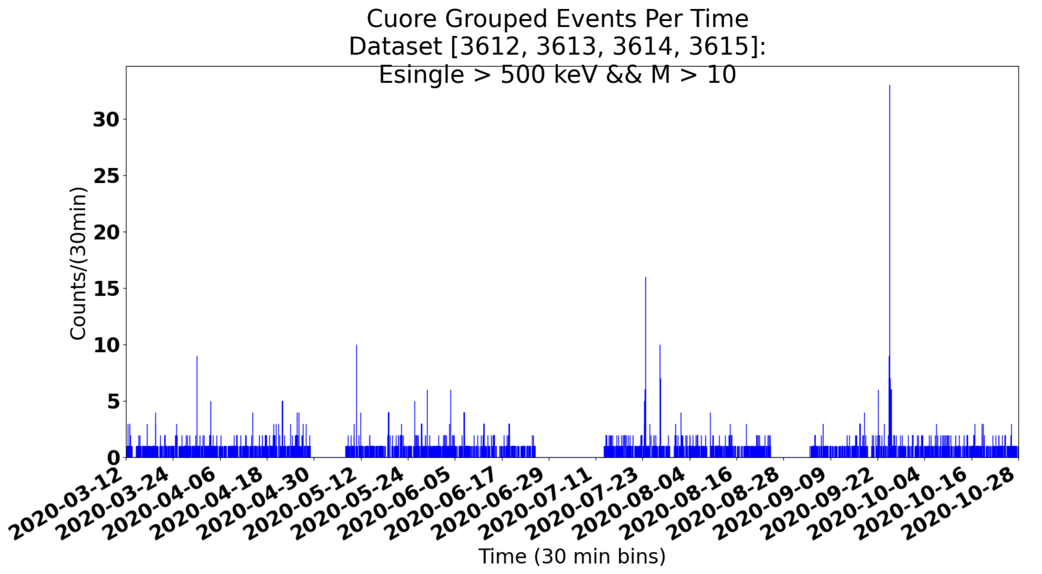


Figure 4.6 Events seen by CUORE after Mar 2020 with [M > 10, 500keV < E < 10MeV]. Events are shown as a histogram of 30 minute bins allowing time to progress along the x-axis.

further clean their data as well, and obtain stronger results on unrelated physics.

There are two known nearby sources of human activity. One source is the road traffic of cars and trucks driving through the tunnel by the laboratory. Another source

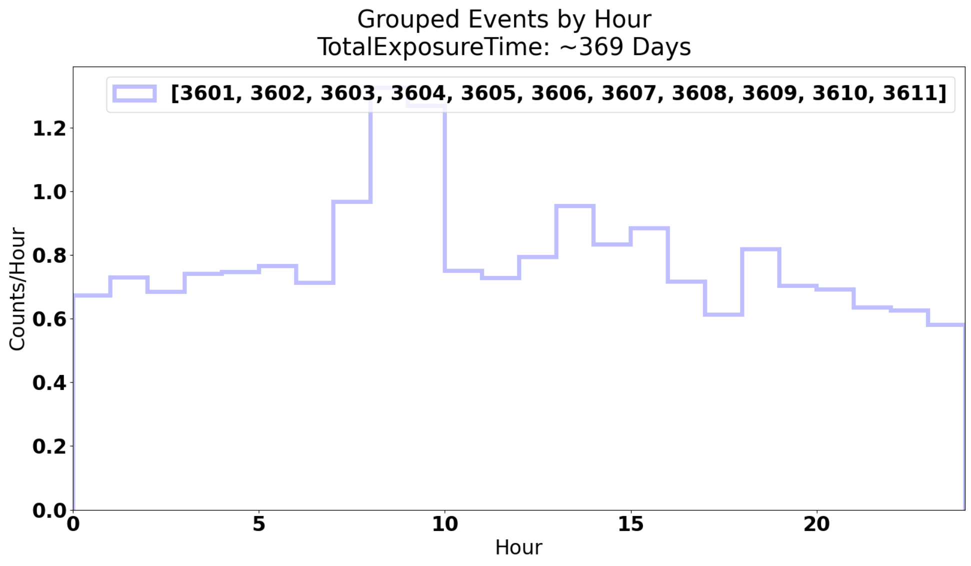


Figure 4.7 Events seen by CUORE before Mar 2020 with [M > 10, 500keV < E < 10MeV]. Events are shown by hour of the day.

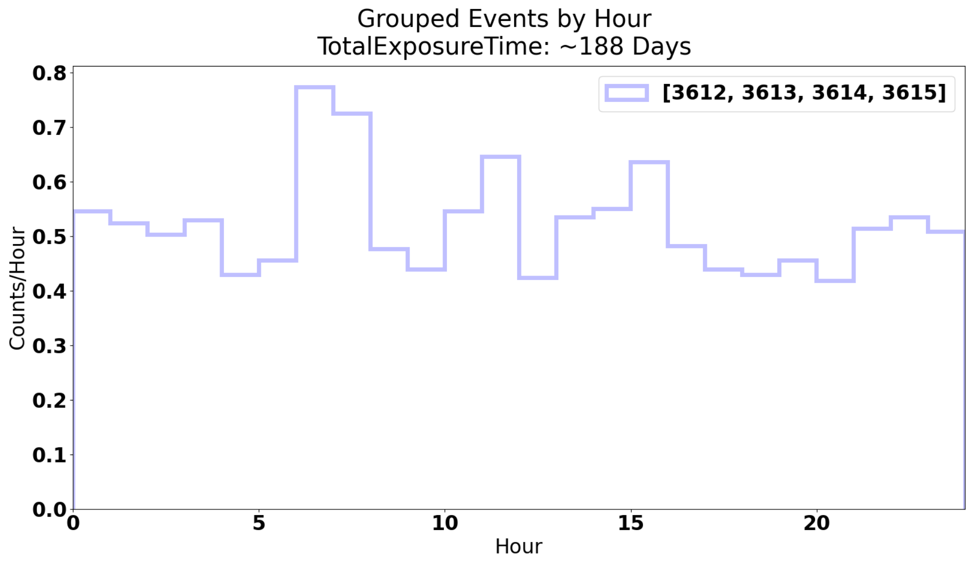


Figure 4.8 Events seen by CUORE after Mar 2020 with [M > 10, 500keV < E < 10MeV]. Events are shown by hour of the day.

is experiment maintenance related work performed by people inside the laboratory cave. CUORE personnel are looking into accessing already existing human activity data sources, as well as potentially creating noise on purpose.

There are some existing data sources which can be used to narrow down the source of contamination. There exists a road traffic camera dataset for traffic through the

LNGS highway tunnel. Separately there may exist a foot traffic dataset, with entry and exit log information for humans performing active work within the lab as a whole, or nearby detector huts.

An alternative, easier to implement, and more direct approach to finding the source may be to actively create the noise on purpose. We could commission a large heavy truck to drive back and forth through the tunnel. We can log the exact movement over time for the truck, including starting, stopping, accelerating etc. We can also have people walk into the Grand Sasso Laboratory and simulate various kinds of manual labor performed by other scientists in the lab.

The CUORE team is beginning an investigation into this time-based high multiplicity event count rate discrepancy. Both active and passive analysis methods are being considered. Understanding how non-particle-physics based contamination might effect the CUORE detector could have an important impact for how data analysis is performed within underground cryogenic experiments.

Chapter 5

Maximum Entropy to Counting Statistics

5.1 Entropy Definition

First lets introduce the concept of entropy in the context of probability theory, using precise language:

$$\begin{aligned} \boxed{X}\ &\text{is some random variable we can measure} \\ x\ &\text{is an instance of measuring the random variable } \boxed{X} \\ \boxed{X}\ &\text{has a probability density function: } \boxed{X}_{pdf}(x) \\ \boxed{X}_{pdf}\ &\text{has an entropy} \end{aligned} \tag{5.1}$$

The entropy of a density function is defined as follows:

$$Entropy\left[\boxed{X}_{pdf}(x)\right] = -\int_{-\infty}^{\infty} \boxed{X}_{pdf}(x)log(\boxed{X}_{pdf}(x))dx \tag{5.2}$$

The entropy is a functional of a function which returns a single number. One can choose a distribution which maximizes the entropy, while satisfying any limiting constraints. Any constraints placed upon the distribution are considered "information". An axiom in information theory states that the maximum entropy distribution, is the best guess we can have at the underlying true distribution without data, and only starting with constraints. This can be written:

$$\begin{gathered} \boxed{X}_{pdf,best}(x) = Max\left[Entropy\left[\boxed{X}_{pdf}(x)\right]\right] \\ \text{...subject to any constraints} \end{gathered} \tag{5.3}$$

The mathematics of maximizing functionals such as entropy is beyond the scope of this thesis. Of specific interest to the reader may be studying calculus of variations, and Lagrange multipliers. In practice, there are only a handful of constraints physicists might place upon a distribution and the resulting maximum entropy solutions can be found in existing literature without requiring novel derivation.

5.2 Lagrange Multipliers

Lagrange multipliers are useful tools if we have an arbitrary function $f(x)$ we need to maximize subject to a constraint $g(x) = c$. The Lagrangian is then defined as a new function which includes the original function f as well as the contraint g, and introduces a new parameter λ_{lag} which we call the "lagrange multiplier":

$$\mathcal{L} = f(x) - \lambda_{lag}(g(x) - c) \tag{5.4}$$

To find the maximum of f subject to the constraint g, we can instead find he global maximum of $\mathcal{L}$ by solving the system of equations defined by:

$$\nabla_{x,\lambda_{lag}} \mathcal{L} = 0 \tag{5.5}$$

Which is a system of two equations and two unknowns. In general the Lagrange multiplier result is always a critical point but not always an extrema. When solving general problems one has to worry about finding saddle points with the method or checking boundaries as part of the constraints. The "Karush–Kuhn–Tucker conditions" determine whether or not saddle points and boundaries need to be considered. Fortunately, in the context of Maximum Entropy, it turns out that one will always obtain a maximum, and saddle points and boundary points need not be considered.

5.3 Entropy Derivation of the Exponential Distribution

Here we will derive a density function from a set of constraints using the maximum entropy principle. We will choose an example which is both simple, and yields a result that is useful to us later. We will use Lagrange Multipliers to maximize the entropy functional.

Let $\boxed{T}$ be the random variable of interest, which is constrained to be positive. Let us also state that the expected value of $\boxed{T}$ is known: $E\Big[\boxed{T}\Big] = \lambda_t$. We can now ask what is probability density function $\boxed{T}_{pdf}(t)$ which maximizes the entropy subject to the constraints? We will show that the result is the negative exponential distribution (Eqn B.1). The constraint of positivity $\boxed{X} > 0$ can be included by limiting integration from 0 to ∞.

The first constraint is that probabilities must add or integrate to unity:

$$1 = \int_0^\infty dt \boxed{T}_{pdf}(t) \tag{5.6}$$

The constraint of expectation $E\Big[\boxed{T}\Big] = \lambda_t$ can be written in equation form:

$$\lambda_t = \int_0^\infty dt \Big(t \boxed{T}_{pdf}(t) \Big) \tag{5.7}$$

The thing we want to maximize is the entropy (as seen in Eqn 5.2)

$$Entropy = -\int_0^\infty dt \boxed{T}_{pdf}(t) log(\boxed{T}_{pdf}(t)) \tag{5.8}$$

Our Lagrangian can then be defined by starting with entropy and subtracting additional terms for each constraint, with additional constants defined as Lagrange multipliers λ_j:

$$\begin{aligned} J \equiv & \int_0^\infty dt \boxed{T}_{pdf}(t) log(\boxed{T}_{pdf}(t)) \\ & - \lambda_{lag,0}(1 - \int_0^\infty dt \boxed{T}_{pdf}(x)) \\ & - \lambda_{lag,1}(\lambda_t - \int_0^\infty dt \Big(t \boxed{T}_{pdf}(t) \Big)) \end{aligned} \tag{5.9}$$

From our knowledge of Lagrange multipliers, we know we need to take derivatives and set them equal to zero. However with simplistic Lagrange multipliers we are supposed to take the gradient and set it equal to zero. Instead of taking the gradient we can use a trick taken from calculus of variations. We can instead take the functional derivative and set it equal to zero. This is only possible because each term in our defined Lagrangian J contains an integral over our function of interest $\boxed{T}_{pdf}$. Choosing to take the functional derivative instead of taking the gradient is not obvious and is done with knowledge of the calculus of variations.

$$\begin{aligned} 0 &= \frac{dJ}{d\boxed{T}_{pdf}} \\ 0 &= ln(\boxed{T}_{pdf}) + 1 - \lambda_{lag,0} - \lambda_{lag,1}t \end{aligned} \tag{5.10}$$

We can then solve for the form of $\boxed{T}_{pdf}$ in terms of $\lambda_{lag,0}$ and $\lambda_{lag,1}$:

$$\begin{aligned} \boxed{T}_{pdf}(t) &= e^{1-\lambda_{lag,0}-\lambda_{lag,1}t} \\ &= e^{1-\lambda_{lag,0}}e^{-\lambda_{lag,1}t} \end{aligned} \tag{5.11}$$

From here we need only to actually solve for the constants $\lambda_{lag,i}$. To solve for them, we can use each constraint, which will produce additional equations equal to the number of unknowns. We start with the first constraint of probabilities integrating to unity:

$$\begin{aligned} 1 &= \int_0^\infty dte^{1-\lambda_{lag,0}}e^{-\lambda_{lag,1}t} \\ &= e^{1-\lambda_{lag,0}} \int_0^\infty dte^{-\lambda_{lag,1}t} \\ &= e^{1-\lambda_{lag,0}} \frac{1}{\lambda_{lag,1}} \end{aligned} \tag{5.12}$$

Now we satisfy the second constraint of known expectation:

$$\begin{aligned} \lambda_t &= \int_0^\infty dt\Big(t\ e^{1-\lambda_{lag,0}}e^{-\lambda_{lag,1}t}\Big) \\ &= e^{1-\lambda_{lag,0}} \int_0^\infty dt\Big(t\ e^{-\lambda_{lag,1}t}\Big) \\ &= e^{1-\lambda_{lag,0}} \frac{1}{\lambda_{lag,1}^2} \end{aligned} \tag{5.13}$$

Substituting the first constraint equation into the second solves for λ_1:

$$\lambda_t = \frac{1}{\lambda_{lag,1}} \tag{5.14}$$

Substituting into the first constraint equation solves for the constant out front:

$$e^{1-\lambda_{lag,0}} = \lambda_{lag,1} = \frac{1}{\lambda_t} \tag{5.15}$$

Substituting it into our final form of the density function of $\boxed{T}_{pdf}$:

$$\begin{aligned}\boxed{T}_{pdf}(t) &= e^{1-\lambda_{lag,0}}e^{-\lambda_1 t}\\ &= \frac{1}{\lambda_t}e^{-\frac{1}{\lambda_t}t}\end{aligned} \tag{5.16}$$

We have obtained the negative exponential distribution as expected. We obtained the distribution using the maximum entropy princple. We chose the probability density function out of all possible choices, which maximized the entropy, and still satisfied the constraints we needed to satisfy. Note that this negative exponential distribution is not just a good choice given the constraints. It is the only best choice given those constraints. In a very real way, we have proven that using the negative exponential distribution to model when some future event happens, is not only a reasonable thing to do, it is the correct thing to do.

5.4 Detectors Underground: Counting Experiments

In experimental physics we often have a detector sitting underground or inside a mountain and we want to model the rate of decay. Let us assume that we have some collection of n particles which will each decay eventually at times $[t_0, t_1, ...t_{n-1}] = \overrightarrow{t}$. Let us also assume that we have information that the expected amount of time for a single particle to decay is λ_t. Lets also assume that the particle will decay in the future so $t > 0$.

We just showed that such a set of constraints will result in the negative exponential distribution as the best possible model for the single particle's decay time:

$$\begin{aligned}\text{Information Constraint} : E[t] &= \lambda_t\\ \text{Probability Density} : \boxed{T}_{pdf}(t) &= \frac{1}{\lambda_t}e^{-\frac{t}{\lambda_t}}\end{aligned} \tag{5.17}$$

Since the events we are observing are independent but share the same underlying expected $E[t] = \lambda_t$ we can easily write the joint probability of observing them all at the times when we do observe them:

$$Prob(\text{See events at times } \overrightarrow{t}|\lambda_t) = \prod_i \frac{1}{\lambda_t} e^{-\frac{t_i}{\lambda_t}} = \frac{1}{\lambda_t^n} e^{-\frac{1}{\lambda_t}\sum_i t_i} \tag{5.18}$$

Note we could actually use this joint likelihood, substituting in the times we did see real events, to get a final result. However, it would not be computationally efficient, or intuitive to do so. Instead, standard practice is to perform counting statistics as shown in the next subsection.

5.4.1 Single Events To Counting Statistics (Poisson)

Instead of modeling a joint distribution for individual events, it is more useful to count how many events we have seen. We can derive the Poisson distribution to model such counting with the maximum entropy principle.

$$Prob(\text{See } k \text{ events at any times } \overrightarrow{t}|\lambda_t) = Poisson \tag{5.19}$$

To obtain the Poisson distribution, we can use the negative exponential distribution which we derived earlier. We must define a time window, t_{start}, t_{end}. There exist some collection of N particles and each is independently capable of some decay process that can happen but is very unlikely to actually happen. (N atoms which can decay).

We want to know: From time 0 to time t_{end} what is the probability that we observe k events? If we define our time window to start at zero then t_{end} is conveniently also the duration of the time window. We can start by asking what the probability is that we see the first single atom decay. If we label each atom in the collection as "atom number i", then the first one has the label: "atom number 1". We know the first atom's decay will happen eventually. We also know the first atom's decay has a certain probability of occurring during the time window of interest. What is that probability? We can calculate it using the negative exponential distribution we derived previously:

$$
\begin{aligned}
Prob(\text{See single event}_i \text{ in window}) &= \int_0^{t_{end}} dt_i \frac{1}{\lambda_t} e^{-\frac{1}{\lambda_t} t_i} \\
&= 1 - e^{t_{end}/\lambda_t} \\
&\equiv P_i
\end{aligned} \tag{5.20}
$$

We could also ask: What is the probability we see both atom number 1 and atom number 2? (within the allowed time window)

$$Prob(\text{See single event}_1 \text{ and event}_2 \text{ in window}) = P_1 P_2 = (P_i)^2 \tag{5.21}$$

We know that the atoms are identical and independent. Thus within the time window the probability of seeing any single atom decay is equal to that of any other atom. What is the probability we see any k events given that there were N to start with? We can use the binomial distribution with successes and failures. Each event has probability of success P_i. Each event has probability of failure $1 - P_i$. Using the binomial distribution we can find the odds of seeing exactly k successes:

$$Prob(k, n, p = P_i) = \binom{n}{k} P_i^k (1 - P_i)^{n-k} \tag{5.22}$$

Note the expected count λ_{count} is equal to the number of total atoms n times the probability of a single atom decaying P_i. We could prove this by careful calculation using the definition of expectation over the discrete probability mass function defined by the binomial distribution above. Instead we will state the intuitive relationship between λ_{count} and λ_t.

$$
\begin{aligned}
\lambda_{count} &\equiv \lambda_c \equiv nP_i \\
\lambda_{count} &= n(1 - e^{t_{end}/\lambda_t})
\end{aligned} \tag{5.23}
$$

5.5 Counting Statistics: Approximations for Computation

With the binomial distribution we have a probabilistic counting model which is correct. However, calculating factorials of large numbers is not computationally feasible.

We need tricks to reduce the computational cost but still get a good approximation to the true binomial result. In the limit of large n the binomial distribution becomes Poisson, and in the limit of small p the Poisson becomes Gaussian. These two derivations are included for reference in section E.

First lets use the Poisson limit. The limit is that $n \to \infty$, $p \to 0$ and $np \to \lambda_{count}$ which is a constant. The Poisson limit can also be interpreted as assuming replacement of any atoms that decayed.

Essentially as the number of events which can happen (n), goes to infinity, then the Poisson distribution becomes valid and any events that do happen do not significantly reduce the pile of atoms we started with.

$$\begin{aligned} Prob(k, n = \text{BIG}, p = \text{SMALL}) &= Binomial(k, n, p) \\ &= \binom{n}{k} P_i^k (1 - P_i)^{n-k} \\ &\approx Poisson(k, \lambda_{count} = np) \\ &= \frac{\lambda_c^k e^{-\lambda_c}}{k!} \end{aligned} \tag{5.24}$$

Next let us use the Gaussian limit. The limit is that $n \to \infty$ and λ_{count} is large. The resulting Gaussian has $\mu \approx \lambda_{count}$ and $\sigma \approx \sqrt{\lambda_{count}}$. The Gaussian limit can be interpreted as a Poisson distribution for the case where the mean is well above zero. The Poisson becomes particularly non-Gaussian in the limit that the number of expected events is very low ($\lambda_c < 20$). As the mean shifts upwards, and the probability of Gaussian negative events becomes almost zero, the approximation becomes very good. In practice we can use a Gaussian approximation in the limit that λ_{count} is greater than 100.

$$\begin{aligned}
Prob(k, n = \text{BIG}, p = \text{SMALL}) &\approx Poisson(k, \lambda_{count} = np) \\
&= \frac{\lambda_c^k e^{-\lambda_c}}{k!} \\
&\approx Gaussian(k, \mu = np, \sigma = \sqrt{np}) \\
&= \frac{1}{\sqrt{2\pi np}} e^{-\frac{1}{2}\left(\frac{k-np}{\sqrt{np}}\right)^2}
\end{aligned} \tag{5.25}$$

We have made statements about what regions where the above approximations are "good" but we have not made quantitative claims to support the "goodness". Proving convergence and accepted tolerance on the convergence is a level of extra work that could be done. However, we can temper our fears about the closeness of approximation by knowing that the error rate on our counting, and mistakes made when performing classification in later sections, will far outweigh any small error on the approximation of the binomial distributing from using a Poisson or Gaussian distribution. Thus making precise statements on how "big" or "small" parameters lead to "good" quantitative results is unnecessary.

5.5.1 Concrete examples of Binomial Approximations

Assume there is a detector which only has 100 atoms inside. Let us decide to count the number of times within a 1-year window, that atoms decay. We know the probably of any particular atom decaying within the time window is P_i. It turns out we observe exactly 2 decays. Thus $k = 2, n = 100$, $p = P_i$. What is the probability of such an experiment occurring?

$$\begin{aligned}
Prob(k = 2, n = 100, p = P_i) &= Poisson_{pdf}(\lambda_{count} = nP_i, k) \\
&\equiv \frac{(nP_i)^k e^{-(nP_i)}}{k!} \\
&= \frac{(100P_i)^2 e^{-(100P_i)}}{2!}
\end{aligned} \tag{5.26}$$

Now let us do an example where the Gaussian approximation is valid. Let $n = 10^7$ and $k = 100$ and in advance we know know the probably of a single atom decaying within the time window. We also know that the expected number of observed events is knowing $\lambda_{count} = nP_i$. What is the probability of observing those 100 events in terms of P_i ?

$$\begin{aligned}
Prob(k = 100, n = 10^6, p = P_i) &= Poisson_{pdf}(\lambda_{count} = nP_i, k) \\
&\approx Gaussian_{pdf}(x = k = 100, \mu = \lambda_{count}, \sigma^2 = \lambda_{count}) \\
&= Gaussian_{pdf}(k = 100, \mu = nP_i, \sigma^2 = nP_i) \\
&= \frac{1}{\sqrt{2\pi\lambda_{count}}} e^{-\frac{1}{2}\left(\frac{k-\lambda_{count}}{\sqrt{\lambda_{count}}}\right)^2} \\
&= \frac{1}{\sqrt{2\pi 10^7 P_i}} e^{-\frac{1}{2}\left(\frac{100-10^7 P_i}{\sqrt{10^7 P_i}}\right)^2}
\end{aligned} \tag{5.27}$$

5.6 Summary of Relationships Between Expectations

In this section we will try to bring a few different distributions and their expectations together. In addition to the expected lifetime of a single particle λ_t, and the expected total count within a time window λ_{count}, it is also common to consider the decay rate λ_{decay}, and the half-life λ_{half}. Because all these concepts are defined by a single decay process in nature, we can view each of these quantities as different versions of the same information. Here we will provide relationships between all the λ's of interest.

First, lets use the information we have in order to represent the expected number of particles remaining as a function of the half-life. If there were no statistical fluctuations involved, and we could have fractional particles, we could represent the number of remaining particles as a renormalized multiple of the exponential decay probability distribution. We need to make a new copy of it so that there are 1 particles at time $t = 0$ and zero particles at time $t \to \infty$. Such a renormalization is easy and can be done by multiplying the negative exponential distributions by the expected lifetime λ_t. Finally we need to multiply by the number of total particles in the ensemble, because we have more than a single particle.

$$\begin{aligned} n(t) &= n\lambda_t \frac{1}{\lambda_t} e^{-\frac{1}{\lambda_t}t} \\ &= ne^{-\frac{1}{\lambda_t}t} \end{aligned} \tag{5.28}$$

For practical purposes, it is convenient to define a half-life such that the number of expected remaining atoms that exist at the time of half-life will be exactly $N/2$. Using the definition of half-life we can reformulate the same number of time function in terms of half-life:

$$n(t) = ne^{-\frac{ln(2)}{\lambda_{half}}t} \tag{5.29}$$

From knowing the total number of particles we have as a function of time, we can also find the rate of change of the number of particles over time by taking the derivative of the function. For very long half-life's, the rate of change over the time window barely changes, so we can denote the derivative as a constant at the time $t = 0$, with λ_{decay}.

$$\begin{aligned} \frac{dn}{dt} &= \text{rate of change of the ensemble} \\ \frac{dn}{dt} &= -n\frac{1}{\lambda_t}e^{-\frac{1}{\lambda_t}t} \\ \lambda_{decay} &= \left.\frac{dn}{dt}\right|_{t=0} = -n\frac{1}{\lambda_t} \end{aligned} \tag{5.30}$$

Finally we can put all the information together to relate all the λ's. We need to use the relationship between the negative exponential distribution and the Poisson distribution (Eqn: 5.20). We need to use definition of half-life and the formulation of the negative exponential distribution. And we need to use the rate of change function. The relationships between all the non-time λ's can be seen in three separate functions of λ_t:

$$\lambda_{count} = n(1 - e^{t_{end}/\lambda_t}) \tag{5.31}$$

$$\lambda_{decay} = -n\frac{1}{\lambda_t} \tag{5.32}$$

$$\lambda_{half} = \lambda_t / ln(2) \tag{5.33}$$

However, in practice for our analysis we want a convenient relationship between the half-life λ_{half} and the total observation count λ_{count}. We are measuring counts, and we want to obtain a half-life. Therefore we need to represent the half-life as a function of counting. We can start by substituting Eqn 5.33 into Eqn 5.31.

$$\lambda_{count} = n(1 - e^{-t_{end}ln(2)/\lambda_{half}}) \tag{5.34}$$

We need to solve the equation for λ_{half} in a few algebraic steps:

$$\begin{aligned} \frac{\lambda_{count}}{n} - 1 &= -e^{-t_{end}ln(2)/\lambda_{half}} \\ ln(1 - \lambda_{count}/n) &= -t_{end}ln(2)/\lambda_{half} \\ \lambda_{half} &= \frac{-t_{end}ln(2)}{ln(1 - \lambda_{count}/n)} \end{aligned} \tag{5.35}$$

Here lies a hidden final step. $1 - \lambda_{count}/n$ is very close to one for large n. Computation of the log of a number very close to one, will lead to rounding errors. In order to handle the rounding error in the conversion between λ_{count} and λ_{half} we will use the first tool in the physicists toolbox, and try a Taylor-Series. In this case it turns out that a taylor series exactly solves the problem. We will approximate $ln(1 + x)$ as a Taylor-Series:

$$ln(1 + x) \approx x - x^2 + x^3 ... \qquad \forall \quad |x| << 1 \tag{5.36}$$

Letting $x = -\lambda_{count}/n$ then our expression for λ_{half} becomes:

$$\lambda_{half} = \frac{-t_{end} ln(2)}{-\lambda_{count}/n)} = \frac{t_{end} ln(2) n}{\lambda_{count}} \tag{5.37}$$

5.7 Usage of the Maximum Entropy Derivation

We started with a maximized entropy result derived from constraints to obtain the negative exponential distribution for the decay time of a single particle. We used the combination of such a negative exponential distributions to derive the Binomial distribution. Then we showed that the binomial distribution becomes either a Poisson distribution or Gaussian distribution in the large number limit. Finally we showed that the expected count within a window can be transformed into a half-life.

We have shown that that using a Poisson distribution to calculating half-life is the best possible model under the condition that the only information available is the expected value of decay time. The derivation will be useful when finding a final likelihood for a counting analysis which includes classification in the later section (sec 8).

Chapter 6

Classification, and other required concepts

Here we will start with an introduction to artificial intelligence (AI), and where supervised classification fits into the big picture of AI. Then we will explain some basic algorithms that can perform supervised classification. Finally we will discuss how to assess the performance of the classification algorithm, so that we can use it in practice in the next section (sec 8).

Here we broadly define artificial intelligence or "machine learning" as tools that attempt to solve computation problems that require some generalization beyond simple "if-then" statements. Machine Learning is subdivided into "instructive-learning" and "evaluative-learning" (also known as reinforcement learning). Instructive learning reduces to having the computer build functions on the fly which map known input to some output. Evaluative learning instead has the computer attempt actions with a trial and error mindset to learn an environment and obtain rewards. Evaluative learning is designed for the computer to handle situations with completely unknown input data, and is a highly active area of new research. Instructive learning is decades old originally, but is only recently (in the last 10 years) becoming popular among the physics community for performing data analysis because of its ease of use, and increased performance. Supervised classification, lies within the "Instructive" learning paradigm, and will be the limited focus of this work.

Classification tools start with a list of items and have the goal to decide which items are similar enough to belong together in a collection. Here we provide short summaries of both **Supervised** and **Unsupervised** classification, however our research is entirely focused on using previously built supervised learning techniques.

Supervised Classification takes a list of predefined sets, and a list of items already classified, and uses that information to train the classifier. The classifier can then be tested against another set of items with known sets for accuracy. When testing the success rate of a classifier, it is very important to separate data into **training validation**, and **testing** datasets (Sec 6.4).

Unsupervised Classification takes a list of items, and decides upon sets to place them within, in a completely automated fashion. An unsupervised system determines which sets should exist, and how many there should be. Instead of only testing accuracy, one also needs to incorporate information about how many sets should exist compared to how many were created by the algorithm. In our work we will not make use of unsupervised learning.

6.1 Algorithm: Support-vector machines

Support-vector machines (SVM) is perhaps one of the earliest, and also most intuitive ways of performing supervised classification. SVM performs supervised classification from data that contains features from exactly two collections. In basic terms, SVM draws an optimal line of separation between two sets of dots on a scatter plot. The line, in its most simple form, must also be a mathematical line. $y = mx+b$. Therefore in the case of linear SVM with 2 sets(or "classes"), and in 2 dimensions, the entirety of support-vector machines reduces to optimizing m and b that best classifies between two sets of known dots. An illustration is shown with green and blue dots in Figure 6.1.

To calculate the optimal hyperplane for two collections of points that lie in N-D space we need to optimize the location of the line to best widen a boundary between the two collections of points (Eqn 6.1). x_i is a single point within the dataset of interest, w is a normal vector to the hyperplane, and b is a constant. The hyperplane

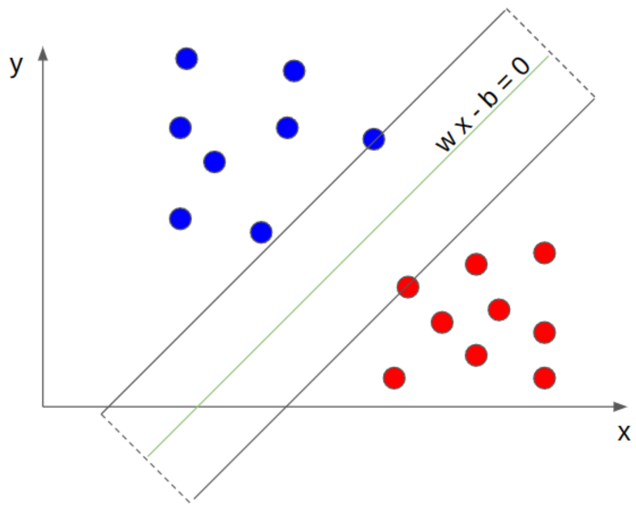


Figure 6.1 Linear Support Vector Machines (SVM) Example Image. This figure's content is toy-data intended to be an abstract illustration. However, for concreteness, the reader may imagine that the blue, and red dots on this figure represent muons and ppp-decay simulations respectively. Here the reader may visualize that only two dimensions are shown (x, y), corresponding to only two features of events in the CUORE detector. If our simulations produced feature values seen on this scatter plot, a line of best separation $wx - b = 0$ and it's parameters w and b would be deduced by the SVM algorithm. The SVM algorithm would then suggest, that any event with features creating a point on the scatter plot above the green-line should be classified as muons, and any event with features below the red-line should be classified as ppp-decays.

satisfies the equation $w^T x - b = 0$. y_i is the Boolean choice assigned to each point in space which in the case of SVM is either 0 or 1, and $\lambda_{classifier} > 0$ is an extra parameter that defines the penalty for points which lie upon the wrong side of the hyper-plane.

$$\lambda_{classifier}||w||^2 + \frac{1}{n}\sum_{i=1} max(0, 1 - y_i(w^T x_i - b)) \tag{6.1}$$

Although ultimatetly not used for our analysis, an extension of support vector machines exists which produces non-linear manifolds of separation.

6.2 Algorithm: Decision Tree and Random Forest

Decision trees are another early adoption method for obtaining supervised classification. Unlike support-vector machines explored above, decision trees easily extend to multiple classes. They are highly intuitive, and are somewhat methodologically contradictory to the basis of our definition of machine learning, because they can be exactly defined by a set of "if-then" statements to obtain a result. Because of their simplicity, they are computationally efficient, and can be presented in human understandable formats. Unfortunately, basic decision trees in practice are not very effective.

Random forests are the next logical extension to decision trees. Instead of a single tree, we can create many. Instead of relying upon the conditions for a single tree to be correct in making a decision, we create many trees with random conditions, then allow each to vote with a prediction (Fig 6.3). With many trees to draw from, and the ability to ask each for a prediction we can get much better accuracy on our ability to classify. There is the drawback however; the more trees that get added, the less interpretable the final result is for humans who want to understand how the choices are made.

When calculating our ppp-decay half-life bound, we rely upon a random forest. We found that the random forest had the best success rates within a reasonable amount of computation time. See sections 6.6.3 and 6.5 for more details.

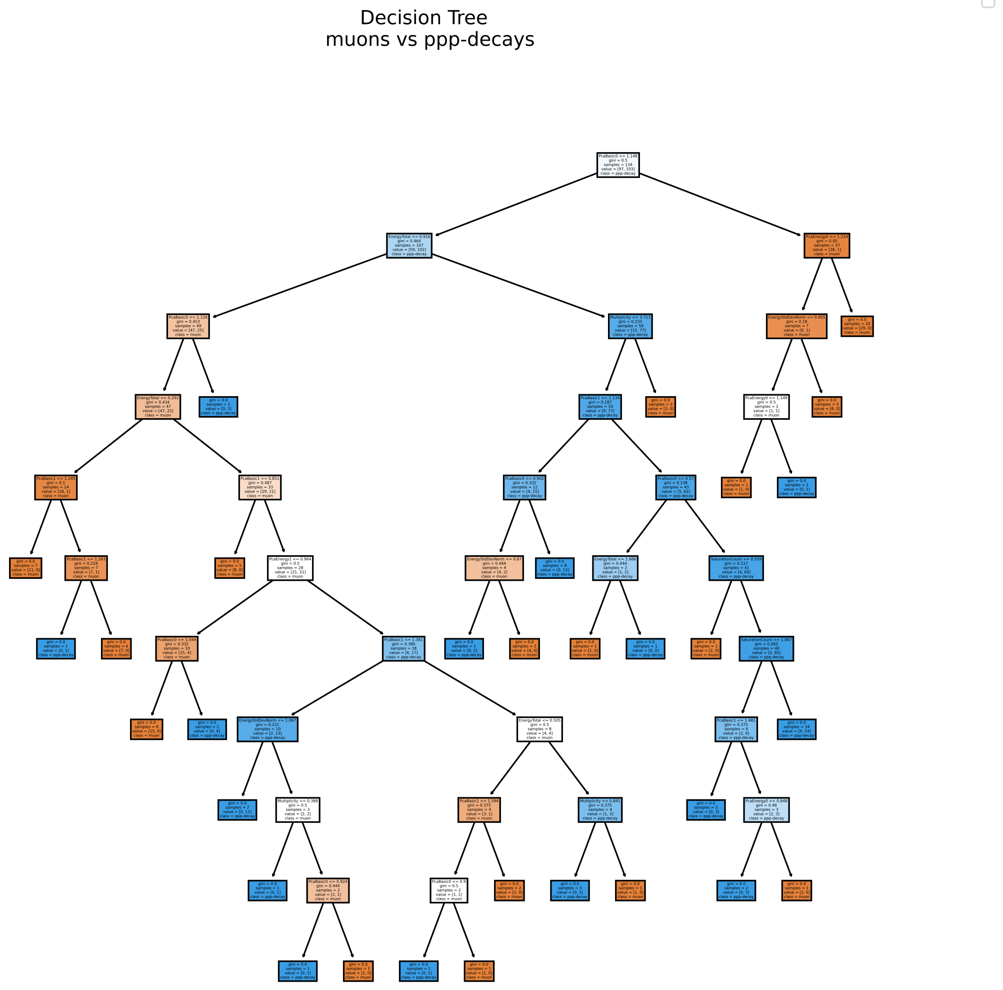


Figure 6.2 Decision Tree Visualization trained and plotted with the "sklearn" python library [9]. This decision tree was generated using 100 of each type of simulation in the CUORE detector. The nodes at the bottom of the tree show when a final decision is reached. Nodes which are more likely to be muons appear orange, while nodes that appear more likely to be ppp-decays are blue. This tree is not very realistic as it was generated with too few simulations. To see a more realistic tree see the one in Fig 6.4. Note there is a trade-off between performance and readability.

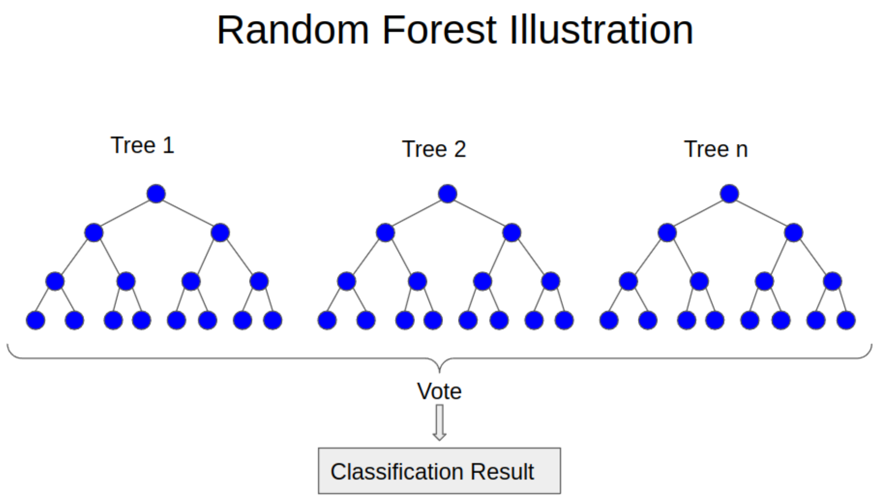


Figure 6.3 Random Forest Visualization. The trees shown in this image are each to be thought of as instances of the larger fully shown tree in Figures 6.2 6.4. Many such trees are made independently. After all the trees are constructed, they can be used to classify an event. Each tree casts a vote upon a single event, and the voting majority is used to choose the final classification.

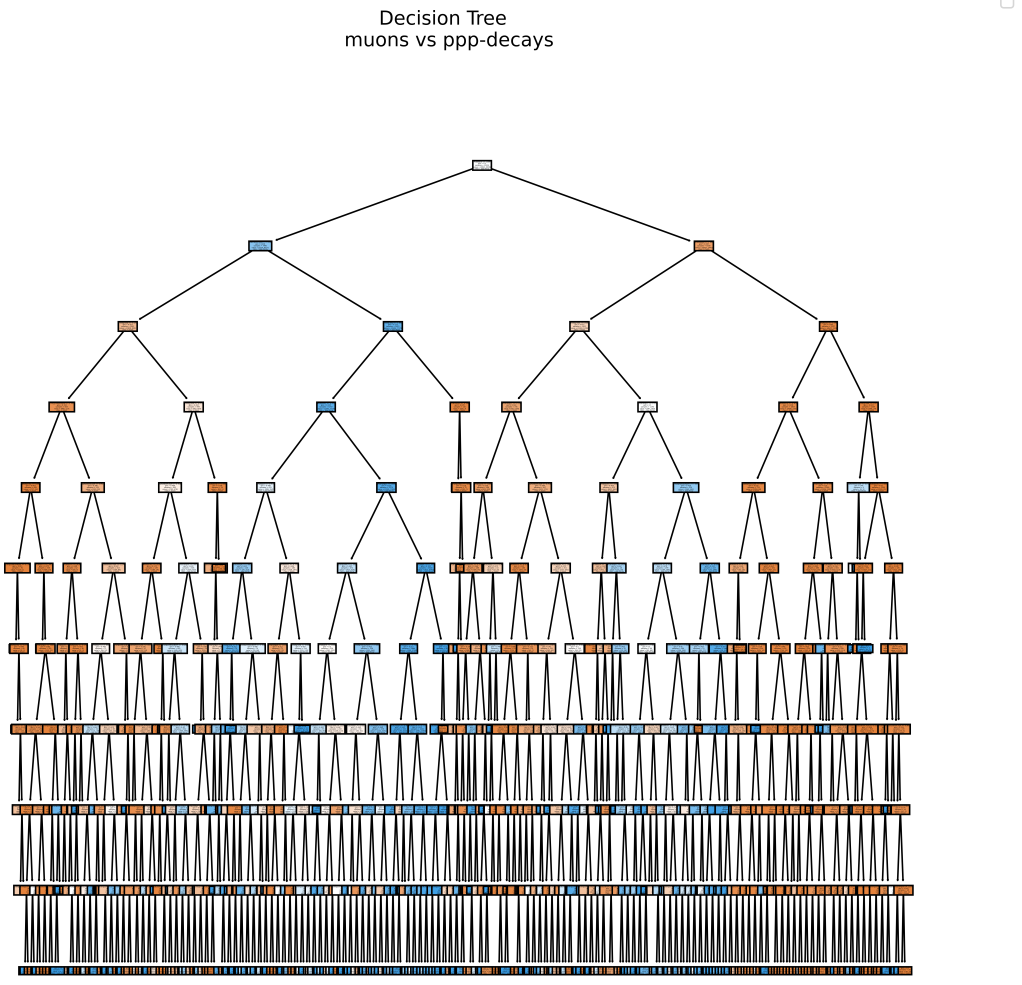


Figure 6.4 Decision Tree Visualization trained and plotted with the "sklearn" python library [9]. This is a decision tree taken from a random forest trained upon 10,000 of each type of simulation in the CUORE detector. The nodes at the bottom of the tree show when a final decision is reached. Nodes which are more likely to be muons appear orange, while nodes that appear more likely to be ppp-decays are blue. This tree is more realistic than the one shown in Fig 6.2. Note there is a trade-off between performance and readability. Our final random forest used to classify muons vs ppp-decays is 100 different versions of trees which look very similar to this one.

6.3 Algorithm: Generative Classifier (Density Estimation)

Suppose that for each class C_j we have a known density function $f_j(\overrightarrow{x})$. Further assume that for class C_j it's points $x_{j,i}$ were drawn from it's density function f_j. What is the probability that some new point x_{new} came from class C_j? We can use the known density functions f_j, and choose whichever class produced the highest density value for the new point. The process of using density functions for each class to choose which class a new point lies within is known as **Generative Classification**. The choice of class for the new point becomes:

$$\text{Class}(x_{new}) = \text{ArgMax}([f_0(x_{new}), f_1(x_{new})...f_n(x_{new})]) \tag{6.2}$$

Because generative classification relies entirely on modeling each group of points as density functions instead of choosing a partition, the entire problem reduces to estimating a probability density function from a set of known points. We could simply choose a parametric model out of the box for each set of points such as a simple Gaussian fit. Choosing simple Gaussians to model each class of points is known as a **Naive Bayes** classifier.

$$f_{i,NaiveBayes} = Gaussian(x, \mu = mean(\overrightarrow{x_i}), \sigma = StdDev(\overrightarrow{x_i})) \tag{6.3}$$

With any generative density estimation model, after defining the model we may want to define the **likelihood** of the model. In general the likelihood is defined as the probability of observing the data which was observed, after assuming the model was completely correct. For models which are densities themselves, the simplest form of the likelihood that we can define is a simple product over each datapoint in the dataset:

$$\mathcal{L}_{simple} = \prod_{n}^{N} f(\overrightarrow{x}_n) \tag{6.4}$$

6.3.1 Kernel Density Estimation

While it is possible to choose Gaussian as a density estimate for a set of points, in practice it is much more practical to have a non-parametric model, which can more easily bend it's shape around the points of interest. The simplest non-parametric density estimation algorithm is known as a "Kernel Density Estimation" [28, 29]. Instead of modeling the points with a single gaussian, we use a sum of smaller Gaussians. Each small Gaussian is centered upon a single observed datapoint. A kernel density estimation has only 1 **hyperparameter** called the "bandwidth" b, which defines the width each individual Gaussian centered about each observed point. We can show an example dataset (Eqn. 6.5), the equation of the kernel density (Eqn. 6.6), and the image of the estimate (Fig. 6.5) using a bandwidth of $b = 2.5$.

$$Dataset1 \equiv [-3, -2, -.5, 5.1, 6] \tag{6.5}$$

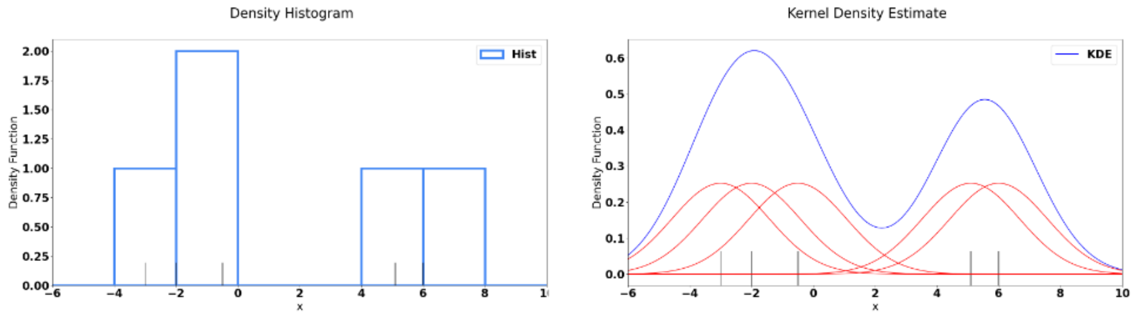


Figure 6.5 Kernel Density Estimation Example This histogram and smooth curve are both estimates for Dataset1 (Eqn 6.5). This particular image shows a hyper parameter of $b \approx 2.5$.

$$f_{kde}(x) = \sum_i Gaussian(\mu = x_i, \sigma = b, x) \tag{6.6}$$

The standard likelihood function defines the probability of generating a sample of points which were observed assuming that some density function $f(x)$ was used to generate the sample. The likelihood is equal to the product of the probability density of each point. We can formally write down the likelihood as follows:

$$\mathcal{L}_{kde,simple} = \prod_k \sum_n Gaussian(\mu = x_n, \sigma = b, x_k) \tag{6.7}$$

For the context of training machine learning algorithms, later we will find bandwidths using the likelihood with proper validation techniques (Sec 6.11). While rules of thumb have been developed for estimating a bandwidth [30, 31], it is illustrative to calculate bandwidths using iterative optimization methods directly upon a likelihood. We will show that further "tweaks" must be made to the likelihood for fitting purposes.

6.4 Hyperparameter Training

After choosing a classification algorithm, we have to train it on our data. In general the algorithm will be a function of the data we give it to train upon, **regular parameters** ($\overrightarrow{b_{regular}}$) which vary when training one single time, as well as some additional **hyper-parameters** ($\overrightarrow{b_{hyper}}$) which introduce changes to the overall structure of the algorithm. To relate the concept to previous reading in this document, examples of hyper-parameters could be: the standard deviation width σ described in the support vector machines radial basis function, the number of leaves n_{leaves} in a decision tree, the number of leaves & trees & features per leaf: $[n_{leaves}, n_{trees}, n_{features}]$ defining a random forest, and the bandwidth b of the kernel density estimation. These hyper-parameters will change how a given model performs, and how complex it becomes.

Before we can start training algorithms, we have be able to ask how well the algorithm is performing. At first glance one might think that we can simply run the classification algorithm on a large set of training data, make claims about the success rates and be done. However in practice, one might produce a trained algorithm that is very good on exactly the data provided to train it, but very bad on new data it has not yet seen. Such an issue is known as the "over-fitting" problem. We need to carefully determine how our algorithm might be over-fitting the data before we can make claims about the algorithm's success rates within a larger data analysis.

The over-fitting problem is pervasive through-out machine learning. The entire point of machine learning is to get a computer to handle a broad set of possibilities of input data to produce good decisions. How then, do we get the computer to handle the inputs we know, and also handle new inputs not yet seen?

6.4.1 Over-Fitting: A Kernel Density Estimation (KDE) Example

A very clear concise example of over-fitting can be illustrated as follows. Suppose one has N data points x_i which lie upon the real number line. Suppose also we wish to model these points with univariate probability density function $f(x)$. Thus we have a **density estimation** problem. Suppose further that we have decided to model the density of this single set of points, using the Kernel Density Estimation defined earlier (Eqn 6.6).

Before we can train our model, we need a metric to evaluate its performance. One might naively suggest that a reasonable metric which can be defined, is a standard mathematical likelihood $\mathcal{L}_{kde,simple}$ function defined previously (Eqn 6.7). It turns out that using the simple likelihood to evaluate the best choice of parameter b will yield a terrible solution.

The simple likelihood (Eqn 6.7) will be maximized under the condition that the bandwidth hyper parameter, b, will approach zero $Max(\mathcal{L}) = f_{kde}(b \to 0)$. Mathematically this will reduce our model to a sum of delta functions, each centered about a different observation, instead of a sum of Gaussians. (Reminder, $\lim_{b\to 0} Gaussian(\mu, b, x) = \delta(x - \mu)$). A comparison between bandwidth choices can be seen in figures 6.6 and 6.7.

By using the definition of a likelihood, and finding best fit parameters we have arrived at a drastically wrong solution. How could we have optimized our likelihood model, and come to the conclusion that delta functions are the best choice to model our data? Intuitively we know that choosing tiny bandwidths (such as $b = 0.01$) is wrong and a poor choice of hyperparameter. The problem stems from using the same data to both create the model f, and to construct the likelihood $\mathcal{L}$. The likelihood was maximized by training our model to assume that the data we observed, is 100% probable, and any data we have not yet observed, is almost impossible. Upon further reflection our model was trained in exactly the way we would expect under such conditions. In order to avoid such an obvious over-fitting, we need to modify our objective. Our likelihood function should reflect the probably of seeing some yet unseen point, instead of assuming the points we have already seen were the only possibilities. Changing the philosophy used to build the likelihood, will naturally cause the likelihood to conclude $b = 0.001$ as a poor choice of hyperparameter and instead favor a more reasonable bandwidth such as $b = 1$ for our example.

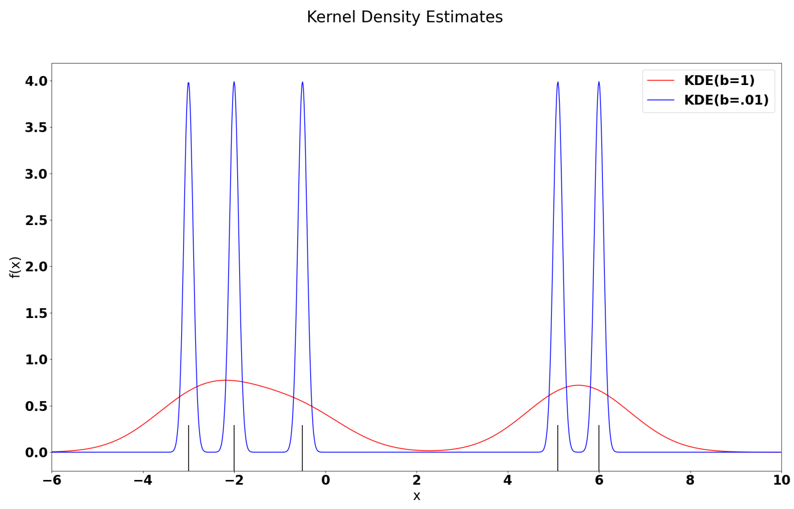


Figure 6.6 Kernel Density Estimation with different Bandwidths. We use Dataset1 (Eqn: 6.5) which is the dataset shown in (Fig 6.5).

6.4.2 Training, Validation, Testing: (KDE Example Continued)

To avoid the over-fitting problem, a standard technique to quantify how well our algorithm is performing is to split data into different groups. We can split our full pool of data into 3 distinct groups: Training, Validation, Testing. The largest portion of data is used for training. The second largest portion is used to validate our training, and the third portion is used to quantify performance upon completion.

For our illustrations above the Dataset1 (Eqn: 6.5) defined above was so small that attempting to split the data with training validation and testing would not leave us enough data to even attempt a kernel density estimation (KDE). There exist techniques for handling a "small data problem" as well, which we will discuss later in sec 6.4.3. For now, let us define another dataset which has a few more data points than (Eqn 6.5) but which was drawn from the same underlying bi-modal distribution.

$$Dataset2 \equiv [-3, -2.5, -4, -2.7, -2, -1. - .5, 4., 4.5, 4.7, 5.1, 5.5, 6] \tag{6.8}$$

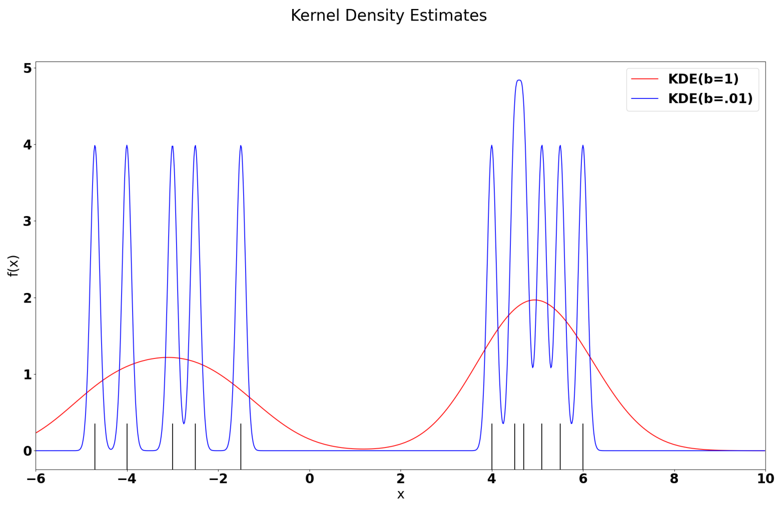


Figure 6.7 Kernel Density Estimation with different Bandwidths (b=.01, b=1). We use Dataset2 (Eqn: 6.8), which has data-points shown in black markers near the x-axis.

Randomly select 3 datapoints for validation, and 3 for testing, leaving only 6 for training:

$$\begin{aligned} Dataset2_{validation} &\equiv [-2.7, 4.7, 6] \\ Dataset2_{testing} &\equiv [-2, -3, 5.5] \\ Dataset2_{training} &\equiv [-2.5, -4, -1. - .5, 4., 4.5, 4.7, 5.1] \end{aligned} \tag{6.9}$$

Our model is then a sum of gaussians only over the training dataset.

$$f_{training}(x) = \sum_{training,i} Gaussian(\mu = x_{training,i}, \sigma = b, x) \tag{6.10}$$

Our validation likelihood function (the function we need to optimize) is now only calculated by multiplying over the validation datapoints (instead of all the datapoints):

$$\begin{aligned} \mathcal{L}_{validation} &= \prod_{validation,j} f(x_j, b) \\ &= f(-2.7, b) f(4.7, b) f(6, b) \\ max(\mathcal{L}_{validation}) &= \mathcal{L}_{validation}\Big|_{b=.791} \end{aligned} \tag{6.11}$$

Now we can actually run a simple optimization method on the kernel density estimate created from dataset 2. Maximizing $\mathcal{L}_{validation}$ by varying b (Eqn 6.11) turns out to yield the best bandwidth from the validation data of $b \approx 0.791$.

Our final likelihood test result is then calculated using only the test data, and using the best bandwidth obtained from the training-validation process. By sequestering a portion of the data from the training process entirely, we know that the model design has not been influenced by the test data. In the general case of fitting any model (density estimation, regression, classification), we can "trust" the test likelihood result, as long as it was not calculated/tainted with data used to construct the model in the first place. For completeness we calculate the test likelihood, and we can plot the final trained model with the best bandwidth according to the validation.

$$\begin{aligned}\mathcal{L}_{test} &= \prod_{test,i} f(x_i, b = 0.79) \\ &= f(-2, b = 0.79) f(-3, b = 0.79) f(5.5, b = 0.79)\end{aligned} \tag{6.12}$$

6.4.3 Cross Validation: Introduction

The simplest way to avoid over-fitting is to split data into training, validation, and testing partitions (sec 6.4.2). However, by splitting the data into such partitions, we are reducing the potential size of our training data that could have been used to improve the model. For example in our KDE model, we only had 12 data-points to start with, and then half were sequestered away to avoid the over-fitting problem, thus we only had 6 to create the model in the first place. **Cross-Validation** defines the set of more sophisticated techniques which are used to prevent over-fitting, while allowing larger portions of the dataset to be used in the training of the model.

The basic trick to each cross validation algorithm is to run the training, validation step we ran before, many times with different portions of the data each time. For example, in our KDE example, what if instead of having predefined training, validation, testing, data partitions, we lump the training and validation data-sets into a single partition of 9 data-points. **Twofold** cross-validation uses half the data to train, and half to validate, then swaps halves, retrains for both cases, and takes the average of both cases as our score. **Leave-one-out** validation uses all but one data-point as training, then validates on the left-out point, and then switches through all possible points, and again averages at the end. A **k-fold** validation achieves a balance between two-fold and leave-one-out validation techniques, by dividing the data into equal fractions, and isolating one equal part for validation, and repeating each part, and again averaging the results at the end.

In general one can play many versions of such games with choosing different subsets for training, and for validation, then taking an average. By performing these validation games, we can increase the amount of data we are using to both train, and to validate, at the cost of extra computation. In general it is advisable to have some validation at the very least for training hyper-parameters, and at a minimum do the training/validation/testing split. In practice however, it is most accepted to use some version of a k-fold validation, which achieves a balance between the maximum use of data and computational cost.

These validation games are worth playing more carefully, as we enter situations were the data are more scarce. For example, consider in the highly limiting case of having only 12 data-points in our KDE example. One could choose to brute force through every possible choice of training/validation data subsets and remove testing entirely. The brute force example takes an average over all of the $\binom{12}{k}$ $\forall k < 12$ possibilities. Such an extreme approach extracts as much information as possible from each datapoint, at the cost of more computation operations. We also can feel justified to avoid wasting data on a test, because we have included every possible test already within the validation. We could not possibly favor one subset of data over another, and thus we avoid the over-fitting problem.

The takeaway is that there is a trade-off between ease of execution (computationally and interpretability), and robustness of the cross-validation technique to make use of all the data. There is no generally accepted best practice, which will work for any dataset. However, if the size of the available dataset feels large, it is almost always easiest to form the simplistic data partitions of 50-70% training, and the rest validation & testing. If the size of the available data is small, one should consider using some kind of multiple-validation technique that tries many different partitions and averages over them.

6.5 Assessing Classification Performance: Confusion Matrix

The most common way of visualizing the performance of a machine learning classification algorithm is to use a **confusion matrix**. After a classification algorithm is trained upon one dataset, it is tested upon another. The test is to take a set of events for which we have the true classification, and ask the algorithm to attempt to classify those same events using only the features of those events. The algorithm's prediction can then be visually compared against the truth using the confusion matrix. A confusion matrix is a square matrix with the number of rows and columns equal to the number of kinds of events that need to be classified. Each possible case of correct classification and incorrect classification is included within the matrix. Correct predictions are represented by the counts along the diagonal of the matrix. A perfect classification algorithm will result in a confusion matrix that contains all zeros off the diagonal.

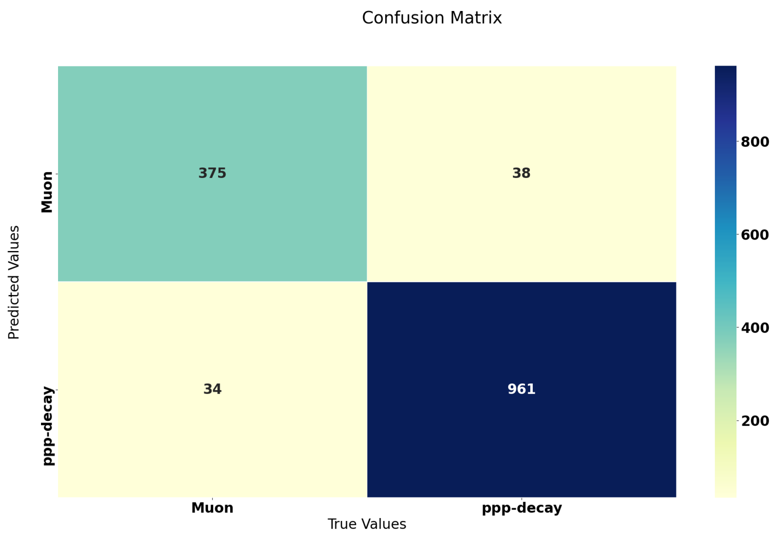


Figure 6.8 Example Simulated Confusion Matrix plotted with Seaborn python package [10]. The true source is represented by the horizontal axis, and the classification identification is represented by the vertical axis. Here 34 simulated muons are falsely identified as ppp-decays (signal), and 961 simulated ppp-decays are correctly identified as signal. Also 375 events are correctly identified as muons and 38 ppp-decay events are falsely identified as muons.

Here we provide an example (Fig 6.8) with only 2 sources, for a total of $2 \times 2 = 4$

possible elements in the matrix. Our confusion matrix shows a false negative and false positive rate of our signal and background sources.

6.6 Classify CUORE Simulations: ppp-decays vs muons

We now have some completed CUORE simulations with associated features, an understanding of classification techniques, knowledge of using validation to avoid overfitting. Here we will put all the pieces together to classify muons vs ppp-decays.

6.6.1 Choosing a classification Algorithm

First to choose a classification algorithm, we have to say something about how each performs. We have taken the approach of attempting a suite of standard tools an applying all of them to different amounts of test simulation data. Because we use python for the majority of our data analysis code, and we wish to use an existing classification algorithm, it was convenient for us to use the package "sklearn" [32]. We use the full list of available sklearn package classification algorithms with default parameters upon 10,000 ppp-decay muons simulations. We then choose the algorithm which performs the best. We have chosen to use the random forest algorithm.

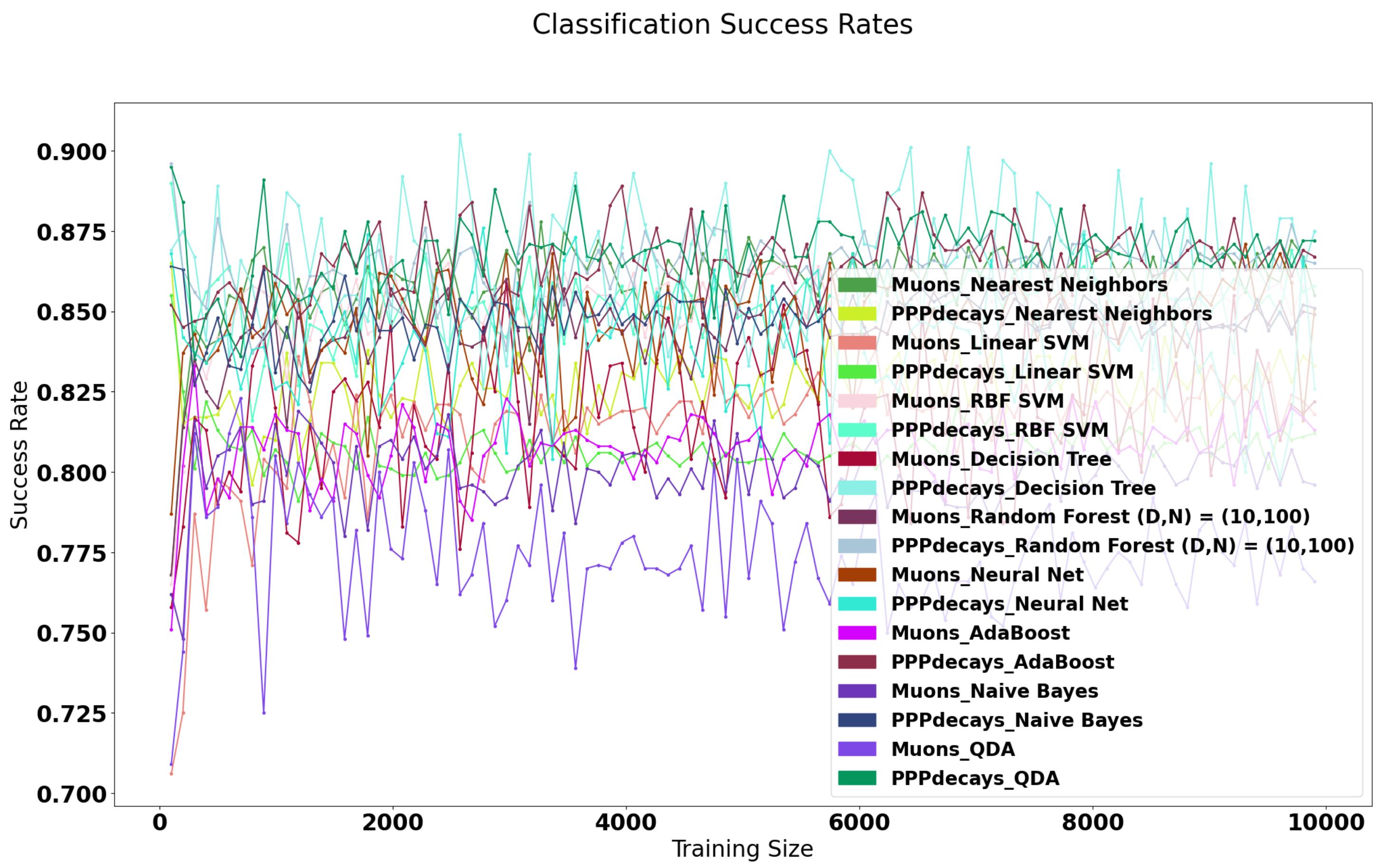


Figure 6.9 Many different available classification algorithms within the "sklearn" python package [9] are run against the CUORE muon and ppp-decay simulations. Their success rates at classifying both signal and background are shown. The figure is difficult to interpret but is complete. As more data is used the classification algorithms perform better and better. A separate validation set of data is isolated from the training data for each training run. We find that a random forest has the highest success rate, in conjunction with a low statistical variance on the success rate.

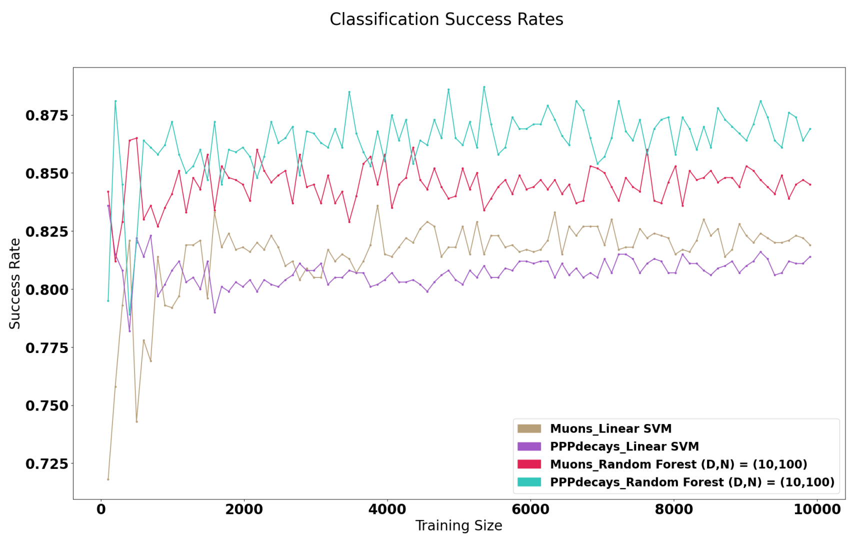


Figure 6.10 Classification success rates of using Sklearn's linear support vector machines, and random forests upon CUORE simulations of muons and ppp-decays [9]. The x-axis is the same as shown in figure 6.9. Both algorithms can differentiate signal from background better than a coin flip, and both improve as the amount of simulation data available to train upon is increased. However the random forest outperforms the support vector machines.

6.6.2 Validating our algorithm

In practice we will be using "Random Forests" (sec 6.2). In general a random forest has the benefit of being easy to visualize, quick to train, and have great performance. We can see from trying out the default parameters on many different algorithms, we can see that random forests was one of the best performers. Although not presented here, from running the training program, it can be shown that training the random forests was computationally efficient compared to some of the other algorithms (e.g. Gaussian process regression was prohibitively slow, and could not even be attempted), Additionally because the random forest is quick to execute we can attempt different visualization schemes.

Although we have attempted many algorithms already (Fig 6.9) we have not yet obtained best possible performance from the algorithm of interest. The support vector machines algorithm training (Fig 6.10) did not require any hyper-parameters to train, so its performance cannot be improved. The random forest algorithm however, has a few different parameters we can tweak to potentially improve performance upon the same dataset. Hyper-parameters that define the random forest include: the number of trees, the depth of each tree, and the number of features to select upon in each node of the tree. Due to the nature of the data we are using, and the ease of interpretation, we will limit the number of features per node to be exactly 1. The two remaining hyper-parameters we can train are number of trees (n_estimators), and the depth of each tree (max_depth).

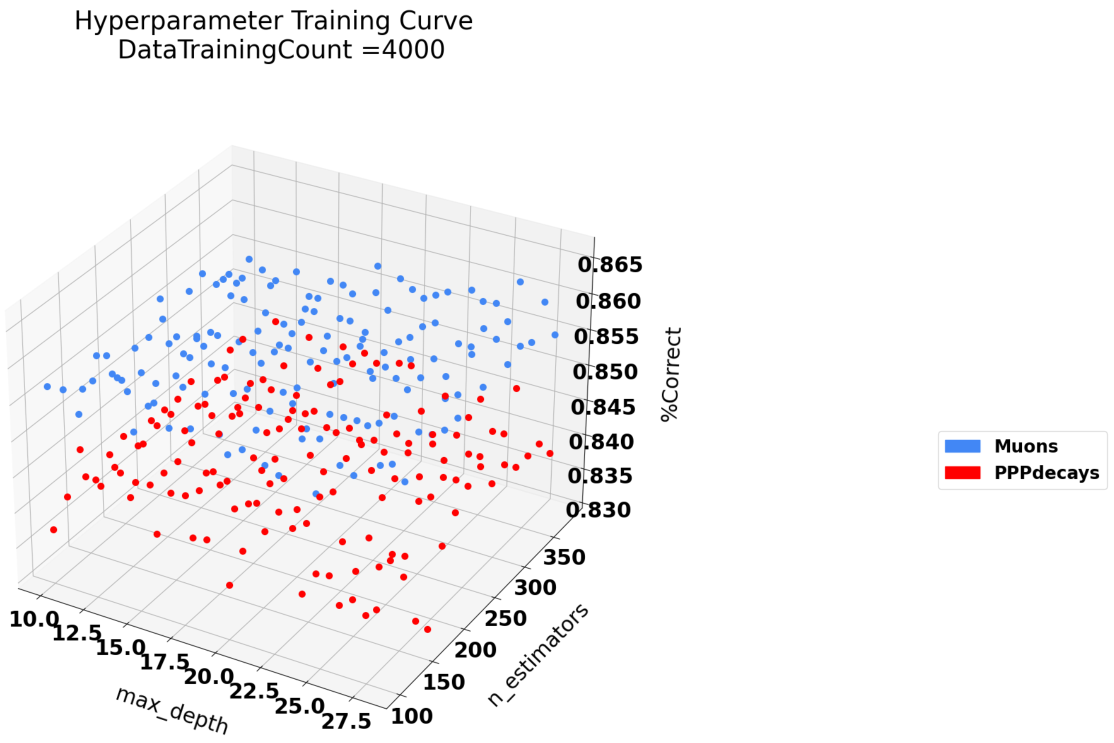


Figure 6.11 Validation hyperparameter investigation for random forests used upon 5000 ppp-decay and muon simulations. There is slight improvement as both the tree count increases, and as the depth increases, however we find diminishing returns for increasing parameters beyond their default values of of (n_estimators = 100), (max_depth=10) on our specific classification problem.

We tried various ranges for random forest depth and tree count, and found that the performance did not improve much beyond the default parameters, which have depth of 10, and number of trees to be 100. In Fig 6.11. We decided to use the default parameters available in the sklearn machine learning package [32]. Possible future work for improving the analysis could be to try throwing more computation power behind larger random forests with millions of trees to see if the performance improves, or if there is some phase shift where many more simulations are required to see that having more trees in the random forest provides some kind of step functional improvement.

6.6.3 Training the chosen algorithm

We can also see from Figs (6.9, 6.10) that although the performance increased as we get more data, there are diminishing returns to the improvement of the result. After about 5000 simulations of each type of data, the test results upon the same 1000 separate data-points do not improve much as the count of the number of simulations is increased to 6000. We can vet our conclusion that the amount of data will not improve the classification rates by much by further increasing the simulation count. We can see that increasing the simulation count from 6000 simulations of each kind of event, to 100,000 of each kind of event only slightly increases the classification success rates. One could actually fit a curve to the classification success rate curve as a function of simulation count, and show that obtaining 1 Million of each kind of simulation will only improve the classification rate by a fraction of a percent. We can then look at the sensitivity plot (Sec 9, 9.7) and see that such a small improvement of classification success rate will not actually improve our final result. For these reasons we feel that training our classifier upon only 100,000 of each kind of simulated event is sufficient to obtain a strong final half-life bound on a triple proton decay search.

When performing our final analysis we use the random forest trained on the largest possible training dataset. We classify muons correctly about 85% of the time and ppp-decays correctly about 88% of the time. We will assume that the standard deviation on the classification success is negligible to the analysis. In Fig 6.12 one can clearly see the standard deviations are less than 1% compared to the success rates in the mid 80's.

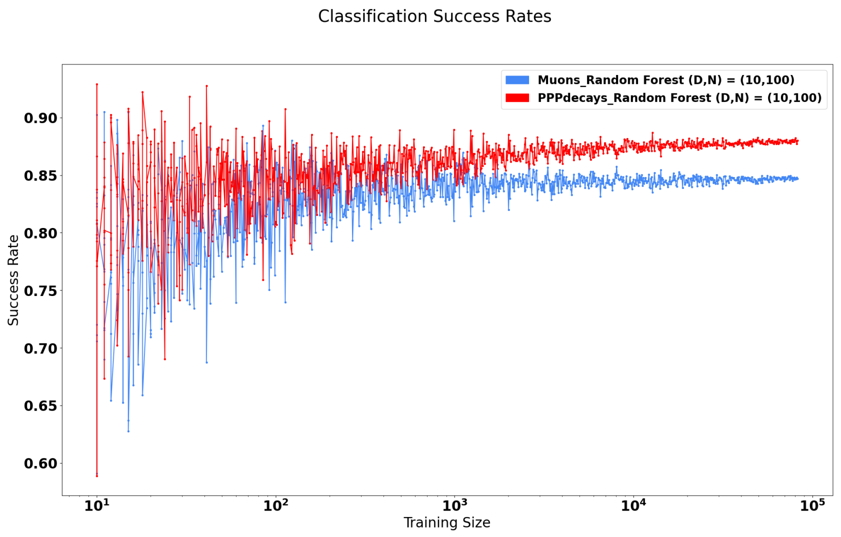


Figure 6.12 Classification success rates of using Sklearn's random forest with default hyper-parameters (n_estimators = 100), (max_depth=10), upon CUORE qshields simulations of muons and ppp-decays [9]. The x-axis on this plot is in log-scale as compared to that shown previously in figure 6.9. One can see that there are diminishing returns to the value of adding additional simulations to the training dataset. Beyond having 10,000 of each kind of simulation, there is little improvement.

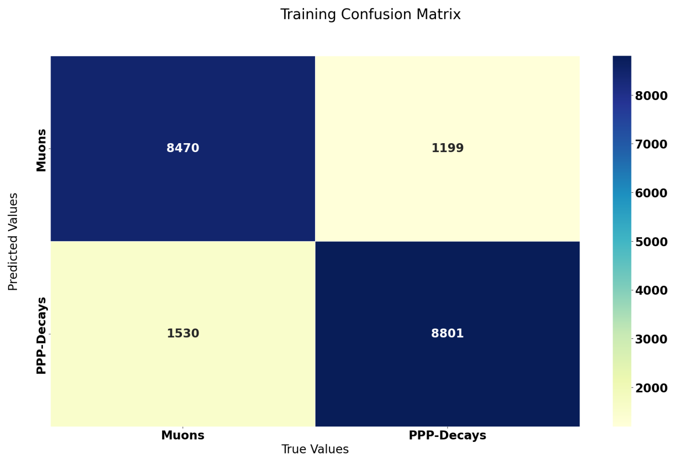


Figure 6.13 The confusion matrix of results of training a random forest with 90k training samples, on a 10k test dataset. We classify muons correctly about 85% of the time. We classify ppp-decays correctly about 88% of the time. This confusion matrix was used to produce the last datapoint in Fig 6.12. Figure was produced using the Seaborn python package [10].

6.6.4 Classification Results on real data

Previously (Fig 6.12) we have chosen, validated, and trained on simulations a random forest classification algorithm. Now we use the algorithm upon the 2364 events within the real data-sets [3612, 3613, 3614, 3615]. Our most well trained random forest thinks it has seen 386 events that look like most like ppp-decays, and 1978 events that look most like muons. These numbers are in line with an assumption that all the observed events are actually muons, and the random forest algorithm has mis-classified 386 of them as triple proton decays.

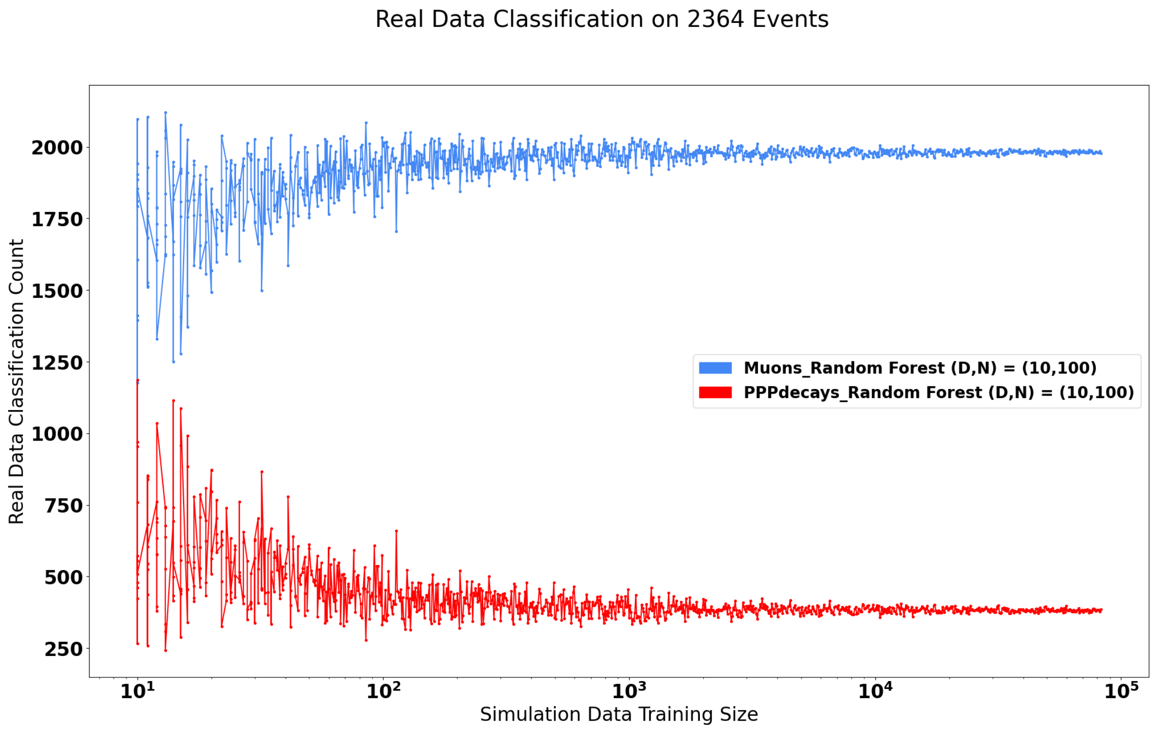


Figure 6.14 Results of classifying the 2364 real events from data-sets [3612, 3613, 3614, 3615] with multiplicity greater than 10, in the CUORE detector. Each data-point on this figure, corresponds to a datapoint on Fig 6.12. Our most well trained random forest thinks it has seen 386 events that look like most like ppp-decays, and 1978 events that look most like muons.

Chapter 7

Poisson Counting Analysis: No Classification

7.1 Background Alone

For our analysis we need the probability density function describing the expected number of counts, in relation to the number observed, only for background material. We will assume the background density function is Poisson distributed (for reasons discussed in Sec 5):

$$Bkg_{pdf}(\lambda_{bkg}, k_{bkg}) \approx \frac{\lambda_{bkg}^{k_{bkg}} e^{-\lambda_{bkg}}}{\Gamma(k_{bkg}+1)} \tag{7.1}$$

Additionally, we may want to define a prior for the expected number of background events λ_{bkg}. We may not be certain the true value of λ_{bkg} and wish to express the uncertainty in a distribution. We will treat λ_{bkg} instead as a random variable, denoted with a box around it as $\boxed{\lambda_{bkg} \text{ prior}}$. The full joint likelihood for the observed background count is then the Poisson distribution multiplied by the density function of the prior:

$$Bkg_{pdf}(\lambda_{bkg}, k_{bkg}) \approx \frac{\lambda_{bkg}^{k_{bkg}} e^{-\lambda_{bkg}}}{\Gamma(k_{bkg}+1)} \boxed{\lambda_{bkg} \text{ prior}}_{pdf} \tag{7.2}$$

7.1.1 Example: Derive expected background count prior

One way one may derive a prior for the expected observed background count λ_{bkg}, would be to write it in terms of $\lambda_{half}, t_{end}, N$ which are the half-life, the time we observe the experiment, and the number of atoms respectively. We can use the previously shown relationships between $\lambda's$ (derived in sec 5.6).

$$\begin{aligned} \lambda_{half} &\approx \frac{t_{end} ln(2) N}{\lambda_{bkg}} \\ \lambda_{bkg} &\approx \frac{t_{end} ln(2) N}{\lambda_{half}} \end{aligned} \tag{7.3}$$

In the example, we assume that time is measured precisely, the effective background half-life is also known precisely but instead of having a fixed number or atoms N, we want to allow for the number to be modeled as a random variable. We then denote the random variable as $\boxed{N}$ which has a Gaussian density function.

$$\boxed{N}_{pdf} \equiv Normal_{pdf}(\mu_N, \sigma_N, v_N) \tag{7.4}$$

λ_{bkg} and N are now random variables instead of regular variables, so we will denote them now with boxes $\boxed{\lambda_{bkg}}$, $\boxed{N}$. The same algebra holds for the relationship between the random variables for their definitions, but the relationship between their pdf's is non-trivial.

$$\boxed{\lambda_{bkg}} = \frac{t\ ln(2)}{\lambda_{half}} \boxed{N} \tag{7.5}$$

$$\boxed{\lambda_{bkg}}_{pdf} \neq \frac{t\ ln(2)}{\lambda_{half}} \boxed{N}_{pdf} \tag{7.6}$$

We must carefully determine what $\boxed{\lambda_{bkg}}_{pdf}$ given $\boxed{N}_{pdf}$. We can use one of two different techniques to calculate densities from functions of random variables. We can use the distribution function technique, or we can use monotonic Jacobean transformation technique. In this particular case our new random variable is a constant times a Gaussian, and we can look up the answer without re-deriving the general solution. The density of a constant times a Gaussian, is a new Gaussian with mean and standard deviation scaled by the constant as follows:

$$\boxed{\text{Example } \lambda_{bkg} \text{ prior}}_{pdf} = \mathcal{N}(\frac{t\ ln(2)}{\lambda_{half}}\mu_N, \frac{t\ ln(2)}{\lambda_{half}}\sigma_N, v_\lambda) \tag{7.7}$$

The procedure outlined in the example, can be extended for more general cases. In our example, we knew the relationship between had uncertainty in the number of atoms N which caused uncertainty in λ_{bkg}.

7.2 Signal Alone

Separately, we can calculate an expected signal count. We will assume that the source of the signal is completely independent from the background. The only information we have is that the there exists an expected signal count λ_{sig}, which could be any positive number with equal probability. The constraint of positivity is a variant of the Jeffrey's Prior. The constraint gives us two pieces of information. First: a Poisson distribution to model the signal is the best choice (Shown in Sec 5). Second: the prior on the expected signal count reduces to a uniform distribution from 0 to inf which only effects normalization at the end of our analysis. Thus the full signal distribution, is a Poisson distribution (for reasons discussed in Sec 5) with unknown λ_{sig} times a constant prior that can be ignored:

$$\begin{aligned} Sig_{pdf}(\lambda_{sig}, k_{sig}) &\approx \frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)} \times \boxed{\text{No Info } \lambda_{sig}}_{pdf} \\ &\propto \frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)} \end{aligned} \tag{7.8}$$

7.3 Background + Signal

We can combine the background and the signal into a combined total count rate. Here we will introduce the sum of total observations k_{tot}, and the sum of expected values λ_{tot}. If we didn't have any information other than totals, we could model the full experiment with it's own Poisson distribution:

But we also know that signal, background, and total are related as follows:

$$\begin{aligned}\lambda_{tot} &= \lambda_{sig} + \lambda_{bkg} \\ k_{tot} &= k_{sig} + k_{bkg}\end{aligned} \tag{7.9}$$

And we know we can represent the number of total counts with the joint distribution below:

$$\begin{aligned}SigBkg_{pdf}(\lambda_{sig}, k_{sig}, \lambda_{bkg}, k_{bkg}) &= Sig_{pdf}(\lambda_{sig}, k_{sig}) Bkg_{pdf}(\lambda_{bkg}, k_{bkg}) \\ &= \frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)} \frac{\lambda_{bkg}^{k_{bkg}} e^{-\lambda_{bkg}}}{\Gamma(k_{bkg}+1)} \boxed{\lambda_{bkg} \text{ prior}}_{pdf}(\lambda_{bkg})\end{aligned} \tag{7.10}$$

We know that we saw a total number of events, but we cannot know if they were signal or background. Our goal is to place a constraint upon the signal observed, so we wish to marginalize over background. However, we did not observe the background count, only the total count. In our joint likelihood, we can replace instances of expected background count λ_{bkg} using the relationship between total count and signal count. $\lambda_{bkg} = \lambda_{tot} - \lambda_{sig}$. We arrive at an expression which is only dependent upon signal, and total count as follows:

$$\begin{aligned}SigBkg_{pdf}(\lambda_{sig}, k_{sig}, \lambda_{tot}, k_{tot}) = \left(\frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)}\right)\left(\frac{(\lambda_{tot} - \lambda_{sig})^{k_{tot}-k_{sig}} e^{-(\lambda_{tot}-\lambda_{sig})}}{\Gamma(k_{tot} - k_{sig} + 1)}\right) \\ \times \boxed{\lambda_{bkg} \text{ prior}}_{pdf}(\lambda_{tot} - \lambda_{sig})\end{aligned} \tag{7.11}$$

We can now obtain a distribution in λ_{sig}, k_{sig} only conditional on k_{tot}:

$$Sig_{pdf}(\lambda_{sig}) \propto \int_{k_{sig}} \int_{\lambda_{tot}} SigBkg_{pdf}(\lambda_{sig}, k_{sig}, \lambda_{tot}, k_{tot} = \text{Known Value}) \tag{7.12}$$

So the final result is a function of 4 variables, with 2 integrals, and 1 substituted value, thus only 1 free variable remains.

7.4 Signal with Exact Background

Given the special case where λ_{bkg} is known exactly, and is no longer a distribution, we can write down the joint distribution again without the background prior:

$$SigBkg_{pdf}(\lambda_{sig}, k_{sig}, \lambda_{tot}, k_{tot}) = \left(\frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)}\right)\left(\frac{(\lambda_{tot}-\lambda_{sig})^{k_{tot}-k_{sig}} e^{-(\lambda_{tot}-\lambda_{sig})}}{\Gamma(k_{tot}-k_{sig}+1)}\right) \tag{7.13}$$

And again we can substitute in the values which we do know, and marginalize over those which we do not. However, because the final form of the equation no longer has a Gaussian inside, we can actually do the integrals above by hand, instead of using MCMC. The known values are marked in red below for clarity:

$$\begin{aligned}
Sig_{pdf}(\lambda_{sig}) &\propto \int_{k_{sig}} SigBkg_{pdf}(\lambda_{sig}, k_{sig}, \lambda_{tot} = \lambda_{sig} + \lambda_{bkg(KNOWN)}, \\
&\qquad k_{tot} = k_{tot(KNOWN)}) \\
&= \int_{k_{sig}} \left(\frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)}\right)\left(\frac{\lambda_{bkg}{}^{k_{tot}-k_{sig}} e^{-\lambda_{bkg}}}{\Gamma(k_{tot}-k_{sig}+1)}\right) \\
&= \underbrace{\cancel{(\lambda_{bkg})^{k_{tot}} e^{-(\lambda_{bkg})}}}_{Constant} \int_{k_{sig}} \left(\frac{\lambda_{sig}^{k_{sig}} e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)}\right)\left(\frac{(\lambda_{bkg})^{-k_{sig}}}{\Gamma(k_{tot}-k_{sig}+1)}\right) \\
&\propto \int_{k_{sig}} \lambda_{sig}^{k_{sig}} (\lambda_{bkg})^{-k_{sig}} \left(\frac{e^{-\lambda_{sig}}}{\Gamma(k_{sig}+1)}\right)\left(\frac{1}{\Gamma(k_{tot}-k_{sig}+1)}\right) \\
&= e^{-\lambda_{sig}} \int_{k_{sig}} \left(\frac{\lambda_{sig}}{\lambda_{bkg}}\right)^{k_{sig}} \left(\frac{1}{\Gamma(k_{sig}+1)\Gamma(k_{tot}-k_{sig}+1)}\right)
\end{aligned} \tag{7.14}$$

Introduce a temporary variable, c, which stores some constants allowing for less typing. Also define the integral, I, which we want to solve.

$$\begin{aligned}
c &\equiv \left(\frac{\lambda_{sig}}{\lambda_{bkg}}\right) \\
Sig_{pdf}(\lambda_{sig}) &\equiv e^{-\lambda_{sig}} I \\
I &\equiv \int_{k_{sig}} \frac{c^{k_{sig}}}{\Gamma(k_{sig}+1)\Gamma(k_{tot}-k_{sig}+1)}
\end{aligned} \tag{7.15}$$

Reformulating the integration as a discrete sum we can do the following:

$$\begin{aligned} I &\equiv \sum_{k_{sig}} c^{k_{sig}} \frac{1}{k_{sig}!(k_{tot}-k_{sig})!} \\ &= \sum_{k_{sig}} c^{k_{sig}} \frac{1}{k_{tot}!} \binom{k_{tot}}{k_{sig}} \\ &= \frac{1}{k_{tot}!} \sum_{k_{sig}} c^{k_{sig}} \binom{k_{tot}}{k_{sig}} \\ &= \frac{1}{k_{tot}!} (1+c)^{k_{tot}} \end{aligned} \tag{7.16}$$

Which leads to a final result:

$$\begin{aligned} Sig_{pdf}(\lambda_{sig}) &\equiv e^{-\lambda_{sig}} I \\ &= e^{-\lambda_{sig}} \frac{1}{k_{tot}!} (1+c)^{k_{tot}} \\ &= e^{-\lambda_{sig}} \frac{1}{k_{tot}!} \left(1+\frac{\lambda_{sig}}{\lambda_{bkg}}\right)^{k_{tot}} \\ &\approx e^{-\lambda_{sig}} \frac{1}{\Gamma(k_{tot}+1)} \left(1+\frac{\lambda_{sig}}{\lambda_{bkg}}\right)^{k_{tot}} \end{aligned} \tag{7.17}$$

For normalization re-introduce the constant removed earlier:

$$Sig_{pdf}(\lambda_{sig}) \propto e^{-\lambda_{sig}} \frac{1}{\Gamma(k_{tot}+1)} \left(1+\frac{\lambda_{sig}}{\lambda_{bkg}}\right)^{k_{tot}} \underbrace{(\lambda_{bkg})^{k_{tot}} e^{-(\lambda_{bkg})}}_{Constant} \tag{7.18}$$

Note that in the Gaussian limit: λ_{tot} is the mean and variance of this distribution.

Chapter 8

Poisson Counting Analysis: With Classification

8.1 Poisson Classification Likelihood

We need to transform classification results in the form of a confusion matrix defined in Section 6.5) into a statement of detection or non-detection for likelihood analysis. From the matrix we can deduce a false positive and false negative rate for each potential source of signal. While the technique can be extended to any number of sources of data, in our case we only have data produced from two independent Poisson processes with expected count rates λ_{bkg} for muons, and λ_{sig} for ppp-decays:

$$\begin{aligned} \lambda_{bkg} &= \text{muon count rate} \\ \lambda_{sig} &= \text{ppp decay rate} \end{aligned} \tag{8.1}$$

Our two corresponding classification success rates are $p_{sig,good}$ and $p_{bkg,good}$ defined as follows:

$$\begin{aligned} p_{sig,good} &= \text{Probability of classifying true signal correctly} \\ p_{bkg,good} &= \text{Probability of classifying true background correctly} \\ p_{sig,bad} &= 1 - p_{sig,good} = \text{Probability classify signal incorrectly} \\ p_{bkg,bad} &= 1 - p_{bkg,good} = \text{Probability classify background incorrectly} \end{aligned} \tag{8.2}$$

We need to include these classification probabilities into the the likelihood. We need to re-write the previous calculated likelihood allowing our previous k_{sig} and k_{bkg} to now become random variables. To incorporate the difference between observed counts, and hidden true counts we need the new variables below:

$$\begin{aligned} k_{sig,true} &= \text{ True signal count in the detector} \\ k_{bkg,true} &= \text{ True background count in the detector} \\ k_{sig,obs} &= \text{ Observed signal count by attempting to classify} \\ k_{bkg,obs} &= \text{ Observed background count by attempting to classify} \end{aligned} \tag{8.3}$$

To determine the total likelihood, we need to consider the contributions from each part of the confusion matrix separately. We will consider the cases in separate peices below:

8.1.1 Consider Signal Seen as Signal ONLY:

If we knew the true observed signal we could represent our count probabilities with a single simple Poisson distribution. Instead each event can be seen as signal or background. Thus we need to multiply by the binomial distribution.

$$P(\lambda_{sig}, k_{sig,true}, k_{sig,obs}) = \text{Poisson}(\lambda_{sig}, k_{sig,true})\text{Binom}(k_{sig,obs}, k_{sig,true}, p_{sig,good}) \tag{8.4}$$

To isolate the λ_{sig}, we need to marginalize over the information we don't know, and substitute the information we do know. We do not know the true value of the number of signal events ($k_{sig,true}$), but we do know the number of signal we have observed ($k_{sig,obs}$). We can then marginalize the Poisson times a binomial in Eqn 8.4 over the true number of signal events. The result reduces to a different Poisson distribution as follows (algebra shown in Appendix D):

$$\begin{aligned} P(\lambda_{sig}, k_{sig,obs}) &= \sum_{k_{sig,true}=k_{sig,obs}}^{\infty} \text{Poisson}(\lambda_{sig}, k_{sig,true})\text{Binom}(k_{sig,obs}, k_{sig,true}, p_{sig,good}) \\ &= \text{Poisson}(p_{sig,good}\lambda_{sig}, k_{sig,obs}) \end{aligned} \tag{8.5}$$

8.1.2 Consider Background Seen as Signal ONLY:

If we could classify events perfectly the background data source's contribution to our signal would obviously be zero. However because we cannot classify perfectly, we need to calculate an expected count. Fortunately the mathematics used to derive Eqn 8.5 is the same as the mathematics used to derive Eqn 8.7.

$$P(\lambda_{bkg}, k_{bkg,true}, k_{sig,obs}) = \text{Poisson}(\lambda_{bkg}, k_{bkg,true})\text{Binom}(k_{sig,obs}, k_{bkg,true}, p_{bkg,bad}) \tag{8.6}$$

$$\begin{aligned} P(\lambda_{bkg}, k_{sig,obs}) &= \sum_{k_{bkg,true}=k_{sig,obs}}^{\infty} \text{Poisson}(\lambda_{bkg}, k_{bkg,true})\text{Binom}(k_{sig,obs}, k_{bkg,true}, p_{bkg,bad}) \\ &= \text{Poisson}(p_{bkg,bad}\lambda_{bkg}, k_{sig,obs}) \end{aligned} \tag{8.7}$$

8.1.3 Consider ALL events seen as Signal:

We need to add together (Sig seen as sig) + (Bkg seen as signal).

$$\begin{aligned} \boxed{k_{obs,sig,S}}\&\boxed{k_{obs,sig,B}} &\text{ are random variables} \\ \boxed{k_{obs,sig,S}} &\text{ is described by pdf } \boxed{\text{Sig seen as Sig}}_{pdf} \\ \boxed{k_{obs,sig,S}} &\text{ is described by pdf } \boxed{\text{Bkg seen as Sig}}_{pdf} \end{aligned} \tag{8.8}$$

$$\begin{aligned} \boxed{\text{Sig seen as Sig}}_{pdf} &= \text{Poisson}\left(p_{sig,good}\lambda_{signal}, k_{obs,sig}\right) \\ \boxed{\text{Bkg seen as Sig}}_{pdf} &= \text{Poisson}\left(p_{bkg,bad}\lambda_{bkg}, k_{obs,sig}\right) \end{aligned} \tag{8.9}$$

The two random variables of interest ($\boxed{k_{obs,sig,S}}$, $\boxed{k_{obs,sig,B}}$)underlying variables in their respective probability density functions are independent and uncorrelated. Thus we can do a simple density convolution to add them, which reduces to yet another poisson distribution:

$$\begin{aligned}\boxed{\text{Sig Seen Tot}}_{pdf} &= Convolve\Bigg(\text{(Sig seen as sig)}, \text{(Bkg seen as Sig)}\Bigg) \\ &= \mathcal{L}_{joint} = \text{Poisson}\Bigg(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg}, \quad k_{obs,sig}\Bigg)\end{aligned} \tag{8.10}$$

If we know λ_{bkg} exactly we can substitute into equation 8.10. Alternatively if we have extra information about λ_{bkg} but do not know it's exact value then we can include the information at the end as a prior pdf and integrate over it (just as in section 7.3).

$$Sig_{pdf}(\lambda_{sig}) = \int d\lambda_{bkg}\text{Poisson}\Bigg(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg}, \quad k_{obs,sig}\Bigg)Bkg_{pdf}(\lambda_{bkg}) \tag{8.11}$$

8.2 Figure of Merit

We can ask how does the final probability density function change as a function of the classification probabilities? In the limit of large statistics, Poisson distributions become Gaussian distributions with $\sigma = \sqrt{\lambda}$ and $\mu = \lambda$ (See Section E for details). Our major source of background is muons, which will have large enough statistics for the large limit to apply. To obtain a simple figure of merit we represent the total observation count probability density function as a Gaussian instead of a Poisson:

$$\begin{aligned}\boxed{\text{Sig Seen Tot}}_{pdf}(k_{sig,obs}) &= \text{Poisson}\Bigg(\underbrace{p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg},}_{\lambda_{sig,obs}} \quad k_{obs,sig}\Bigg) \\ &\approx \mathcal{G}auss\Bigg(k_{obs}, \mu = \lambda_{sig,obs}, \sigma^2 = \lambda_{sig,obs}\Bigg)\end{aligned} \tag{8.12}$$

A common figure of merit is the **Signal to Noise Ratio** (SNR) [33] defined as the mean divided by the standard deviation:

$$SNR = \frac{\mu}{\sigma} \tag{8.13}$$

For our observed signal μ_{sig}, vs our observed background σ_{bkg}, the SNR is then:

$$\sigma_{bkg} \propto \sqrt{\mu_{bkg}} \propto \sqrt{p_{bkg,bad}} \tag{8.14}$$

$$\mu_{bkg} \propto p_{bkg,bad} \tag{8.15}$$

$$\mu_{sig} \propto p_{sig,good} \tag{8.16}$$

$$\text{and the Figure of Merit} \equiv FOM = \mu_{sig}/\sigma_{bkg} \propto \frac{p_{sig,good}}{\sqrt{p_{bkg,bad}}} \tag{8.17}$$

8.3 Future work: Figure of Merit to Optimize Classification

The figure of merit could be used as a metric to use to guide the machine learning step of our analysis pipeline. In our case, the machine learning component of our analysis is a classification algorithm. We relied upon a default "accuracy" metric. The analysis could be improved by switching from accuracy to a different metric.

Earlier, we chose which classification algorithm to use, a set of hyper-parameters, and regular parameters. We chose a random forest as the algorithm amongst a list of well accepted competitors (Sec 6.6.1). We manually chose two hyper-parameters (Sec 6.6.2). To make the choice of algorithm and hyper-parameters, we had to rely upon sklearn's default methodology for training regular parameters. The default method for training regular parameters upon data is to use an "accuracy" score which is an average of classification success probabilities. Maximizing accuracy is equivalent to maximizing the sum of the counts along the diagonal of the confusion matrix.

Instead of using accuracy to score performance, one could change the scoring function or "objective" function to some other metric for success. For our analysis, using the figure of merit (Eqn 8.17) would be a better choice for a classification algorithm to optimize. One could use the figure of merit scoring function for choice of algorithm, hyper-parameter validation, and final classification training. (We did not consider training the random forest and the support vector machines to optimize the figure of merit when choosing between the two algorithms. We did not force the algorithms to favor internal parameters which mis-classify signal over parameters which mis-classify background.) Using the figure of merit scoring function would obtain a stronger final result, with the same set of CUORE observations.

Changing the scoring function within an existing machine learning algorithm is non-trivial. Each machine learning algorithm relies upon a unique internal set of linear algebra tricks to train accuracy efficiently. Thus if we change the scoring function, we may also have to change each classification algorithm's internal optimization method. Using the same generic parameter optimization technique on all the classification algorithms, (such as simulated annealing) is one option to reduce the scope of the work. However, generic parameter optimization techniques are both computationally slower, and less reliable, than existing well tested accuracy-based linear methods. Designing an efficient linear-algebra-based optimization method for a new scoring function on a single classification algorithm has the potential to be an entire research paper.

The level of improvement from using a figure of merit as a scoring function is difficult to estimate in advance of performing the work. After changing the scoring function, it is not obvious exactly how a machine learning classification algorithm will shift events around within a confusion matrix.

Due to the difficulty of efficient implementation, we relegate re-training the various machine learning algorithms based upon a figure of merit to possible future work. This subsection serves as a note to the reader that using the figure of merit instead of accuracy could be used to improve a future analysis, without obtaining more observation data.

Chapter 9

Sensitivity Calculation

9.1 Calculate Marginal Posterior

The posterior was originally calculated in a previous section (Eqn: 8.11) and is re-written here for ease of reading. For illustration purposes, we will pick a number for the expected background Muon flux rate. Let us assume that we expect about 6×10^4 detectable muons to hit our detector over a 5 year period. We will define $Gaussian(x_{bkg}, \mu_{bkg}, \sigma_{bkg})$ to represent our knowledge of the the total number of muons we expect to observe without using any data. We define $\lambda_{sig}, \lambda_{bkg}$ to be the expected count totals observed from the experiment's measurement of signal and background Poisson processes respectively. The posterior is a function of both λ_{sig} and λ_{bkg}.

$$\begin{aligned} Post_{pdf}(\lambda_{sig}, \lambda_{bkg}) = &\text{Poisson}\left(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg}, \quad k_{obs,sig}\right) \\ &\times Gaussian_{pdf}(x = \lambda_{bkg}, \mu = \mu_{bkg}, \sigma = \sigma_{bkg} \approx \sqrt{\mu_{bkg}}) \end{aligned} \tag{9.1}$$

The goal of the analysis is to place bounds on the Poisson process which generates the "signal" or in this case, the ppp-decay. We must take the posterior and marginalize over all parameters except for the ones of interest. In this case everything is known except for two parameters: $\lambda_{sig}, \lambda_{bkg}$. The final bounds on half-life are direct functions of the bound we can place on λ_{sig}, which requires marginalizing over λ_{bkg}.

We can marginalize over the joint posterior, to get the posterior of interest in multiple ways. Marginalization is most commonly done using Markov Chain Monte

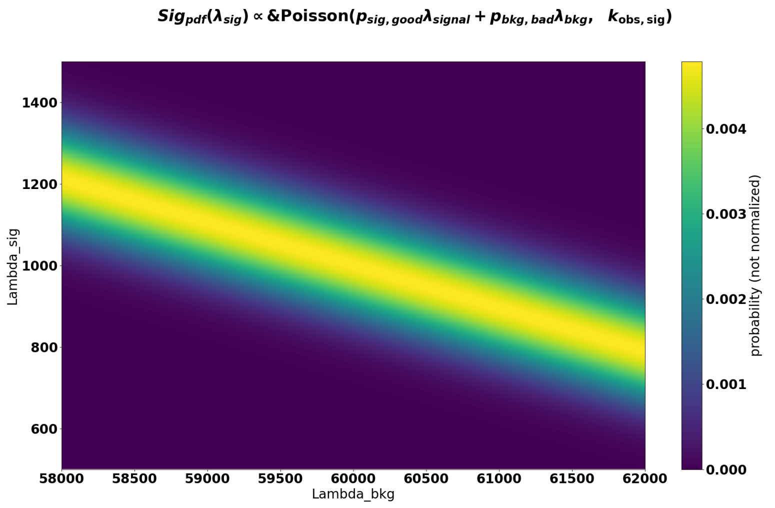


Figure 9.1 Example toy-data posterior, without using any prior information. Image constructed using toy-values: $p_{sig,good} = 0.95$, $p_{bkg,bad} = 0.1$, $k_{obs,sig} = 61000$. Our final result using actual observated events, and a carefully calculated muon prior, is shown later in Fig 10.1.

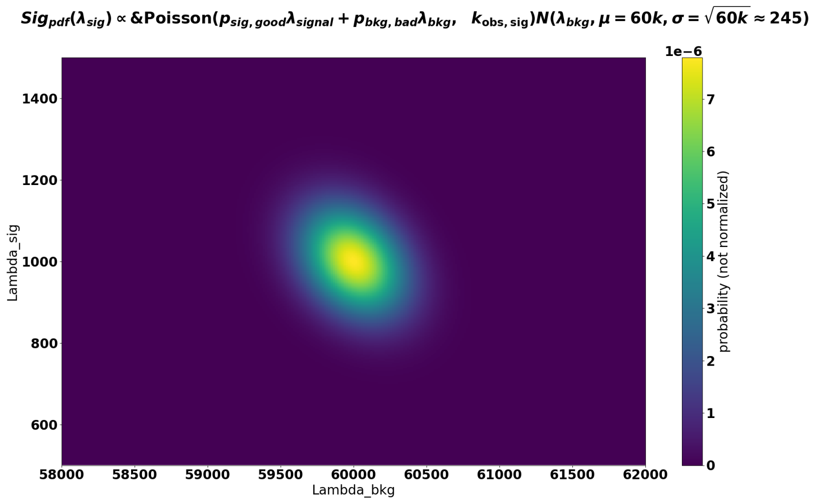


Figure 9.2 Example of a toy-data posterior produced by including the prior information about the number of muons we expect to see hit the detector. The image constructed using toy-values for a number of observed events, and a number of expected total muons. $p_{sig,good} = 0.95$, $p_{bkg,bad} = 0.1$, $k_{obs,sig} = 6.1 \times 10^4$, $\mu_{bkg} = 6 \times 10^4$ Our final result using actual observated events, and a carefully calculated muon prior, is shown later in Fig 10.1.

Carlo (MCMC) (Sec 9.1.1) (Fig 9.3). Alternatively we can marginalize over λ_{bkg} symbolically instead of computationally (Sec 9.1.2). A third approach is to approximate the marginalization with a **profile likelihood** (Sec 9.1.3).

9.1.1 Marginalization: MCMC

First, as a benchmark, we should marginalize the posterior using an MCMC algorithm. The posterior in equation 8.11 is sampled many times using the **emcee** algorithm. The MCMC technique produces sample values which asymptotically approach those drawn from the posterior distribution. The samples themselves can be plotted in scatter plots and histograms to visualize the shape of the distribution. Figure 9.3 is known as a **corner plot** and shows every possible histogram, as well as every possible 2D scatter plot for the samples drawn from the posterior distribution. Because we wish investigate the signal, we can inspect the histogram of λ_{sig} alone and consider it as the marginalized posterior.

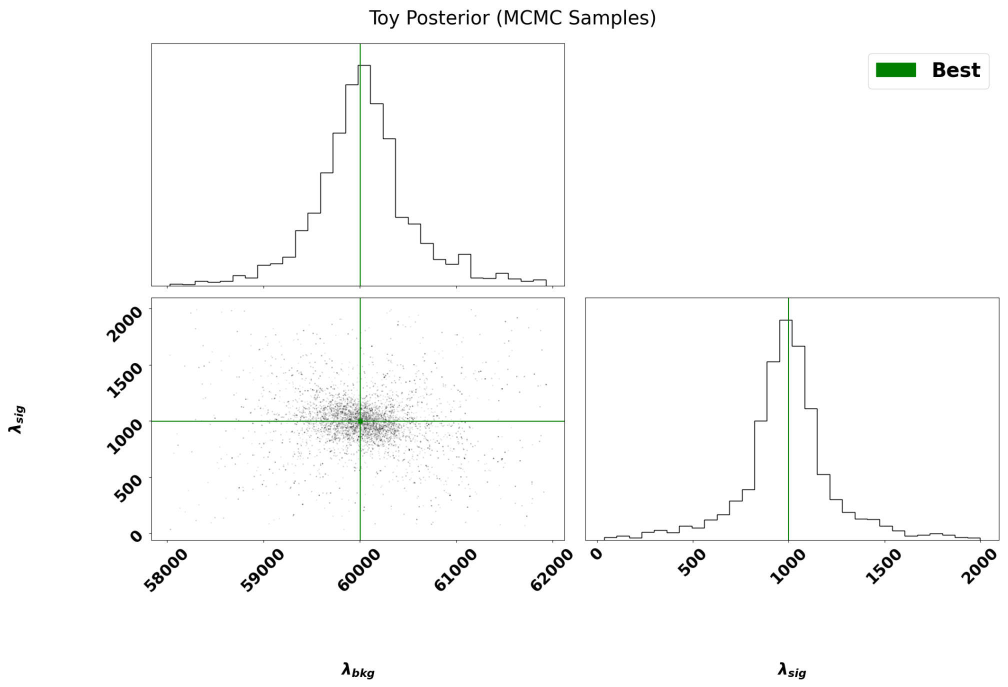


Figure 9.3 Toy-Posterior corner plot generated by sampling Eqn. 8.11 using the MCMC algorithm "emcee" [11]. Known values are: $p_{sig,good} = 0.95$, $p_{bkg,bad} = 0.1$, $k_{obs,sig} = 6.1 \times 10^4$, $\mu_{bkg} = 6 \times 10^4$

9.1.2 Marginalization: Algebra

Here we will go through the algebra to use to perform the integral of interest symbolically (without using MCMC numerical algorithms). We first setup the integral, then use the Laplace method to approximate a result. The symbolic form of the joint posterior is below (copied from Eqn 8.11):

$$\begin{aligned} SigPost_{pdf}(\lambda_{sig}) = \int d\lambda_{bkg} \mathrm{Poisson}\Big(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg}, \quad k_{obs,sig} \Big) \\ \times\, Gaussian_{pdf}(x = \lambda_{bkg}, \mu = \mu_{bkg}, \sigma = \sigma_{bkg} \approx \sqrt{\mu_{bkg}}) \end{aligned} \tag{9.2}$$

Redefine variables for more spatially efficient but less explicit notation:

$$\begin{aligned} p_1 \equiv p_{sig,good} &\equiv \text{probability to classify true signal correctly} \\ \lambda_1 \equiv \lambda_{sig} &\equiv \text{expected total count from signal Poisson process} \\ p_2 \equiv p_{bkg,bad} &\equiv \text{probability to classify true background incorrectly} \\ \lambda_2 \equiv \lambda_{bkg} &\equiv \text{expected total count from background Poisson process} \\ \mu \equiv \mu_{bkg} &\equiv \text{expected total count from background prior to CUORE} \\ \sigma \equiv \sigma_{bkg} &\equiv \text{std-dev total count from background prior to CUORE} \\ k \equiv k_{obs,sig} &\equiv \text{number of events seen as signal after classification} \end{aligned} \tag{9.3}$$

The integral of interest is now:

$$\begin{aligned} I &\equiv SigPost_{pdf}(\lambda_1) \\ &= \int_0^\infty d\lambda_2 \mathrm{Poisson}(p_1\lambda_1 + p_2\lambda_2, \quad k) Gaussian_{pdf}(\lambda_2, \mu, \sigma) \end{aligned} \tag{9.4}$$

The full symbolic integral using the definitions of Gaussian and Poisson is written:

$$I = \int_0^\infty d\lambda_2 \frac{(p_1\lambda_1 + p_2\lambda_2)^k e^{-(p_1\lambda_1 + p_2\lambda_2)}}{\Gamma(k+1)} \frac{e^{-\frac{1}{2}\left(\frac{\lambda_2 - \mu}{\sigma}\right)^2}}{\sigma\sqrt{2\pi}} \tag{9.5}$$

Bring constant terms outside the integral where possible:

$$I = \underbrace{\frac{e^{-p_1\lambda_1}}{\Gamma(k+1)\sigma\sqrt{2\pi}}}_{C} \int_0^\infty d\lambda_2 \Big(p_1\lambda_1 + p_2\lambda_2\Big)^k e^{-p_2\lambda_2} e^{-\frac{1}{2}\left(\frac{\lambda_2-\mu}{\sigma}\right)^2} \tag{9.6}$$

$$I = C\int_0^\infty d\lambda_2 \Big(p_1\lambda_1 + p_2\lambda_2\Big)^k e^{-p_2\lambda_2} e^{-\frac{1}{2}\left(\frac{\lambda_2-\mu}{\sigma}\right)^2} \tag{9.7}$$

Laplace Method

The Laplace's method requires an approximation, but we can estimate the error, and in the regime of interest for our calculation the error is small. The idea is to re-write the integrand entirely in the exponent labeled as $f(\lambda_2)$:

$$I = C\int_0^\infty d\lambda_2 \,\exp\Bigg(\underbrace{k\,\ln(p_1\lambda_1 + p_2\lambda_2) - p_2\lambda_2 - \frac{1}{2}\Big(\frac{\lambda_2-\mu}{\sigma}\Big)^2}_{f(\lambda_2)}\Bigg) \tag{9.8}$$

Write down f, f', f'' required for a second order taylor expansion of f:

$$f(\lambda_2) = -\lambda_2 p_2 + \log\left((\lambda_1 p_1 + \lambda_2 p_2)^k\right) - \frac{0.5\,(\lambda_2-\mu)^2}{\sigma^2} \tag{9.9}$$

$$f'(\lambda_2) = \frac{kp_2}{\lambda_1 p_1 + \lambda_2 p_2} - \frac{1.0\lambda_2}{\sigma^2} + \frac{1.0\mu}{\sigma^2} - p_2 \tag{9.10}$$

$$f''(\lambda_2) = -\Big(\frac{kp_2^2}{(\lambda_1 p_1 + \lambda_2 p_2)^2} + \frac{1.0}{\sigma^2}\Big) \tag{9.11}$$

Solve for the maximum value of $f(\lambda_2)$ by setting it's derivative equal to zero. Because we are solving a simple quadratic equation, there will be two solutions. The correct solution will be the positive one. We introduce the variable α to allow the solution to fit on one page.

$$\alpha \equiv kp_2^2\sigma^2 + 0.25\lambda_1^2p_1^2 + 0.5\lambda_1\mu p_1p_2 - 0.5\lambda_1p_1p_2^2\sigma^2 + 0.25\mu^2p_2^2 - 0.5\mu p_2^3\sigma^2 + 0.25p_2^4\sigma^4 \tag{9.12}$$

$$\lambda_{2,max+} = \frac{0.5\,(-\lambda_1p_1 + \mu p_2 - p_2^2\sigma^2 + 2.0\sqrt{\alpha})}{p_2} \tag{9.13}$$

Approximate f with a second order Taylor expansion about the max value:

$$f_{Taylor}(\lambda_2) \approx f(\lambda_{2,max+}) + 0 + \frac{1}{2}f''(\lambda_{2,max+})(\lambda_2 - \lambda_{2,max+})^2 \tag{9.14}$$

Substitute the Taylor series approximation back into the integral:

$$I \approx C \int_0^\infty d\lambda_2 \, \exp\Bigg(f(\lambda_{2,max+}) + \frac{1}{2}f''(\lambda_{2,max+})(\lambda_2 - \lambda_{2,max+})^2 \Bigg) \tag{9.15}$$

Pull out the zero'th order term of the expansion from the integral:

$$I \approx C e^{f(\lambda_{2,max+})} \int_0^\infty d\lambda_2 \, \exp\Bigg(\frac{1}{2}f''(\lambda_{2,max+})(\lambda_2 - \lambda_{2,max+})^2 \Bigg) \tag{9.16}$$

We integrate the Gaussian function which can be expressed in terms of the error function 'erf':

$$I \approx C e^{f(\lambda_{2,max+})} \frac{\sqrt{\pi}}{\sqrt{2}} \frac{1}{\sqrt{|f''(\lambda_{2,max+})|}} \Bigg(erf\Big(\frac{|\lambda_{2,max+}|\sqrt{|f''(\lambda_{2,max+})|}}{\sqrt{2}} \Big) + 1 \Bigg) \tag{9.17}$$

We can substitute in content for the constant C and simplify:

$$I \approx \frac{e^{-p_1\lambda_1} e^{f(\lambda_{2,max+})}}{\Gamma(k+1)\sigma\sqrt{2\cancel{\pi}}} \frac{\cancel{\sqrt{\pi}}}{\sqrt{2}} \frac{1}{\sqrt{|f''(\lambda_{2,max+})|}} \Bigg(erf\Big(\frac{|\lambda_{2,max+}|\sqrt{|f''(\lambda_{2,max+})|}}{\sqrt{2}} \Big) + 1 \Bigg) \tag{9.18}$$

After basic simplifications the result works out to be:

$$I \approx \frac{e^{-p_1\lambda_1 + f(\lambda_{2,max+})}}{2\sigma\Gamma(k+1)\sqrt{|f''(\lambda_{2,max+})|}} \Bigg(erf\Big(\frac{|\lambda_{2,max+}|\sqrt{|f''(\lambda_{2,max+})|}}{\sqrt{2}} \Big) + 1 \Bigg) \tag{9.19}$$

The final integration result (Eqn: 9.19) should be a good approximation, with very little computational cost for a final result. We can try out different parameters such as the classification probabilities p_1 and p_2.

A note on extending to multiple sources:

We can also apply Laplace's method for the higher dimensional version of this analysis. If multiple sources of background need to be integrated out, we can apply a multivariate Taylor series, before integrating. The multiple background Laplace approximation integrand will be a multivariate Gaussian, with a covariance matrix.

9.1.3 Marginalization: Approximation with Profile Likelihood

Instead of marginalizing at all (numerically or symbolically) instead we can find the best fit parameters and allow the only parameter of interest to vary. Fixing the nuisance parameters to their best fit values and allowing others to vary is called taking a "profile" likelihood. In a poisson analysis with classification, the likelihood of interest is rewritten below (Eqn: 8.11).

$$\begin{aligned} SigPost_{pdf}(\lambda_{sig}) = \int d\lambda_{bkg} \text{Poisson}\Big(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\lambda_{bkg}, \quad k_{obs,sig}\Big) \\ \times\, Gaussian_{pdf}(x = \lambda_{bkg}, \mu = \mu_{bkg}, \sigma = \sigma_{bkg} \approx \sqrt{\mu_{bkg}}) \end{aligned} \tag{9.20}$$

Our likelihood function only has two parameters: λ_{sig} and λ_{bkg}. We only have one nuisance parameter: λ_{bkg}. The best fit value for the nuisance parameter is obvious because it is entirely dependent upon our prior distribution, which in this case is a Gaussian centered at μ_{bkg}. Thus our profile likelihood amounts to plug in the best fit value into the likelihood at $\lambda_{bkg} = \mu_{bkg}$, and allowing the other parameter λ_{sig} to vary.

$$SigPost_{pdf}(\lambda_{sig}) \approx \text{Poisson}\Big(p_{sig,good}\lambda_{signal} + p_{bkg,bad}\mu_{bkg}, \quad k_{obs,sig}\Big) \tag{9.21}$$

Further, so long as the expected total observation count is sufficiently large, we can approximate the profile likelihood with a gaussian:

$$\begin{aligned} SigPost_{pdf}(\lambda_{sig}) \approx \text{Gaussian}\Big(\mu = (k_{obs,sig} - p_{bkg,bad}\mu_{bkg})/p_{sig,good}, \\ \sigma = \sqrt{(k_{obs,sig}}/p_{sig,good}, \lambda_{sig}\Big) \end{aligned} \tag{9.22}$$

Looking at the joint posterior function (Eqn 8.11), it is clear that the closeness of the profile approximation is a function of our confidence in the background signal. If we knew exactly what rate of background events we expect to observe, then we would constrain λ_{bkg} much more sharply (as calculated in Sec 7.4). Another way to interpret the approximation is to look at the Gaussian prior, and ask under which conditions it would become a delta function. In our particular case, because the standard deviation is the square root of the mean, in the limit that we had infinite background signal, the prior distribution would asymptotically approach a delta function.

In practice, as long as the amount of background we expect to see is large, (greater than a few thousand), the error introduced will be minimal. For a final result, we can always double check our answer using an overly cautious high sample count MCMC and make sure both solutions line up. For the sensitivity plot, the profile approximation is ideal, because it will be computationally very cheap, and get us an answer to within a few percent which is really the purpose of the sensitivity plot to begin with.

9.1.4 Comparison of Marginalization Techniques

Each marginalization technique has a different advantage. The MCMC solution is easy to implement and provides a reliable answer. The symbolic solution requires working through the algebra to get a result, but is then both computationally efficient and very accurate. The profile solution requires little algebra, and is computationally efficient, but runs the risk of being highly inaccurate in certain cases. It will turn out that in in the context of a sensitivity plot, the profile likelihood, will be the best choice.

We can compare the computational cost of the MCMC algorithm against those non-numerical techniques. Constructing the corner plot in Fig 9.3 takes 1 compute core unit, about 1.5 minutes, compared to a closed form result which would take less than a millisecond. While it may not seem like much of a difference in absolute terms, a difference of a factor of 10^6 is impactful because of the large number of times the marginalization needs to be done when assessing the sensitivity.

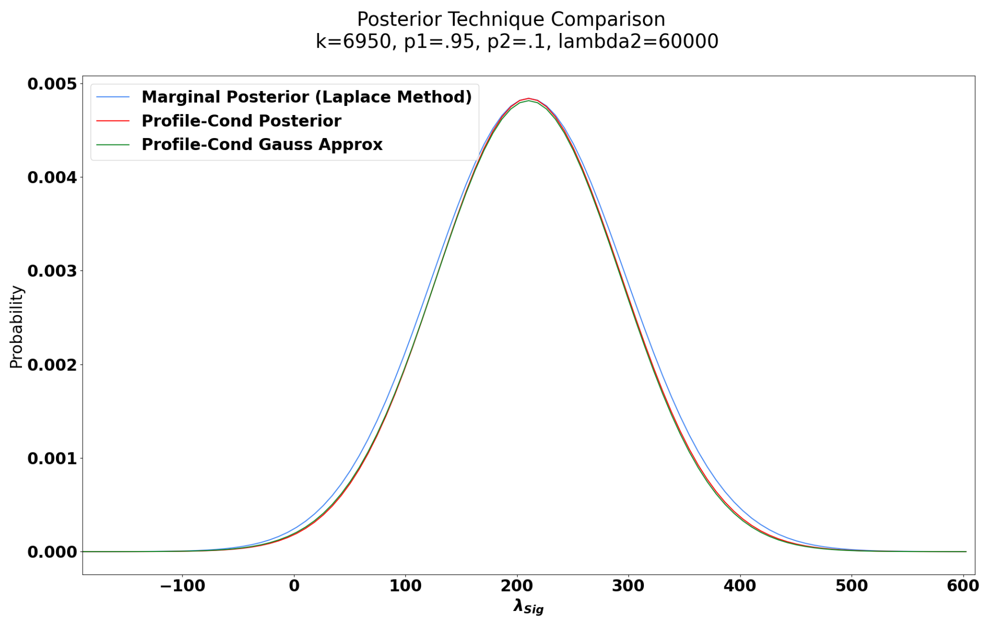


Figure 9.4 Comparison of marginalization techniques: Laplaces Method, Profile posterior, Profile Guassian Approximation. The comparison shown is for a fixed set of classification probabilities.

What we see from comparing the closed form techniques (Fig 9.4) is that the profile likelihood is very close to the Laplace method (which is much more accurate). Essentially, the discrepancy only appears in the tails of the distribution, which we are not concerned with for a sensitivity analysis. For the sensitivity analysis only, we ask: If $\lambda_{sig} = 300$, to what accuracy in standard deviations is that? We don't actually care about the probability density value. Thus using the profile likelihood, and approximating the it as a Gaussian is more than sufficient for a sensitivity analysis.

9.2 Calculate Sensitivity: Fixed Classifier Success Rates

In this work we will define the $n\sigma$ sensitivity to a half-life as the expectation that one would obtain an $n\sigma$ detection result in 50% of many copies of an experiment. In other words, if there were many copies of CUORE, if a ppp-decay actually existed, and noting we cannot classify muons vs ppp-decays perfectly, what $n\sigma$ lower λ_{sig} bound would we achieve in 50% of all the CUORE's?

First we will do the sensitivity calculation for a fixed set of classification success rates. We take our data and classify all the events. We take the events we have seen as signal knowing that some percentage of those were wrongly classified. We use the known rates of classifying successfully. We ask, given those success rates, to what value of expected signal rate would we achieve an n-sigma detection 50% of the time? To walk through the calculation, lets assume a single scenario, where we fix μ_{bkg}, $p_{sig,good}$, $p_{bkg,bad}$ (as shown in Fig 9.5). The other values of interest in the posterior (Eqn:8.11) are λ_{sig}, and $k_{obs,sig}$, and the probability itself P. We can ask: Given the fixed values how does the posterior probability change as λ_{sig}, and $k_{obs,sig}$ are allowed to vary?

9.2.1 MCMC Sensitivity Illustration

The sensitivity result is shown in illustration Fig 9.5, where many elements were drawn by hand for illustration purposes. The posterior mean will increase almost linearly as the number of observed counts goes up (shown in red). A 3σ lower bound can be found along this curve (shown in yellow). The place where the 3σ lower bound on λ_{sig} begins to be greater than zero (shown in orange), is the same thing as a 3σ sensitivity as compared to the null hypothesis.

Figure 9.5 indicates that we can achieve a 3σ detection, if we happen to see about 6450 events seen as signal after classification. Such a number of observations seen as signal would correspond to a best fit number of actual events which were signal to be 450 ($\lambda_{sig} \approx 450$). In our case we would have seen 450 total ppp-decays over 5 years of exposure. 450 events corresponds to a calculable half-life value based on the number of Te^{130} atoms in the detector, and the amount of time we were observing data to perform the analysis. The above image would show that we should expect the half-life to be on the order of 10^{25} years for a triple proton decay, to get 450 events. We would then say we have a 3σ sensitivity to the half life of 10^{25} years. So the entire

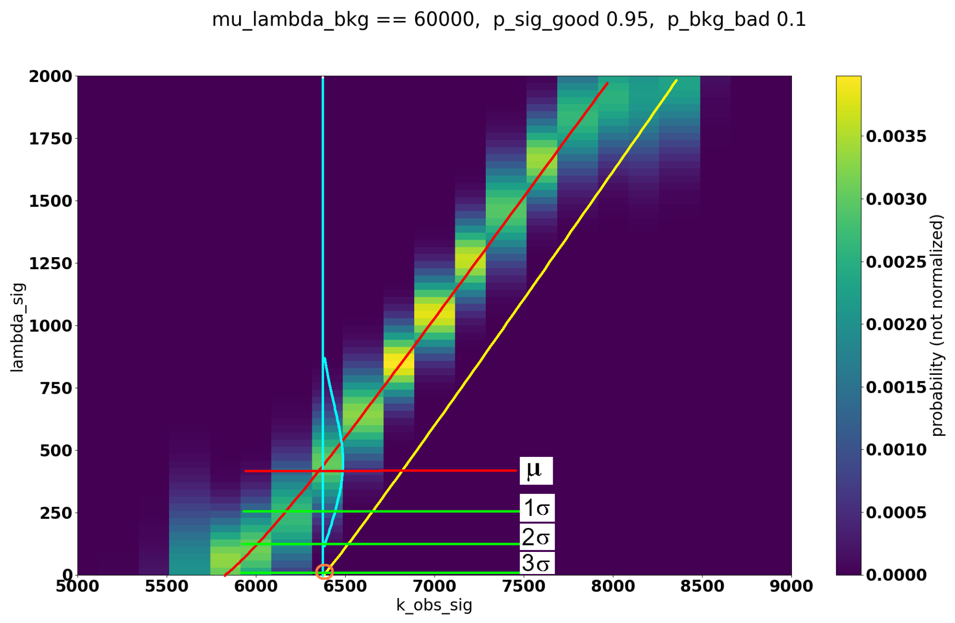


Figure 9.5 A sensitivity calculation illustration for fixed: $\mu_{\lambda_{bkg}} = 60,000$, $p_{sig,good} = 0.95$, $p_{bkg,bad} = 0.1$ Line segments: mu, 1σ, 2σ, 3σ were drawn by hand. The 3σ lower-bound was drawn by hand (drawn in yellow). The mean fit line $\mu_{\lambda_{sig}}$ was drawn by hand (drawn in red) Hand drawn elements were done for illustration purposes. The contour plot behind the lines was calculated numerically using a distinct run of an MCMC to marginalize the posterior for each bin of k_{obs}. The blocky nature of the image is an artifact of chosen bin widths.

illustration image (Fig 9.5) reduces to a 3σ sensitivity result of a single number.

While we could do the work to improve our illustration and instead achieve a real sensitivity result using MCMC, it is not a good use of energy and effort. The biggest problem with MCMC for a sensitivity plot is that it is too computationally inefficient. To construct the contour shown in Fig 9.5, twenty different runs of the MCMC algorithm had to complete and it took a full 100 seconds to run upon a single CPU core. Each instance of the MCMC required a few thousand samples. The image we produced after running an earnest computation attempt was still blocky and highly imprecise. To construct a single smooth copy of the image shown in Fig 9.5 we would like to do away with the bins on the $k_{obs,sig}$ axis and run on the order 1000 fixed values (factor of 100 more), and we should increase the number of MCMC samples within each run by an additional factor of 100. We should expect that producing a single high quality image would take on the order of 10^6 seconds. Thus we will accept that using MCMC to attain a sensitivity calculation is not computationally feasible. We will need another technique.

9.2.2 Profile Likelihood Smooth Result

We would now like to redo all of the logic shown in our illustration example (Sec 9.2.1), but now we want to use real numbers. Instead of using MCMC we can approximate the posterior using the profile likelihood. At each possible set of values of $k_{obs,sig}$ and λ_{sig} we can read a probability from Eqn 9.22, thus we can produce a high quality copy of the image directly. Further, we can use the form of the profile likelihood to obtain a line fit through the mean value, as well as line fits through the n-sigma lower bounds by reading the mean and sigma directly from Eqn 9.22:

$$\begin{aligned}\mu_{profile} &= (k_{obs,sig} - p_{bkg,bad}\mu_{bkg})/p_{sig,good} \\ \sigma_{profile} &= \sqrt{(k_{obs,sig}}/p_{sig,good}\end{aligned} \tag{9.23}$$

We can now plot any $n\sigma$ lower bound as follows:

$$\text{Lower bound} = \mu_{profile} - n\sigma_{profile} \tag{9.24}$$

And we can calculate the sensitivity by asking when the lower bound hits the $k_{obs,sig}$ axis of the figure (Fig 9.5).

$$\begin{aligned}\text{Lower bound} = \mu_{profile} - n\sigma_{profile} &= 0 \\ \mu_{profile} &= n\sigma_{profile} \\ (k_{obs,sig} - p_{bkg,bad}\mu_{bkg})/p_{sig,good} &= n\sqrt{(k_{obs,sig}}/p_{sig,good}\end{aligned} \tag{9.25}$$

We now only need to solve the quadratic equation for the $k_{obs,sig}$ where the line of the lower bound will intersect with the horizontal axis. The intersection point then corresponds to a particular mean value of λ_{sig}. We then say we have an $n\sigma$ sensitivity to particular value of λ_{sig}. The axis intersection point for where the lower $n\sigma$ bound does not have any dependence upon the classification probability for classifying signal correctly ($p_{sig,good}$). The profile likelihood approximation's lack of dependence upon $p_{sig,good}$ is surprising, but not a problem.

$$\begin{aligned} n &= 3 \;\;\text{(we wish to obtain a 3 } \sigma \text{ sensitivity)} \\ b &\equiv -2 * p_2 * 60000 + n^2 \\ k_{obs,sig,min} &\equiv (-b + \sqrt{b^2 - 4 * (p_2 * 60000)^2}/2) \end{aligned} \tag{9.26}$$

The sensitivity however definitely does still have dependence upon both signal, and background classification probabilities. This can been seen quickly by noticing that the intersection lower bound must still be substituted in to get the expected signal count to which we are sensitive.

$$\text{Sensitivity}_{counts} = (k_{obs,sig,min} - p_{bkg,bad} * 60000)/p_{sig,good} \tag{9.27}$$

Note we still must transform the count sensitivity to a half-life sensitivity using the transformation shown in Eqn (5.37). The transformation from count total back to half-life is omitted in this section.

9.3 Calculate Sensitivity: Vary Classifier Success Rates

We now wish to try many possible values for each of $p_{sig,good}$ $p_{bkg,bad}$. If we used MCMC to produce the plot without any parallelization, we would require at least $100sec \times 10^6 = 10^8 sec \approx 3years$ to get a high resolution sensitivity plot. Instead we can plan to make the same plot, more accurately, in a few seconds using the closed form profile likelihood approximation. We can show how the sensitivity changes as a function of the two probabilities. After viewing the plot it becomes clear that reducing background confusion is more valuable to a final result than improving signal classification. Such a statement is consistent with the figure of merit calculated earlier in Sec 8.2

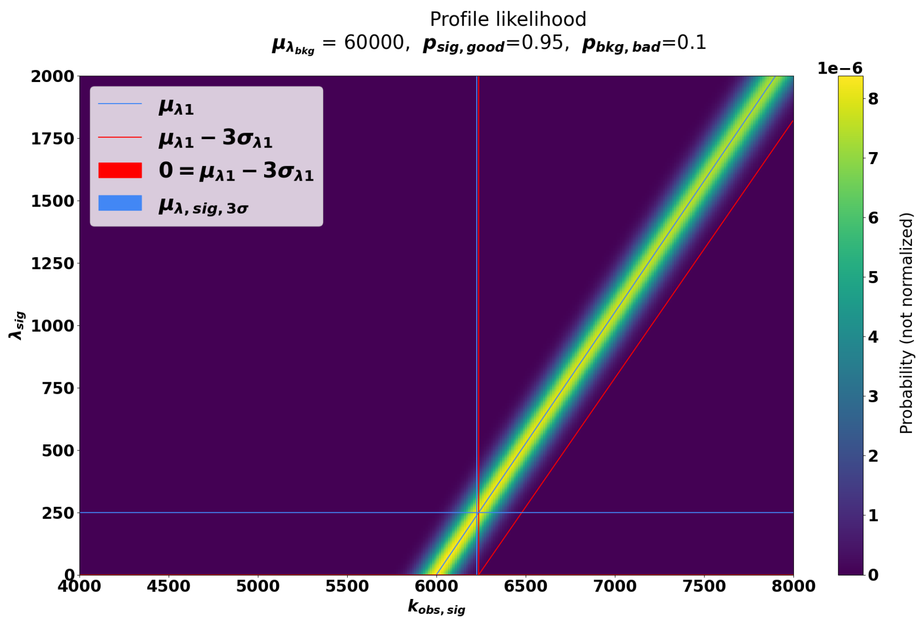


Figure 9.6 A sensitivity calculation for fixed: $\mu_{\lambda_{bkg}} = 60,000$, $p_{sig,good} = 0.95$, $p_{bkg,bad} = 0.1$ The color plot results from Eqn 9.22. The $n\sigma$ lower bound line (red) is calculated symbolically (Eqn: 9.25). The intersection line segments represent the lowest λ_{sig} sensitivity calculated from intersecting μ_{λ_1} with the $k_{obs,sig}$ axis. (Eqn 9.26).

The takeaway from our exercise in sensitivity is that we can use the plot to determine how well we have to do to be competitive with existing literature. We need to obtain a result on the order of $10^{25} yrs$ to compete with previous searches of triple proton decay. We can achieve such a result using the CUORE detector and a machine learning algorithm which can classify correctly enough to land us in a good region of our sensitivity plot.

We should take the lower left corner of the plot with a grain of salt, because in practice we would not bother to use a classification algorithm that performed worse than a coin flip for both signal and background. In fact, as a sanity check, if we do fix the classification probabilities to both being 50%, then we should arrive at a result similar to the counting statistics result we already derived without classification (Sec 7.3).

In practice we should expect a much better result than using counting statistics

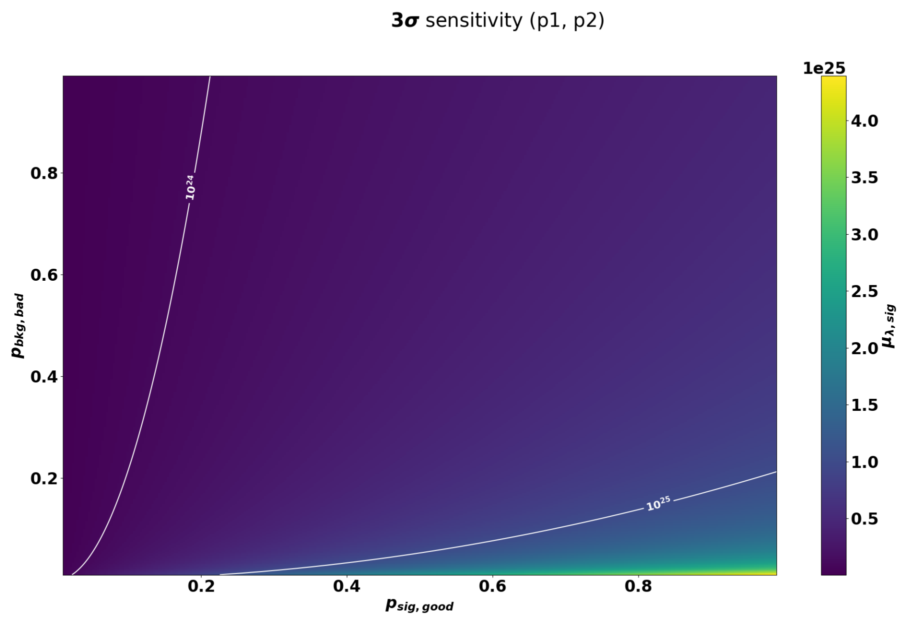


Figure 9.7 A sensitivity calculation for fixed: $\mu_{\lambda_{bkg}} = 60,000$. The classification probabilities: $p_{sig,good}$, $p_{bkg,bad}$ are allowed to vary. Each pixel on this plot maps to a copy of the sensitivity calculation machinery behind Fig 9.6. Upon the mostly-blue density plot, contours are shown as white-lines for specific half-life Sensitivities.

alone. We can obtain at least a 85% success rate on classifying both signal and background, which will land us squarely on the correct side of the 10^{25} year 3σ contour for 5 years of CUORE exposure. One can always improve improve classification probabilities even further by tweaking our machine learning algorithms and attempt to push the result to the furthest lower right hand side of the sensitivity plot.

Chapter 10

Results and Conclusion

10.1 Summary of Results

In this section we outline how we obtained a 2σ lower-bound of ${}^{130}Te$ Triple-Proton-Decay half-life of $7.43 \times 10^{24} yrs$. We obtain the result from using the posterior shown in Eqn. 8.11. To achieve a final result, we combined observed real data count values, an expected muon count (Sec 2.3), and classification success rates. After obtaining a final joint posterior, we marginalize it over the background nuisance parameter to obtain a distribution of expected observed signal count rate. Finally, we transform the expected observed signal count rate distribution into a half-life distribution which is used obtain a bound.

10.2 Posterior Classification Probabilities

To obtain a final result we are using a machine learning random forest model to classify real events. We showed the classification success rates in Fig 6.12. We showed what the machine learning algorithm believed about the real data in Fig 6.14. We will assume that the machine learning algorithm has done its best job using the maximum level of training samples, which corresponds to the last datapoint on each figure, and has a training result confusion matrix shown in Fig 6.13. Thus we can read off that the classification success success probabilities are set to $P_{sig,good} = 0.88$ and $P_{bkg,bad} = 0.15$. Of the 2364 real observed events, the algorithm believed 386 events looked like PPP-decays, and 1978 looked like muons. Thus using notation from our posterior equation (Eqn 8.11) the total number of events we think we see as signal are $k_{obs,sig} = 386$

10.3 Expected Muon Count vs Real Observation Count

There is general agreement between the number of muons we expect to see (2537), and the number of total muon candidate events observed by CUORE (2364) during the 188 days of time-exposure, with 200kg of ^{130}Te. We have used a directionally averaged muon flux number from the Borexino collaboration to calculate expected count. The CUORE observed muon candidate event count is in agreement with the expected range.

To improve the expected total muon count prior, one would need the expected muon flux, as a function of direction and energy. One could calculate a more accurate expected CUORE observed muon count prior using data from MACRO, LVD, and Borexino combined, and modify the corresponding CUORE qshields simulations accordingly. [34, 35, 24, 21]. Approximating a closed form symbolic expression representing the direction and energy dependent muon flux would be required.

10.4 Marginalize the Final Posterior

A final posterior has been achieved substituting in machine learning training results, and substituting in the calculated muon prior distribution. We now have a joint probability density function of λ_{sig} and λ_{bkg}. As the only desired final result is that of ppp-decay, we must marginalize over the CUORE's observed muon counts. To perform the marginalization, we use an MCMC algorithm to explore the posterior joint density function. We can see the final MCMC result in Figures [9.3 10.3, 10.2]. From the MCMC samples, we find that 95% of them lie above our lower-bound of ^{130}Te triple proton decay half-life of $7.43 \times 10^{24} yrs$, and thus is a 2σ lower bound.

We can also report a 90% confidence level or 1.64σ result lower half-life bound is $7.91 \times 10^{24} yrs$.

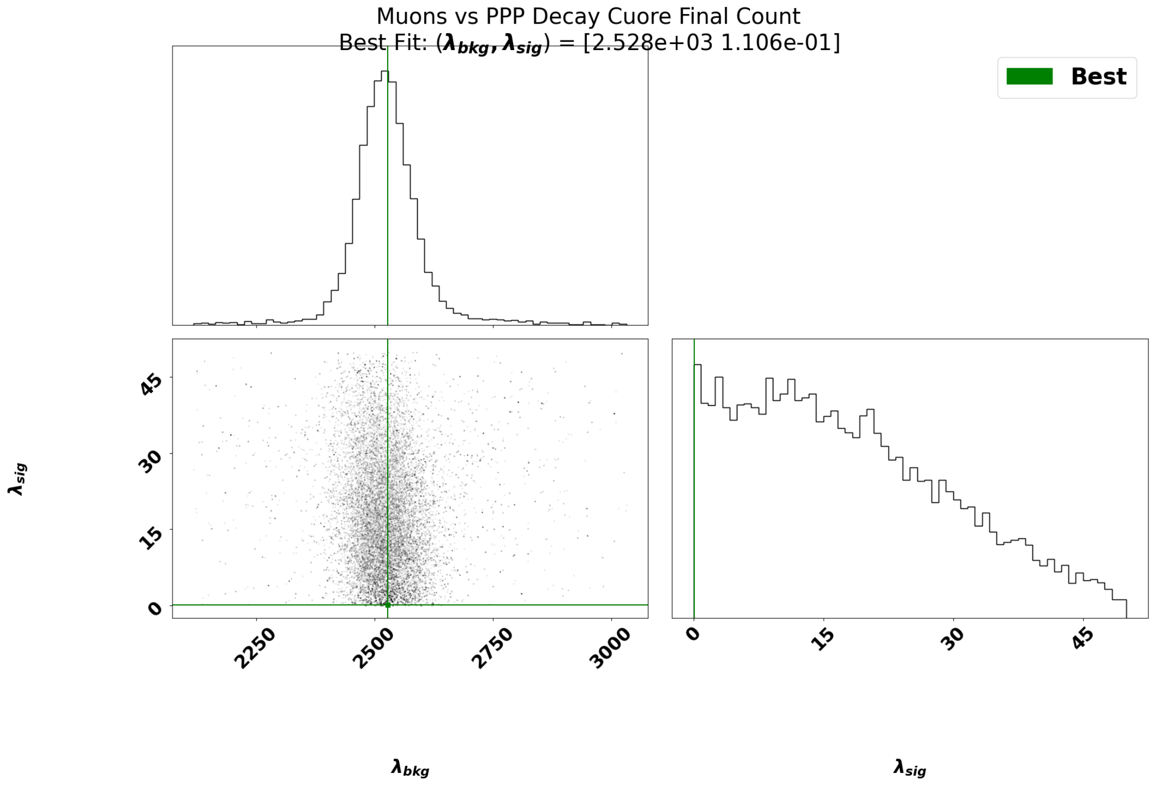


Figure 10.1 Corner plot of our posterior distribution. (Eqn. 8.11) The possible values of λ_{sig} and λ_{bkg} are fed through an MCMC algorithm (emcee) [11], using a half-Ton Year's worth of real data from the CUORE experiment.

10.5 Conclusion

Using previously built computational tools as arranged in Fig 1.5, we built a workflow which can search for signal against background using machine learning classification algorithms. We chose features of single events which can be used by a classification algorithm to differentiate different kinds of events within the detector. We combined Poisson counting statistics, with classification success probabilities to obtain a joint Bayesian likelihood analysis. From a closed form joint likelihood function we obtained an $n\sigma$ sensitivity calculation with various levels of approximation when marginalizing nuisance parameters. We used our general poisson counting classification framework,

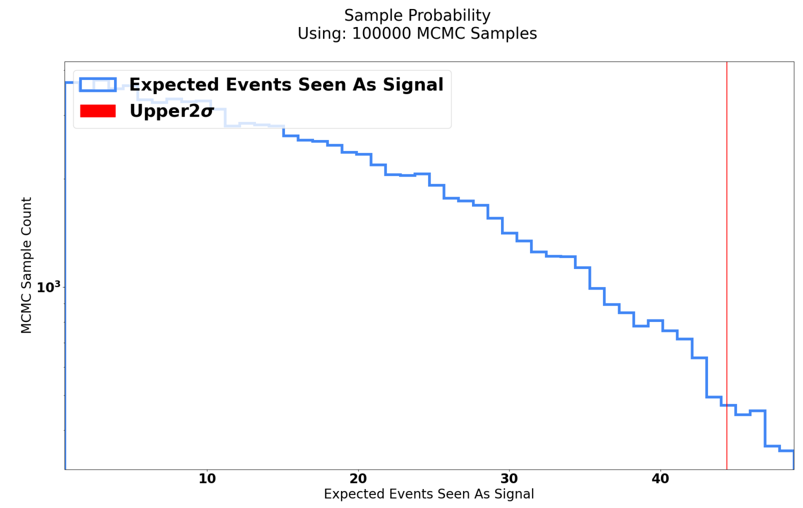


Figure 10.2 Marginal posterior distribution plot of the number of the expected total ppp-count λ_{sig} over the true exposure time of the real data we observed. These are the same trial values of λ_{sig} which were attempted by an MCMC algorithm (emcee) [11], shown in Fig 10.1. The result was achieved using the 188 Days of exposure time covered in datasets [3612,3613, 3614, 3615] the CUORE experiment (Sec 4).

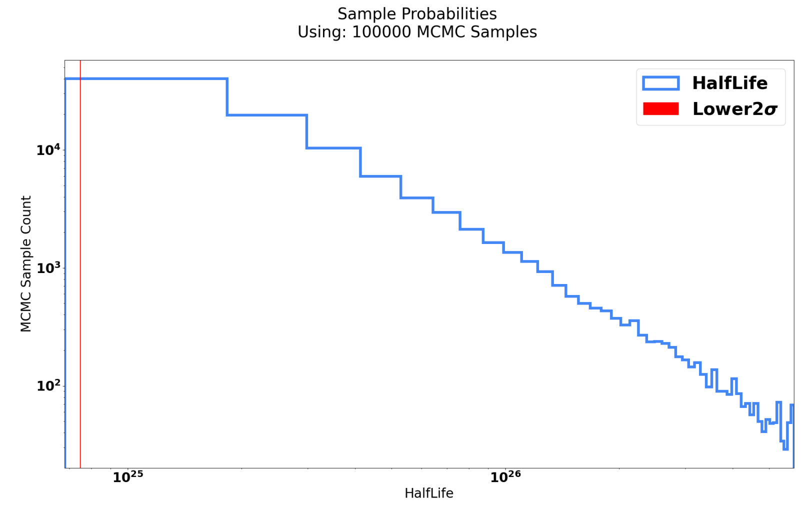


Figure 10.3 Marginal posterior distribution plot of the half-life. Lower 2σ bound shown in red at $7.43 \times 10^{24} yrs$. Each of the samples used to generate this histogram are the same as those in the MCMC figures 10.1 and 10.2. Each possible sample value for λ_{sig} maps to a corresponding half-life of ppp-decay. Shown here are possible half-life values sampled according to how likely they are. Mathematically, this histogram can be interpreted as the right-tail of a Gaussian distribution that has been cut off at zero. Thus, a lower 2σ bound has meaning, and an upper 2σ bound has no physical meaning.

to perform a search for triple proton decay (ppp-decay) against a background of atmospheric muons in the CUORE experiment.

We obtained a half-life lower-bound of $7.43 \times 10^{24} yrs$ at 2σ or 95% confidence level. Our ppp-decay half-life lower bound result is comparable to existing results. The Majorana Demonstrator [17] obtained a 90% confidence $5 \times 10^{25} yr$ lower bound ppp-decay for Germanium. The EXO200 collaboration [18] obtained a 90% confidence $2 \times 10^{23} yr$ lower bound ppp-decay for Xenon. Direct comparison of these different ppp-decay results is complicated. Given the ppp-decay theory each kind of atom nucleus has it's own decay calculation. Any combination of 3 protons within a nucleus has a probability of decay however simple combinatorics and Feynman diagrams are not applicable. Nuclear matrix elements must be calculated for each kind of atom separately.

The result could be further improved through the use of more data exposure time or better classification optimized with a figure of merit. Our data exposure-time was limited by the amount of cleaner data after Mar 2020, which did not show any signs of contamination. CUORE is actively investigating the data contamination, and upon completion the same analysis used in this work can be run upon at least twice as much total data to improve the result. Our classification algorithm performance was limited by the current ability of well accepted algorithms to classify single events. The classification performance additionally was not optimized to make use of a figure of merit, which would favor misclassifying signal over misclassifying background.

In the pursuit of a ppp-decay bound, we developed a generic framework (Fig 1.5) which can be used by CUORE, to search for other models which may produce high-energy high-multiplicity events. Additionally, technique of combining both counting statistics and machine learning classification success rates may also have application for other underground counting experiments.

Appendix A

Muon Count Prior - Additional Calculations

Here we consider calculation details regarding the total number of muons we expect to hit CUORE over a regular time period. We assume that the muons hit CUORE at an average regular rate that must be compatible with flux measurements reported at LNGS by the MACRO [34] and Borexino [24] experiments. We use the CUORE qshields simulation software in conjunction with the accepted LNGS muon flux to calculate an expected CUORE observed muon count rate. While the CUORE qshields simulation software is only capable of starting muons on the half-sphere, we will show methods for simulations that start muons on a simple flat disk, as well as the half-sphere.

A.1 General Calculation of Expected Count

To calculate expected count for an experiment given an surface of initial particles, we need to define flux, and hit probability. A surface can have varying flux and varying probabilities of hitting the detector at different initial locations. We define associated functions $flux()$ and $hitprob()$ such that they integrate to intuitive averages as follows:

$$\frac{\text{Count Through Surface}}{\text{Time}} = \int_{Surface} dA\ flux(location) \tag{A.1}$$

$$\text{Average Hit Probability} \times Area = \int_{Surface} dA\ hitprob(location) \tag{A.2}$$

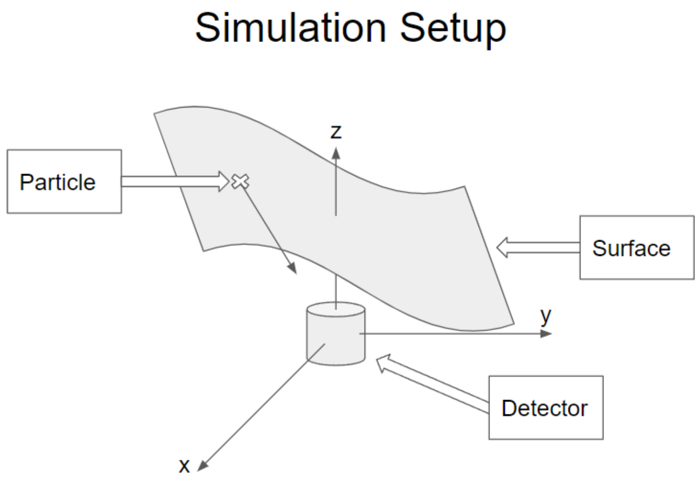


Figure A.1 General simulation setup. Particles are started upon a surface according to some joint distribution of location and direction. A detector sits under the surface. Each particle started upon the surface travels downwards and has a chance to hit or miss the detector.

The total number of particles we expect to hit a detector can be intuitively realized as the product of the time, flux, hit-probability, and area. Because we have non-constant flux and hit-probability, an integral is required:

$$Count = Time \times \int_{surface} dA \; flux(location) \; hitprob(location) \tag{A.3}$$

A.2 Initial Location Setups to Expected Counts

Here we consider how to calculate the expected count for a few simulation scenarios. We will consider a hypothetical simple disk model (A.2.1) as well as the non-uniformly distributed half-sphere (sec A.2.2) model used by the CUORE qshields simulation software. As the initial location surface becomes more difficult so does the calculation.

A.2.1 Initial Locations - Simple Disk (hypothetical)

A simple disk is placed on top of CUORE. Muons are spawned by the simulation from within the disk. The initial locations are chosen to be uniformly random upon

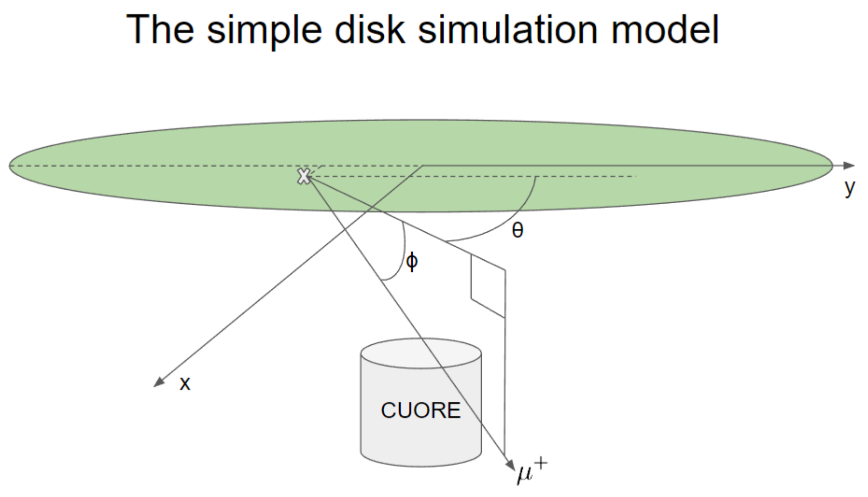


Figure A.2 A simple disk initial condition. Muons are started upon the disk with uniformly random location probability. They are started with directions matching the directional distribution measured inside the LNGS laboratory.

the disk. The initial directions emulate nature, are non-uniform, and the simulation accounts for the proper directional-distribution.

A disk at height $h = 5$ meters above CUORE with a radius of $r = 5$ meters should be sufficient to provide a reasonable estimate of total hit count. As the disk becomes larger, the muons at the edge of the disk have a lower and lower probability of hitting CUORE. If a muon started at the edge of the disk has zero probability of hitting CUORE then the disk radius is sufficiently large for reasonable calculations. Inside LNGS, muons do not have inclination angles larger than about 45 degrees. The maximum disk radius required is approximately equal to the height of the disk above the CUORE detector ($h = r = 5$). The choice of 5 meters for the height of the disk above CUORE does not have a mathematical justification.

We can start with the general count equation (Eqn. A.3) to proceed:

$$\begin{aligned} Count &= Time \times \int_{disk} dA \ \underbrace{flux(location)}_{constant} \ hitprob(location) \\ &= Time \times (\text{Constant Muon Flux}) \underbrace{\int_{disk} dA \ hitprob(location)}_{\text{Area}\times\text{(Avg Hit Prob)}} \end{aligned} \tag{A.4}$$

Thus the total number of muons we expect to hit CUORE from a disk simplifies to a simple product of constant flux, total area, time, and average hit-probability:

$$Count = (\text{Constant Muon Flux}) \times \text{Time} \times \text{Area} \times (\text{Average Hit Probability}) \tag{A.5}$$

The "constant muon flux" is the LNGS-Borexino reported value of $3.41\times10^{-4}m^{-2}s^{-1}$. The "average hit probability" is the fraction of the total number of simulated muons created upon the disk which hit CUORE and deposit a measurable amount of energy (this number includes any dependence upon energy cuts which are also applied to real data). The time is the exposure duration considered in seconds. Finally the "Area" is simply the total surface are of the disk ($Area = \pi r^2$).

$$Count_{\text{disk}} = (3.41 \times 10^{-4}m^{-2}s^{-1})(188Days)(78.5m^2)\frac{\text{(Number of Hits)}}{\text{(Number of Simulations)}} \tag{A.6}$$

A.2.2 Initial Locations - Half-Sphere (CUORE qshields)

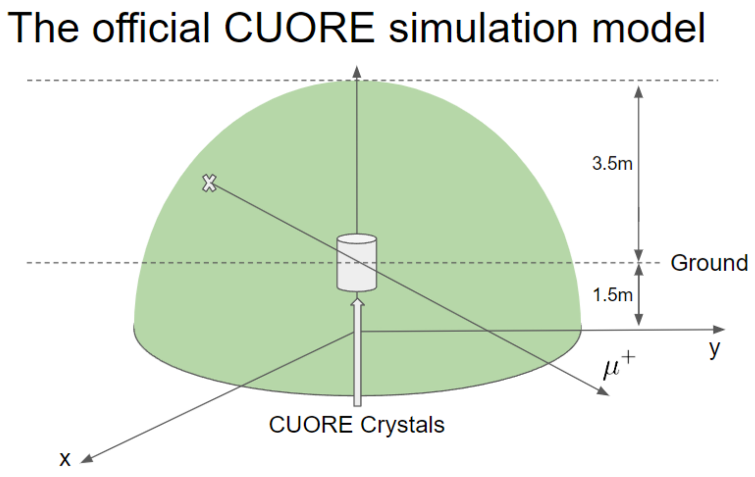


Figure A.3 The CUORE qshields half sphere simulation model. Muons are started upon the half-sphere with non-uniform location probability. Their initial direction emulates that of LNGS but disallows the possibility of being directed outside the half-sphere. We chose a radius for the half-sphere of 5 meters. The half-sphere is downshifted from the center of CUORE by 1.5 meters.

A half-sphere is centered 1.5 meters below CUORE crystals. The qshields simulation allows a choice of radius. We have been advised by collaboration members to use a 5 meter radius. The distribution of initial location and direction of the muons started on the half-sphere are intended to match the angular distribution of LNGS reported by the MACRO and Borexino collaborations.

To calculate expected count we need the general equation (Eqn. A.3):

$$\begin{aligned} Count &= Time \times \int_{half-sphere} dA \ \underbrace{flux(location)}_{\text{not constant}} \ hitprob(location) \\ &= Time \int_0^{2\pi} \int_0^{\pi/2} R^2 sin(\theta) d\theta d\phi \ flux(\theta,\phi) \ hitprob(\theta,\phi) \end{aligned} \tag{A.7}$$

Where $flux(\theta, \phi)$ is the muon flux as a function of start location on the surface of the half-sphere. At the top of the sphere, $flux(\phi = 0)$ should equal the reported LNGS-MACRO flux of $3.41 \times 10^{-4} m^{-2} s^{-1}$. The "$hitprob()$ function is the probability that a given muon starting at a particular location half-sphere hits CUORE (Eqn A.2).

A.2.3 Approximate Count Integral with Simulation Samples

We do not have explicit closed form symbolic expressions for the two functions of $flux()$ and $hitprob()$ functions. Instead we have two sets of samples that are representative of the two functions generated by the CUORE qshields simulation code. Let us denote the set of initial-location samples as "IL Samples" (Fig A.8), most of which will never hit CUORE. The initial-location samples can be used to approximate the muon flux. Let us denote the set of hit location samples as "HL Samples" (Fig 2.10) all of which have hit CUORE. The hit-location simulation samples can be used to approximate the hit-probability. We will describe and use the same nearest neighbors method to approximate both functions from their corresponding sample sets.

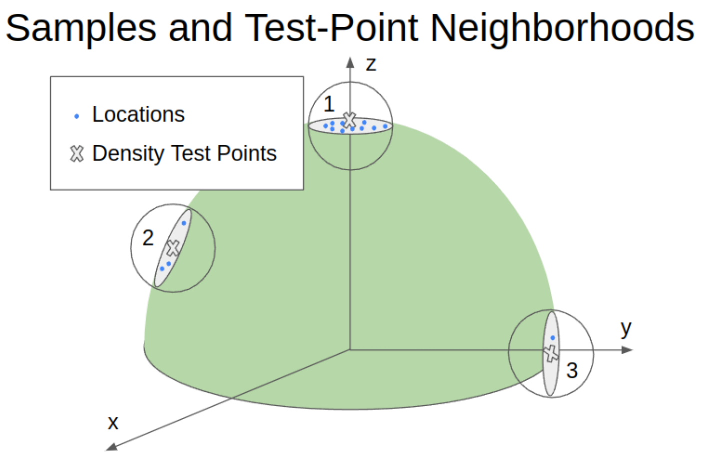


Figure A.4 A set of samples lie upon a half-sphere. Test point (1) is created exactly upon the top of the half-sphere. Separately, additional test points are generated at uniformly random locations upon the half-sphere (2). Cases like test point (3) are problematic if they include area below the half-sphere. The samples, and a list of test point locations, can be used to calculate an effective fraction estimate.

The nearest neighbor method approximates a density function at any given test point using neighboring samples. At any given test point (labeled 2 in Fig A.4), we can count the number of location neighbors within some sphere of constant radius. The density at the test point location will be proportional to the number of neighbors.

In the example figure, test point (2) contains 3 neighbors, whereas the top-sphere test point contains 10. Thus assuming the samples were correctly generated, we would expect the density at test point (2) to be equal to 3/10 of test point (1). We do however need be careful that test points close to the bottom edge of the half-sphere are not under-counting neighbors (test point 3 in Fig A.4). For test point locations like (3) either a missing area fraction must be calculated, or such points must be avoided entirely. To account for cases like location 3, we can avoid any test points lower than a certain height and choose a small radius for neighbor counting.

It is convenient to define an "Average Neighbor Count" as average number of sample neighbors for uniformly random generated test points upon the half-sphere. Let x_n be a uniformly randomly generated test point on the half-sphere (e.g. location 2 in Fig A.4). Let *SampleSet* be a set of samples with which we would like to begin counting neighbors. Let $NeighborCount(SampleSet, x)$ be a function that returns the count of how many samples from *SampleSet* are neighbors to the test point x. Then the average neighbor count can be calculated using the following arithmetic mean:

$$\text{Average Neighbor Count} = \frac{1}{N}\sum_{n=1}^{N} \text{NeighborCount}\ (SamplesSet, x_n) \tag{A.8}$$

To estimate the muon flux using the IL samples, we treat the top of the sphere (labeled 1 in fig A.4) as a special location (x_{top}) that matches the reported LNGS flux number. The neighbor count for a test point lower down on the half-sphere (labeled 2 in Fig A.4) will be some number less than that of the LNGS reported value. For any given test point we can compare its neighbor count against the top-sphere neighbor count.

$$flux(x) = \underbrace{(\text{LNGS Muon Flux})}_{3.41\times10^{-4}m^{-2}s^{-1}} \frac{\text{NeighborCount}(Samples_{IL}, x)}{\text{NeighborCount}(Samples_{IL}, x_{top})} \tag{A.9}$$

To estimate the hit probability using the HL samples, we can define the probability of a hit at a particular location in terms of a ratio to the average hit probability.

$$hitprob(x) = \underbrace{\left(\frac{\text{HL Samples Hit Count}}{\text{HL Samples Start Count}}\right)}_{\text{Average Hit Probability}} \frac{\text{NeighborCount}(Samples_{HL}, x)}{\text{NeighborCountAvg}(Samples_{HL})} \tag{A.10}$$

We can now combine our $flux(), hitprob()$ nearest neighbor approximations with the integration (Eqn. A.7) to obtain a result using the samples. Instead of integrating with symbols, we can use a summation over many test points. An integration over a surface, is equivalent to averaging uniformly distributed test-locations and multiplying by the area.

$$I \equiv \int_{surface} dS\ f(\mathrm{x}) \approx \frac{\text{Area}}{N} \sum_{n=1}^{N} f(\mathrm{x}_n) \tag{A.11}$$

Next we can apply the sum approximation to our integral of interest:

$$\begin{aligned} \text{Count} &= \text{Time} \int_0^{2\pi} \int_0^{\pi/2} R^2 sin(\theta) d\theta d\phi\ flux(location)\ hitprob(location) \\ &\approx \text{Time} \times \frac{\text{Area}}{N} \sum_{n=1}^{N} flux(x_n) hitprob(x_n) \end{aligned} \tag{A.12}$$

Using Eqn. A.12 we can find an expected count for the CUORE detector using our two sets of samples. The method finds an expected count of 5144 muons compared to 2364 events observed. Because there appears to be large discrepancy when using the nearest neighbors method on sample sets generated with CUORE qshields, we have opted for the simpler upper sphere surface method outlined previously (Sec 2.3), which predicts a closer to observed count of 2537 expected muons.

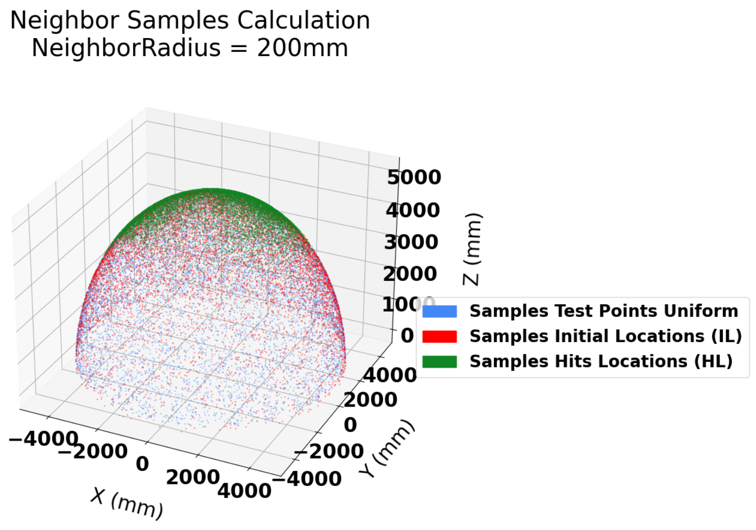


Figure A.5 10,000 IL Samples (red), and 10,000 HL Samples (green), are generated by the CUORE qshields simulations upon a half-sphere. Separately, we generated 10,000 test points (blue) uniformly randomly upon the same half-sphere.

A.3 Qshields Simulation Initial Conditions (Additional Plots)

In this section we show additional plots of CUORE-qshields software's initial conditions of simulated muons. The initial locations and directions are non-uniform upon the surface of a half sphere in such a manner intended to match conditions at LNGS. The initial directions only allow for muons to travel inwards from an initial location upon the half-sphere. The initial energies are a negative exponential function which visually match those reported by the MACRO [34] and Borexino [24] experiments. The CUORE qshields code comments have additional reference to a Particle Data Group (PDG) paper [25]. Figures in this section are provided as reference for how the CUORE qsheilds code creates initial muons before any particle-interaction.

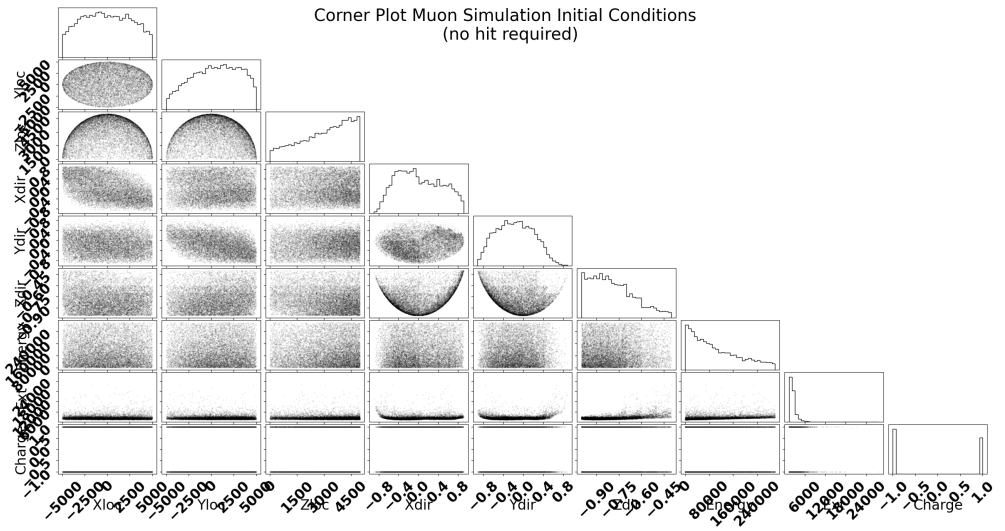


Figure A.6 Corner plot [8] of the initial conditions of muons spawned by the qshields software. Initial muon locations are non-uniform on a half-sphere. Directions are intended to be those that emulate conditions at LNGS, however the cases where muons point outwards from the half-sphere is disallowed by the simulation software.

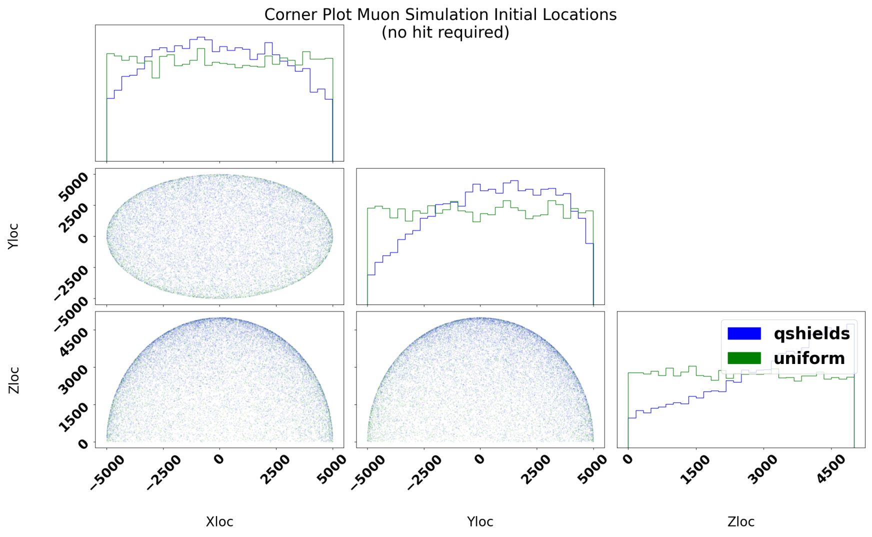


Figure A.7 Corner plot [8] of the initial locations of muons spawned by the qshields software. Initial muon locations are non-uniform on a half-sphere.

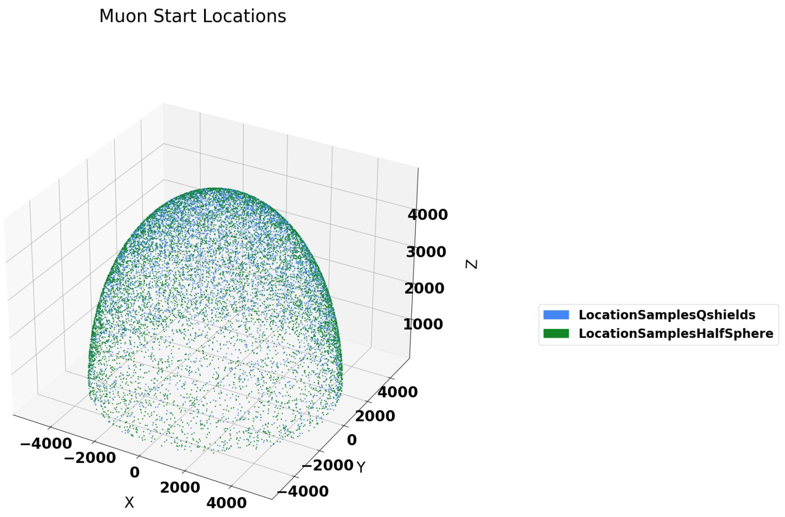


Figure A.8 Scatter plot of the initial locations of muons spawned by the qshields software. Initial muon locations are non-uniform on a half-sphere.

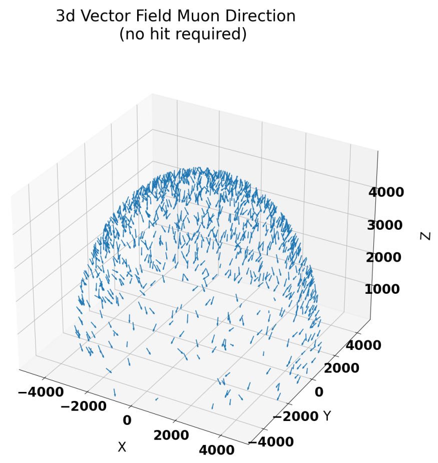


Figure A.9 Scatter plot of the initial direction arrows of muons spawned by the qshields software. Directions are intended to be those that emulate conditions at LNGS, however the cases where muons point outwards from the half-sphere is disallowed by the simulation software.

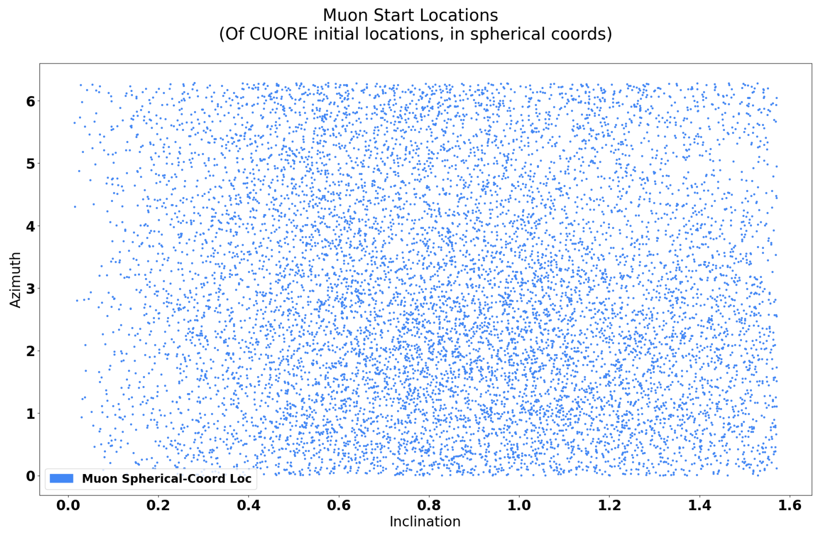


Figure A.10 Scatter plot of the initial direction for inclination and azimuth of muons spawned by the qshields software. Directions are intended to be those that emulate conditions at LNGS, however the cases where muons point outwards from the half-sphere is disallowed by the simulation software.

Appendix B

Basic Distributions

Basic distributions are written here for reference. Descriptions their variables is provided within the context of their most common usages. The distributions introduced here will be needed to perform a likelihood analysis. Of specific note to the reader, is that standard convention has λ describe different values in the context of different distributions. To perform the distinction between different λ's we can use different subscripts.

An **Exponential Distribution** commonly describes the situation where we are waiting for an event to happen and want to account for how long we have to wait until it does happen. Let t be the delay time until an event happens and λ_t be the expected amount of time until an event happens.

$$Prob(t) = Exp_{pdf}(\lambda_t, t) = \frac{1}{\lambda_t} e^{-\frac{1}{\lambda_t} t} \tag{B.1}$$

A **Binomial Distribution** commonly describes the situation where there are repeated trials of a repeatable experiment which can only have 2 outcomes $A \& B$. The probability p is that of outcome A, and n is the number of total times we run the experiment. Finally, k is the number of times that the experiment produced outcome A.

$$Prob(k) = Binomial_{pdf}(k, n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{B.2}$$

A **Poisson Distribution** commonly describes how many total events occur within a fixed time window, and k is the counted number of events, while λ_c is the number we expect to count.

$$Prob(k) = Poisson_{pdf}(\lambda_c, k) = \frac{\lambda_c^k e^{-\lambda_c}}{k!} \approx \frac{\lambda_c^k e^{-\lambda_c}}{\Gamma(k+1)} \tag{B.3}$$

A **Gaussian Distribution** is the most commonly used distribution, and is widely used to describe arbitrary random variables. It is used to describe measurements of a continuous quantity measured with repeated experiments. The quantity k is the value measured once, μ is the value we expect to measure, and σ is the standard deviation of the value measured.

$$Prob(k) = Gaussian_{pdf}(k, \mu, \sigma,) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{k-\mu}{\sigma}\right)^2} \tag{B.4}$$

Appendix C

Basic Distributions Visualized

A few basic distributions are shown here for illustration purposes. Although normally shown as normalized functions within the context of probability theory, it is instructive to also view them as multidimensional functions of both data and parameters. A few simple distributions can be visualized in the univariate context by fixing their parameters $(\lambda_t, \lambda_c, \mu, \sigma)$ to fixed concrete floating point values. However, within the context of our analysis, we will also need to visualize each instead as multivariate function of both the random variable of interest (t or k) as well as the parameters involved $(\lambda_t, \lambda_c, \mu, \sigma)$.

Univariate Context: Negative Exponential Distribution

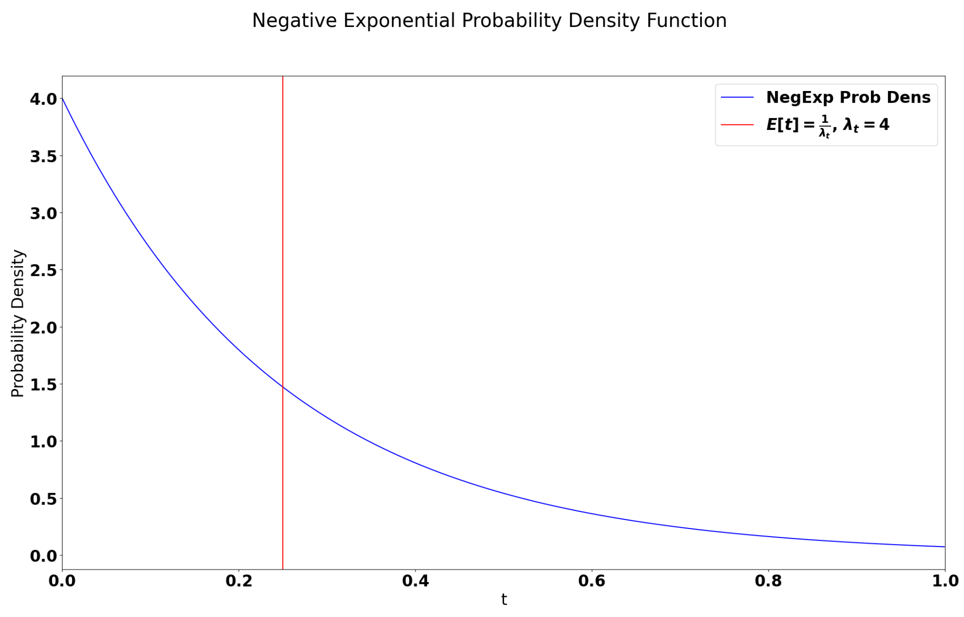


Figure C.1 Normalized PDF Negative Exponential

Multivariate Context: Exponential Distribution

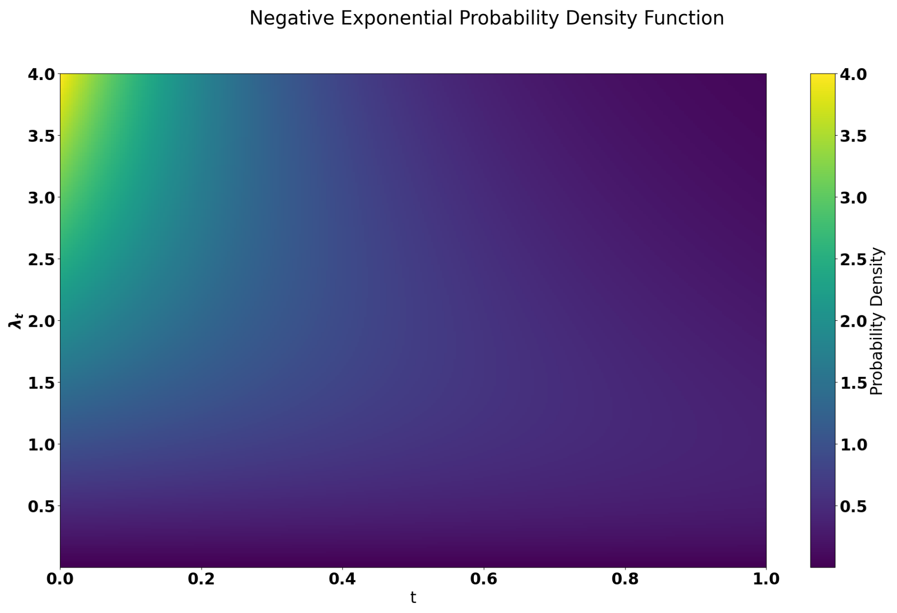


Figure C.2 Un-normalized Joint PDF Negative Exponential

Multivariate Context: Poisson Distribution

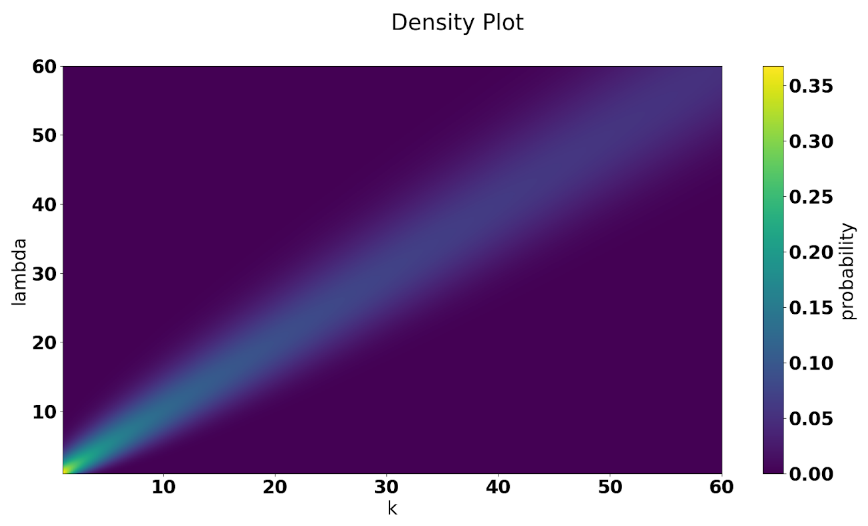


Figure C.3 Un-normalized Joint PDF Poisson

Appendix D

Useful Density Marginalizations

Marginalizing Poisson × Binomial The following deriviation / proof was provided by Jon Oullet

$$\begin{aligned}
P(\lambda, k_{true}, k_{obs}) &= \text{Poisson}(\lambda, k_{true})\text{Binom}(k_{obs}, k_{true}, p) \\
&= \frac{\lambda^{k_{true}} e^{-\lambda}}{k_{true}!} \frac{k_{true}!}{k_{obs}!(k_{true} - k_{obs})!} p^{k_{obs}} (1-p)^{k_{true} - k_{obs}}
\end{aligned}$$

Note the number of true observations (k_{true}) must be greater than the number observed k_{obs}. Marginalize over all possible values of k_{true}:

$$\begin{aligned}
P(\lambda, k_{obs}) &= \sum_{k_{true}=k_{obs}}^{\infty} \text{Poisson}(\lambda, k_{true})\text{Binom}(k_{obs}, k_{true}, p) \\
&= \sum_{k_{true}=k_{obs}}^{\infty} \frac{\lambda^{k_{true}} e^{-\lambda}}{k_{true}!} \frac{k_{true}!}{k_{obs}!(k_{true} - k_{obs})!} p^{k_{obs}} (1-p)^{k_{true} - k_{obs}} \\
&= \frac{p^{k_{obs}} e^{-\lambda}}{k_{obs}!} \sum_{k_{true}=k_{obs}}^{\infty} \frac{\lambda^{k_{true}}}{(k_{true} - k_{obs})!} (1-p)^{k_{true} - k_{obs}}
\end{aligned}$$

define $n = k_{true} - k_{obs}$.

$$\begin{aligned}
&= \frac{p^{k_{obs}} e^{-\lambda}}{k_{obs}!} \sum_{n=0}^{\infty} \frac{\lambda^{n+k_{obs}}}{n!} (1-p)^n \\
&= \frac{(p\lambda)^{k_{obs}} e^{-\lambda}}{k_{obs}!} \sum_{n=0}^{\infty} \frac{((1-p)\lambda)^n}{n!} \\
&= \frac{(p\lambda)^{k_{obs}} e^{-\lambda}}{k_{obs}!} e^{(1-p)\lambda} \\
&= \frac{(p\lambda)^{k_{obs}} e^{-p\lambda}}{k_{obs}!} \\
&= \text{Poisson}(p\lambda, k_{obs})
\end{aligned}$$

The final mathematical result is then:

$$\sum_{k_{true}=k_{obs}}^{\infty} \text{Poisson}(\lambda, k_{true}) \text{Binom}(k_{obs}, k_{true}, p) = \text{Poisson}(p\lambda, k_{obs}) \tag{D.1}$$

Appendix E

Useful Density Approximations

Definition of Exponential (e) The actual definition of e can be useful in other approximations:

$$e \equiv \lim_{n\to\infty}(1+\frac{1}{n})^n \tag{E.1}$$

Stirlings Formula Stirling's formula is an approximation to factorial in limit of large n:

$$\begin{aligned} \lim_{n\to\infty} ln(n!) &\approx nln(n) - n + \frac{1}{2}ln(2\pi n) \\ \lim_{n\to\infty} n! &\approx \sqrt{2\pi n}\; e^{-n}n^n \end{aligned} \tag{E.2}$$

The proof of Stirling's formula can be done using a Taylor series. While the basic execution of the Taylor series is pretty easy, showing that the radius of convergence spans the positive real number line is more difficult. The proof is omitted.

Binomial limit to Poisson

We need to start with the binomial distribution:

$$f \equiv Binomial_{pdf}(k,n,p) = \binom{n}{k} p^k(1-p)^{n-k} \tag{E.3}$$

Define the expected count of total successes with λ_c in terms of p and n.

$$\begin{aligned} \lambda_c &= np \\ p &= \frac{\lambda_c}{n} \end{aligned} \tag{E.4}$$

Plug our new representation of p into our distribution and expand out some terms:

$$\begin{aligned} f &= \binom{n}{k}\left(\frac{\lambda_c}{n}\right)^k\left(1-\frac{\lambda_c}{n}\right)^{n-k} \\ &= \frac{n!}{k!(n-k)!}\left(\frac{\lambda_c}{n}\right)^k\left(1-\frac{\lambda_c}{n}\right)^n\left(1-\frac{\lambda_c}{n}\right)^{-k} \\ &= \frac{\lambda_c^k}{k!}\frac{n!}{(n-k)!}\left(\frac{1}{n}\right)^k\left(1-\frac{\lambda_c}{n}\right)^n\left(1-\frac{\lambda_c}{n}\right)^{-k} \end{aligned} \tag{E.5}$$

Take the limit that n becomes large (and p is small so that λ_c is finite). Note that a few steps are easy but missing to show the convergence of each part of the expression to the amount shown in the under-brace. The first part approaches the value of 1 which can be shown by canceling out factorial terms. The middle part's limit can be easily derived using the definition of e. The last part trivially reduces to 1 because it has a term dividing by n which goes to zero.

$$\begin{aligned} \lim_{n\to\infty} f &= \frac{\lambda_c^k}{k!}\lim_{n\to\infty}\underbrace{\frac{n!}{(n-k)!}\left(\frac{1}{n}\right)^k}_{1}\underbrace{\left(1-\frac{\lambda_c}{n}\right)^n}_{e^{-\lambda_c}}\underbrace{\left(1-\frac{\lambda_c}{n}\right)^{-k}}_{1} \\ &= \frac{\lambda_c^k}{k!}e^{-\lambda_c} \\ &= Poisson(k, \lambda_c = \frac{n}{p}) \end{aligned} \tag{E.6}$$

Thus in the limit that the number of total trials n becomes very large, and the probability p of individual trial success is very low, a binomial distribution is well approximated by a poisson distribution.

Poisson limit to Gaussian In the limit that the expected number of observations becomes large ($\lambda \gtrapprox 100$) a Poisson distribution can be well approximated with a Gaussian distribution. A proof is not provided.

$$\lim_{\lambda\to\infty} Poisson_{pdf}(\lambda, k) \approx Gaussian_{pdf}(x, \mu = \lambda, \sigma^2 = \lambda) \tag{E.7}$$

Bibliography

[1] S. Dell'Oro, D. Q. Adams, C. Alduino, K. Alfonso, F. T. Avignone III, O. Azzolini, G. Bari, F. Bellini, G. Benato, M. Biassoni, et al., arXiv:1905.07667 [nucl-ex] (2019), arXiv: 1905.07667, URL http://arxiv.org/abs/1905.07667.

[2] D. Adams, C. Alduino, F. Alessandria, K. Alfonso, E. Andreotti, F. Avignone, O. Azzolini, M. Balata, I. Bandac, T. Banks, et al., Progress in Particle and Nuclear Physics **122**, 103902 (2022), ISSN 01466410, URL https://linkinghub.elsevier.com/retrieve/pii/S0146641021000612.

[3] N. Chott, Ph.D. thesis, University of South Carolina, Columbia, SC (2016), URL https://scholarcommons.sc.edu/etd/3984.

[4] The CUORE Collaboration, D. Q. Adams, C. Alduino, K. Alfonso, F. T. Avignone, O. Azzolini, G. Bari, F. Bellini, G. Benato, M. Beretta, et al., Nature **604**, 53 (2022), ISSN 0028-0836, 1476-4687, URL https://www.nature.com/articles/s41586-022-04497-4.

[5] B. S. Wang, Ph.D. thesis, University of California, Berkeley, Berkeley (2014), aDS Bibcode: 2014PhDT.......106W, URL https://ui.adsabs.harvard.edu/abs/2014PhDT.......106W.

[6] C. Alduino, F. Alessandria, K. Alfonso, E. Andreotti, C. Arnaboldi, F. Avignone, O. Azzolini, M. Balata, I. Bandac, T. Banks, et al., Physical Review Letters **120**, 132501 (2018), ISSN 0031-9007, 1079-7114, URL https://link.aps.org/doi/10.1103/PhysRevLett.120.132501.

[7] C. Alduino, K. Alfonso, D. R. Artusa, F. T. Avignone, O. Azzolini, T. I. Banks, G. Bari, J. W. Beeman, F. Bellini, G. Benato, et al., The European Physical Journal C **77**, 543 (2017), ISSN 1434-6044, 1434-6052, URL http://link.springer.com/10.1140/epjc/s10052-017-5080-6.

[8] D. Foreman-Mackey, The Journal of Open Source Software **1**, 24 (2016), ISSN 2475-9066, URL http://joss.theoj.org/papers/10.21105/joss.00024.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Journal of Machine Learning Research **12**, 2825 (2011), ISSN 1533-7928, URL http://jmlr.org/papers/v12/pedregosa11a.html.

[10] M. L. Waskom, Journal of Open Source Software **6**, 3021 (2021), ISSN 2475-9066, URL https://joss.theoj.org/papers/10.21105/joss.03021.

[11] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman, Publications of the Astronomical Society of the Pacific **125**, 306 (2013), ISSN 00046280, 15383873, URL http://iopscience.iop.org/article/10.1086/670067.

[12] K. S. Babu, E. Kearns, U. Al-Binni, S. Banerjee, D. V. Baxter, Z. Berezhiani, M. Bergevin, S. Bhattacharya, S. Brice, R. Brock, et al., arXiv:1311.5285 [hep-ex, physics:hep-ph] (2013), report of the Community Summer Study (Snowmass 2013), Intensity Frontier – Baryon Number Violation Group, URL http://arxiv.org/abs/1311.5285.

[13] H. Georgi and S. L. Glashow, Physical Review Letters **32**, 438 (1974), URL https://link.aps.org/doi/10.1103/PhysRevLett.32.438.

[14] The Super-Kamiokande Collaboration, M. Tanaka, K. Abe, C. Bronner, Y. Hayato, M. Ikeda, S. Imaizumi, H. Ito, J. Kameda, Y. Kataoka, et al., Physi-

cal Review D **101**, 052011 (2020), URL https://link.aps.org/doi/10.1103/PhysRevD.101.052011.

[15] K. S. Babu, I. Gogoladze, and K. Wang, Physics Letters B **570**, 32 (2003), ISSN 0370-2693, URL http://www.sciencedirect.com/science/article/pii/S0370269303010578.

[16] B. L. G. Bakker, A. I. Veselov, and M. A. Zubkov, Physics Letters B **620**, 156 (2005), ISSN 0370-2693, URL https://www.sciencedirect.com/science/article/pii/S0370269305008166.

[17] Majorana Collaboration, S. Alvis, I. Arnquist, F. Avignone, A. Barabash, C. Barton, V. Basu, F. Bertrand, B. Bos, V. Brudanin, et al., Physical Review D **99**, 072004 (2019), URL https://link.aps.org/doi/10.1103/PhysRevD.99.072004.

[18] J. Albert, G. Anton, I. Badhrees, P. Barbeau, R. Bayerlein, D. Beck, V. Belov, M. Breidenbach, T. Brunner, G. Cao, et al., Physical Review D **97**, 072007 (2018), ISSN 2470-0010, 2470-0029, URL https://link.aps.org/doi/10.1103/PhysRevD.97.072007.

[19] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003), ISSN 01689002, URL https://linkinghub.elsevier.com/retrieve/pii/S0168900203013688.

[20] Particle Data Group, M. Tanabashi, K. Hagiwara, K. Hikasa, K. Nakamura, Y. Sumino, F. Takahashi, J. Tanaka, K. Agashe, G. Aielli, et al., Physical Review D **98**, 030001 (2018), URL https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[21] M. Ambrosio, R. Antolini, C. Aramo, G. Auriemma, A. Baldini, G. C. Barbarino, B. C. Barish, G. Battistoni, R. Bellotti, C. Bemporad, et al., Astroparticle Physics **10**, 11 (1999), ISSN 0927-6505, URL http://www.sciencedirect.com/science/article/pii/S0927650598000371.

[22] M. Biassoni, Thesis, INFN, Milan Bicocca (2013), URL https://inspirehep.net/literature/1615506.

[23] W. Feller, *An introduction to probability theory and its applications. Vol. 1*, vol. 1 of *Wiley series in probability and mathematical statistics* (Wiley, S.l., 2009), 3rd ed., ISBN 9780471257080.

[24] G. Bellini, J. Benziger, D. Bick, S. Bonetti, M. Buizza Avanzini, B. Caccianiga, L. Cadonati, F. Calaprice, C. Carraro, A. Chavarria, et al., Journal of Instrumentation **6**, 5005 (2011), aDS Bibcode: 2011JInst...6.5005B, URL https://ui.adsabs.harvard.edu/abs/2011JInst...6.5005B.

[25] W.-M. Yao et al., Journal of Physics G: Nuclear and Particle Physics **33**, 245 (2006), ISSN 0954-3899, 1361-6471, revised August 2007 by T.K. Gaisser and T. Stanev, URL https://iopscience.iop.org/article/10.1088/0954-3899/33/1/001.

[26] R. Brun, F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta, V. Vassilev, S. Linev, D. Piparo, G. GANIS, et al., *root-project/root: v6.18/02* (2019), URL https://zenodo.org/record/3895860.

[27] B. Rene, *RootTalk: Re: What does ROOT stand for?* (1988), rootTalk Email Archive, URL https://root.cern.ch/root/roottalk/roottalk98/0718.html#:~:text=ROOT%20really%20means%20the%20%22roots,Object%2DOriented%20Technology%22%20Rene%20Brun.

[28] M. Rosenblatt, The Annals of Mathematical Statistics **27**, 832 (1956), ISSN 0003-4851, URL http://projecteuclid.org/euclid.aoms/1177728190.

[29] E. Parzen, The Annals of Mathematical Statistics **33**, 1065 (1962), ISSN 0003-4851, URL http://projecteuclid.org/euclid.aoms/1177704472.

[30] B. W. Silverman, *Density estimation for statistics and data analysis*, no. 26 in Monographs on statistics and applied probability (Chapman & Hall/CRC, Boca Raton, 1998), ISBN 9780412246203.

[31] D. W. Scott, Biometrika **66**, 605 (1979), ISSN 0006-3444, 1464-3510, URL https://academic.oup.com/biomet/article-lookup/doi/10.1093/biomet/66.3.605.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, et al., eprint arXiv:1201.0490 (2012), URL https://arxiv.org/abs/1201.0490.

[33] F. George and B. M. G. Kibria, Current Research in Biostatistics **2**, 44 (2012), ISSN 2524-2229, URL https://thescipub.com/abstract/amjbsp.2011.44.55.

[34] M. Ambrosio, R. Antolini, G. Auriemma, R. Baker, A. Baldini, G. C. Barbarino, B. C. Barish, G. Battistoni, R. Bellotti, C. Bemporad, et al., Physical Review D **52**, 3793 (1995), URL https://link.aps.org/doi/10.1103/PhysRevD.52.3793.

[35] M. Agostini, K. Altenmuler, S. Appel, V. Atroshchenko, Z. Bagdasarian, D. Basilico, G. Bellini, J. Benziger, D. Bick, I. Bolognino, et al., Journal of Cosmology and Astroparticle Physics **2019**, 046 (2019), ISSN 1475-7516, URL https://iopscience.iop.org/article/10.1088/1475-7516/2019/02/046.