Nuclear Event Classification via Graphically Driven Computation

Michael Vernon Nelson

A thesis submitted to the faculty of

Brigham Young University

In partial fulfillment of the requirements for the degree of

Bachelor of Science

Lawrence B. Rees, Advisor

Department of Physics and Astronomy

Brigham Young University

April 19, 2019

ABSTRACT

Graphical Methods for Nuclear Event Classification

Michael Vernon Nelson

Department of Physics and Astronomy

Bachelor of Science

Frequently, in nuclear physics research, nuclear events are studied using the electrical or optical signals they produce in various materials. Research relating to neutrons has an especially strong need for robust and efficient signal classification algorithms that can be adapted quickly to varying materials and experiment types. Previous methods are considered, and a new graphically-driven software-based pulse shape discrimination approach is proposed. Our implementation of the method is explained in detail. Use of the method at BYU has significantly reduced the time required to begin new neutron-related research projects and adapt to changes in experimental setup. Furthermore, it has dramatically increased user understanding of the pulse classification process, reduced the number of errors made, and helped identify mistakes made in processing previous datasets.

Keywords: pulse shape discrimination, PSD, neutron, nuclear event classification

Acknowledgments

I would like to thank Lawrence B. Rees and John E. Ellsworth for their significant investments of time and energy over the three years and several projects I have had the opportunity to work with them. I would also like to thank Tyler Jarvis in the Math Department for his exceptional efforts in teaching me computational methods, and for his significant investment into the Applied Computational Mathematics Emphasis (ACME) program over the past several years. Finally, I would like to thank the Department of Physics and Astronomy here at BYU for generously funding my research.

Table of Contents

List of Figures

List of Tables

Chapter 1 Motivation

Research in nuclear fission often involves observing the interactions made by fission emissions in various materials. These emissions generate electrical or optical signals in detectors, and it becomes necessary to use signal analysis to discriminate between the various types of radiation that is of interest. The Nuclear Research Group here at Brigham Young University is often interested in studying neutrons emitted from heavy fission events, and thus has frequent need for an algorithm which can discriminate between the signals created by gamma rays, neutron recoil events, and neutron capture events.

Since the signals related to each of these events vary significantly between different types of materials, much effort has been expended adjusting or rewriting previous pulse shape discrimination (PSD) algorithms for use in each new research project. In the last five years alone, four separate programs have been written by students in the nuclear research group which attempt to classify nuclear events before performing some other analysis. The focus of this research then is to combine ideas from several former projects, as well as current research in the fields of physics and computer science, to create a simple and efficient program which can accurately classify nuclear events in a variety of materials and electronic setups. Prior research in this area is not covered in depth, but certain necessary ideas will be addressed throughout the paper. For a thorough survey of recent and historical methods in neutron classification see Balmer[5] *et al*.

Statistical Significance and Efficiency Requirements

Studies in nuclear physics rely on large quantities of data to be statistically significant. While gathering data is often straightforward, in many cases it is necessary to filter out a small percentage of events that are of interest. In these cases, very large amounts of data must be filtered through so that the subset of interest is large enough to be statistically significant. The development of this program was strongly motivated by a research project on sister neutrons here at BYU which is interested in a very small subset of collected data, and thus has very high efficiency requirements. To illustrate the need of a high efficiency classification algorithm, I outline some pieces of that study here.

The study in question is interested specifically in sister neutrons, or two neutrons that were emitted from the same radioactive decay. Two high-efficiency neutron detectors are used, and it is required that one neutron is detected in each detector. At one-meter distance each detector occupies approximately 0.04% fractional solid angle, meaning that .04% of emissions from the source will pass through each detector. Assuming on average that four neutrons are emitted per fission event, 10% detector efficiency, and isotropic neutron emission, for every neutron detected in the first detector, there is a .012% chance that a second neutron will pass through the other detector and be detected. Thus, on average, of 10,000 recorded events, only 1.2 events guaranteed to contain one neutron will have a second. The study is further complicated by the fact that very generous bounds are specified regarding when to record events, meaning that only a portion of recorded events contain one neutron. Additionally, a high percentage of radioactive decay events from even the highest neutron-flux sources contain no neutrons at all, and background radiation is largely neutron-free. Taken all together, at minimum, tens of thousands of events must be recorded and filtered through to find a single event of interest, and thousands of those events must be collected for the study to have any statistical validity at all.

Prior Research: Useful Pulse Attributes

The study just discussed is an extreme example because it is interested in such a small subset of data collected, but it illustrates clearly the need for an extremely efficient classification program. In this example, each event was recorded at a resolution of one sample per nanosecond, with approximately 16,000 samples per detector per event. This high resolution can potentially increase the accuracy of pulse shape discrimination methods, but it also increases the computational cost of any classification algorithm that relies on the original waveform data.

Because of this, our algorithm relies primarily upon attributes calculated about pulses within the waveform, and not on the original waveform itself. These attributes include area, width, amplitude, etc. Several pulse attributes identified in previous research have high discrimination power, meaning they are often successful at discriminating between neutron capture and gamma pulses. We rely heavily upon a metric known as the early area ratio. The early area ratio has been mentioned in several prior research papers and is considered in depth by Balmer[5] *et al*. The early area ratio is the ratio of the early area to the total area, with 'early' designated by some fixed cutoff, measured from the start of the pulse. Specifically, the early area is the area contained in the interval between the beginning of the rising edge of the pulse and a pre-designated amount of time behind that point (such as 20 ns, the optimal cutoff varies by material and electrical setup). This is illustrated in **Figure 1.1**.

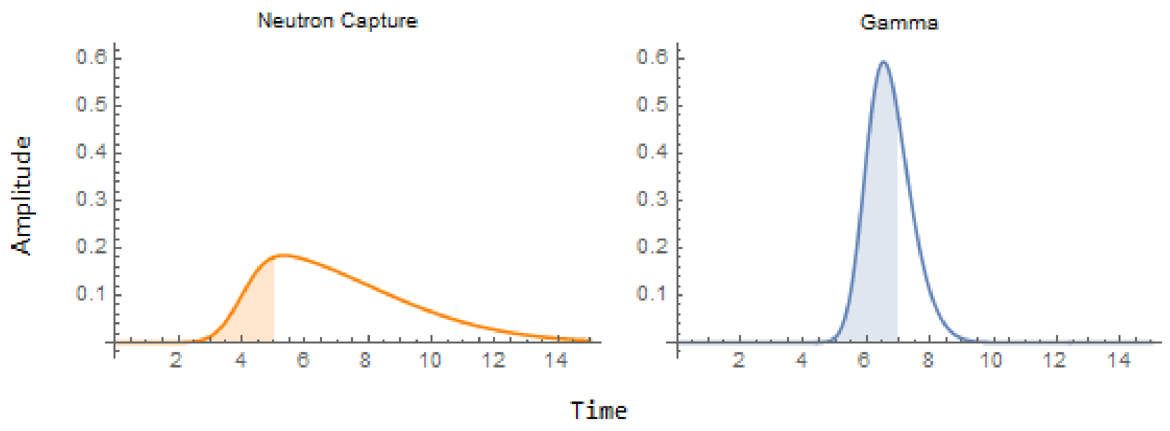

Figure 1.1 The Early Area Ratio is a metric commonly used to distinguish between neutron-capture and gamma energy emissions. It is the area within a pre-determined interval behind the beginning of the rising edge of pulse (shaded), divided by the total area.

Requirements of a Classification Algorithm

After motivating the need for an efficient pulse classification algorithm above, we attempt to quantify the efficiency and other requirements that an optimal solution would satisfy. In Ch. 3 we will identify how GuiSpec, the program we have written to fulfill these needs, meets the requirements.

Efficiency can be divided into two separate categories, namely efficiency during time when user input is required, and when it is not. Researcher input might be required, for example, to adjust classification requirements when experimental changes are made; however, the program should have the ability to operate autonomously to classify data after it is correctly trained.

When user input is required, efficiency can be limited both by slow runtime, and by difficulty of use. Furthermore, since human attention to the program is necessarily more costly to research than simply letting the computer run, ease of use is also a priority. Thus, while this is difficult to quantify, the program should maximize ease of use, minimize required user input time, and in the best case, add additional value to time researchers must use the program by teaching the user about the dataset they are studying.

Efficiency while user input is not required is slightly easier to quantify. Ideally, the algorithm could operate at the same speed data can be collected or faster, however this target varies by experiment and can potentially be very high. Data collection speeds are limited by a number of factors including source activity rate, detector size and proximity to source, detector efficiency, digitizer recording capacity, and the number of digitizers in use. However, a strict upper limit on the rate of data collection can be calculated for any digitizer with the assumption that it is actively recording 100% of the time. For an 8-channel digitizer with a 4 ns sampling rate, this would imply $2x10^9$ samples per second at maximum. In many practical applications, however, digitizer activity is likely to be less than 10%. Furthermore, recorded waveforms tend to be sparsely populated with interesting pulses, which reduces the net burden on pulse classification even further. An excellent goal then would be for any classification program to operate at speeds up to $2x10^8$ samples per second, but in practice the best test is to measure speeds under typical operating conditions for the given research group.

The third requirement is transparency. Any algorithm used in a rigorous academic study must be justifiable to the community. Furthermore, transparency will help the researcher to clearly understand the classification boundaries they are applying and promote a full understanding of the data in question.

Finally, any algorithm designed for pulse classification must be generic enough for use in the wide variety of studies conducted by the Nuclear Research Group at BYU. Furthermore, if the algorithm is designed robustly, this will allow for easy application by groups outside of BYU as well.

Conclusion

The rest of this paper outlines the theory, methodology, and implementation of our classification program. Chapter 2 summarizes several relevant techniques in machine learning and uses them to explain our own classification methodology. Chapter 3 outlines the specific tools and approaches we have developed, as well as the actual implementation of the program GuiSpec, with the key goal of informing future users how to use the program to its full potential. The degree to which GuiSpec satisfies the requirements outlined in this chapter will also be addressed in Ch. 3. Furthermore, appendix A contains documentation of key variables and functions.

Chapter 2 Contemporary Methods

Due to the recent successes in supervised machine learning and other artificial intelligence algorithms, it would seem machine learning is the obvious approach to any classification problem. Such algorithms can easily achieve higher than human classification accuracy on many datasets and can be extremely efficient. However, because the labeled datasets required to tune supervised machine learning algorithms can be difficult to generate, these algorithms are difficult to use for pulse classification in many nuclear physics experiments. In this chapter several machine learning algorithms and principles are discussed. While we do not apply them directly in our own research, they serve to illustrate several principles which we applied. The actual implementation of own algorithm is discussed in Ch. 3.

Inspiration Taken from Machine Learning

In this section we briefly summarize two popular classes of machine learning algorithms (random forest classifiers and artificial neural nets) and identify principles which are applied in our own method. The random forest algorithm and its subsequent iterations are extremely robust and easy to use. Artificial neural networks, which have generated an extreme level of excitement in and outside academia (the field of research is known as deep learning), can be computationally expensive to train but are extremely powerful and applicable to many types of problems. While neither of these two algorithms is fully suited to the needs of our research group, they provide significant insight toward the theory behind our own approach.

2.1.1 Random Forests

Random forest algorithms are simple in theory. A random forest is essentially a collection of dozens, or even hundreds, of decision trees. Objects are classified by weighing the decisions of all the trees in some manner to arrive at a single label or probability. A statistical cost function is used to generate each tree using a random subset of the variables and data available. Decision boundaries (leaves) that don't reduce the cost function more than some tolerance are subsequently pruned away.

2.1.1.1 Directed Construction

Significant improvements and tweaks have been made to the generic random forest algorithm, resulting in several new types of classification and regression algorithms. One such group of algorithms, collectively referred to as boosted tree algorithms (such as the popular Python package XGBoost), offers significant performance improvements. The primary difference in these algorithms is in the way the forest is constructed. The first tree is constructed as explained above; however, subsequent trees are generated in a directed manner. Each new tree after the first is still generated using a random subset of variables and data, but it is constructed in a way that focuses on correctly identifying the data misclassified by the first tree. In our classification algorithm, layers of classification boundaries are constructed using the same directed approach. Classification boundaries are first created using a set of two pulse attributes, then if any pulses are misclassified by the first set of boundaries, they can be corrected by new boundaries using another pair of attributes. This will be explained in much greater detail in Ch. 3.

2.1.1.2 *Inherits from Human Experience*

Random forests can use raw data, such as pixel values in images, and do so quite well, but they are especially well suited to using human-generated features. Since human-created features are used, one might suspect they would not perform significantly better than human performance, but random forests can construct much more complex and efficient classification structure than would be possible with human limitations. In this way a random forest can benefit from human experience.

Because pulse shape discrimination (PSD) algorithms have a long history in neutron research, there is a significant body of research to draw upon. Several useful features have been identified that have high discrimination power, such as the early area ratio outlined in Ch. 1 (see Balmer[5] *et al.* for examples of other features). These features would be ideal for use in a random forest classifier. Similarly, as described in Ch. 3, our classification algorithm relies heavily upon them.

2.1.2 Artificial Neural Nets

Research in deep learning has grown exponentially in the past ten years, and has even captured the attention of Hollywood, as in the movie AlphaGo (2017). Many high-profile systems, such as self-driving cars, rely heavily on deep learning, and deep neural nets have had profound implications in research fields like natural language processing, translation, and computer vision. However, many early successes in deep learning were in classification type problems, and feedforward neural nets can achieve near perfect classification accuracy using only raw data, such as image or audio files. Given enough data, it is nearly certain that modern architectures could achieve accuracy levels higher than trained human performance. While artificial neural nets have been successfully applied to gamma/neutron discrimination (Tambouratzis[1] *et al.*, Balmer[5] *et al.*), they require labeled data. Generating labeled data can require significant investments of time whenever changes are made that can affect pulse shape, such as changes in circuitry, detector material, etc., and in many cases is not feasible.

2.1.2.1 Non-Linear Boundary Formation

Just as the simple decision trees that populate a random forest can create complex classification structures by stacking simple layers of linear boundaries, artificial neural nets are able to recognize and classify abstract and complex objects by stacking layers of simple computations. The key difference between a simple learned decision tree and a neural net is the non-linearity incorporated between each layer of a neural net. By creating linear compositions of a pre-specified non-linear function (such as $|x|$), a sufficiently large artificial neural net can approximate any function. However, since it manipulates that non-linearity with only linear functions, which represent a much smaller function space than non-linear functions, it is still feasible to find the optimal composition. In this way a single non-linear decision tree can perform as well as or better than hundreds of smaller linear decision trees.

Interpretability and Dimension Reduction

One key issue with both artificial neural nets and random forests methods is their inherent lack of interpretability. While a single tree in a random forest is relatively straightforward to interpret, it is not feasible to identify defining trends by observing hundreds of trees. Similarly, although an artificial neural net is a single structure, it likely contains hundreds of thousands or millions of individual parameters, making it nearly impossible to determine why even a single input was classified as it was. Packages exist in multiple computer languages which can help interpret each algorithm, but it would be impossible to interpret them without computer assistance. Similarly, it would be impossible to construct either without computer assistance.

The intractability of each algorithm lies in its high dimensionality. Many machine learning algorithms rely on high dimensionality to separate classes of input. Often, the more dimensions a cluster lies in, the more likely it is to be linearly separable from any other clusters of data present. However, by relying on non-linear classification boundaries, we can dramatically reduce the number of dimensions required to separate input classes. For further context see Murphey[2], support vector machines and the kernel method. The construction of non-linear classification boundaries in covered in detail in Ch. 3.

Conclusion

Unfortunately, the cost functions used to train any supervised machine learning structure depend heavily upon having labeled data. While it is possible to construct labeled datasets, it is impossible to do so without being able to collect pure datasets of each type of event, or already being able to discriminate between events in a mixed dataset. If effort were expended to generate a labeled dataset so that a machine learning classification algorithm could be trained, the trained algorithm would most likely become inaccurate if any major change were made to the experimental setup. Thus, in an environment such as our research group, where new detectors are often being developed, and new student research projects begin frequently, expending effort to generate labeled data is impractical.

Algorithms that can identify distinctions between groups of unlabeled data exist and are referred to collectively as clustering algorithms, or unsupervised machine learning. Clustering algorithms, as the name implies, find natural boundaries to separate unlike data. Usually the data are separated into a specified number of groups. However, evaluating whether the classification boundaries constructed are sufficient for research specifications is difficult, even if initially they appear to fall in the correct locations. As will be outlined in Ch. 3, the tools necessary to evaluate the accuracy of automatically generated decision boundaries can also be used to construct decision boundaries on their own without significantly more effort.

Both machine learning algorithms considered relied on high-dimensional computations to classify even relatively simple objects. Random forests rely on horizonal integration of hundreds or even thousands of shallow linear decision trees, while deep neural nets vertically integrate tens of thousands of simple non-linear computations to form their decisions. Both are considered black box algorithms, meaning the inner workings of each are difficult if not impossible to understand or justify. Because of this, it may be difficult to justify the use of either algorithm in rigorous research in a field that currently has a limited exposure to both.

Our approach uses strengths from each algorithm to create a justifiable, but powerful classification structure. In Ch. 3 we outline the methodology and tools used to generate and justify this classification structure. Thorough explanations of specifics to the program GuiSpec, which is our implementation of the algorithm, is given in appendix A.

Chapter 3 GuiSpec Implementation

In this chapter we document the implementation of our classification algorithm, as well as the other functionality built into our program GuiSpec. The purpose of this is two-fold. First, I hope to document the full capabilities of the program so that it can be used effectively. The second is to convey a clear understanding of the underlying code so that later users can more efficiently modify the program to fit their needs.

GuiSpec is divided into the following sections, which will be discussed in order:

Table 3.A: GuiSpec organization

3.1 Pre-Processing	This includes extracting data from digitizer output files, pulse identification, and pulse attribute calculations.
3.2 Data Investigation	This encompasses the use of two toolboxes, namely the “Region Toolbox” and the “Pulse Identification Toolbox”. Each will be explained in detail.
3.3 Automated Classification	After classification boundaries have been defined, and the pulse identification algorithm tuned, GuiSpec can use these to automatically separate incoming waveforms into pulses and classify each type with the specified label.

Pre-Processing

GuiSpec contains several segments of code taken from previous students' work. All of this is found in the various stages of pre-processing. When this is the case it will be explicitly mentioned.

3.1.1 Digitizer Interfacing

Guispec is formatted to receive the standard output format from the CAEN line of lab digitizers, however the code that converts this format to that used throughout GuiSpec is isolated, so it can easily be replaced. Several parameters, outlined in appendix A, must match the parameters used by the digitizer during data collection. This section of code has been used and adapted by several prior students, and it is unknown who originally wrote it.

3.1.2 Pulse Identification

The pulse identification algorithm was written by former student Gregory Hill at my request. The objective of this algorithm is to identify distinct pulses in each waveform the digitizer records. It relies on the peak finding function, findpeaks(), which comes standard in MATLAB. Peaks found using findpeaks() are used to define where each distinct pulse begins and ends by comparing the height and distance between adjacent peaks using several thresholds (outlined in appendix A). For example, if two adjacent peaks are close enough in amplitude and the distance between them is small enough, they would both be considered part of the same pulse. These thresholds can be tuned in the pulse identification toolbox. The pulse identification toolbox is explained in the Data Investigation section, which follows shortly after this section.

Once this algorithm is further developed and tuned so that it can be trusted, an option to zero-length encode each waveform should be built into GuiSpec by future students. Zero-length encoding refers to deleting portions of the waveform that lie between pulses to reduce the storage space required for output data.

3.1.3 Resolution Improvements

While many researchers do not require higher resolution than what their digitizer provides, the resolution can effectively be increased by using multiple channels on the digitizer for the same input. This is done by dividing the incoming signal and feeding it into multiple adjacent channels, then zipping the output from each of those channels together into a single waveform. Care must be taken in choosing wires of the correct length so that an equal delay exists between each input channel. I explain this technique for the benefit of future students who wish to use it in concert with GuiSpec.

As an example, consider an 8-channel lab digitizer with a 4 ns resolution, meaning each channel samples once every 4 ns. It is possible, and in fact not difficult, to sample a single input every half nanosecond using an 8-way splitter and 8 separate wires with a ½ ns difference in length between each, setting the shortest wire on the first channel. Note that sampling at this high of a rate is not beneficial when integrating circuitry, such as an amplifier, is used. Because of this it is also inadvisable to use this technique with slow or dim PMT’s.

GuiSpec is designed to handle such resolution increases without any additional effort. The variable **splitChannels** is used to specify which, if any, channels are sampling from the same input. GuiSpec is also capable of streaming together inputs taken from multiple digitizers used in sync, although the total number of channels is somewhat limited by the graphic interface.

3.1.4 Pulse Parameter Calculation

GuiSpec prepares for pulse classification by calculating several parameters about each pulse. These include amplitude, width, area, early area ratio, peak start time, and peak end time, among others. Significant efficiency and adaptability are added to GuiSpec by allowing the user to decide which of these are to be calculated and to add new variables. The pulse attributes that are not needed can be switched off, so they are no longer calculated. Additionally, if a new variable is added in the list of attributes to be calculated that GuiSpec does not recognize, it prompts the user with the exact location where code to calculate the new attribute can be added.

Pulse attribute calculation is the most computationally expensive portion of the classification algorithm since it operates on each waveform at full resolution. Going forward GuiSpec relies only on the attributes calculated here, meaning that its efficiency depends only upon the number of pulses identified, and not on the resolution of each pulse. Because of this, all subsequent calculations can easily be performed in real time as user interaction with the graphic interface requires.

Data Investigation

Once the needed attributes of each pulse are calculated, they can be used to investigate how pulses of different types are distinct, and how they can be separated. Ideally, if enough meaningful attributes are collected, each type of pulse will be separated into distinct clusters in an n-dimensional vector space, where each of the n attributes calculated is ascribed one axis. Since we are unable to visualize high-dimensional spaces, we investigate each cluster using two dimensional projections of the space (see **Figure 3.1**). If some attributes are not necessary to obtain sufficient separation, they need not be calculated, and can be disabled in subsequent tests.

What remains then is to summarize the tools developed to aid in identifying clusters, and how to define the boundaries around them for subsequent classification.

3.1.5 Region Toolbox: Cluster Investigation and Boundary Definition

The primary method for visualizing the vector space we have defined using attributes such as width, amplitude, area, etc. is the two-dimensional scatter plot. Scatter plotting all identified pulses across two of these attributes is exactly equivalent to viewing a projection of the whole space onto those two attributes, as mentioned previously. While it may be feasible in some low-dimensional cases to gain an understanding of the entire space, this is not necessary. Computer dimension reduction techniques are also not necessary. Since the total number of dimensions is small enough, it is not difficult to manually identify which dimensions contain the most useful information using two dimensional projections.

Instead of attempting to gain an understanding of the entire space, it is more tractable to choose two important attributes as the two primary axes and view any other sets of axes as non-linear transformations of that original space. In this way a holistic view of an individual cluster can be gained, without the pains of visualizing a high-dimensional vector space. For example: in **Figure 3.1** below, a distinct cluster (blue) can easily be seen plotted against the primary axes (top). However, after a transformation (bottom), it is easily noticeable that several more pulses belong to the cluster that were not identified in the frame because the top and left boundaries were not sharply defined. Some pulses were also included that are obviously not part of the cluster after the transformation. These corrections can easily be made using two regions drawn in the new frame. After two or three such transformations, we can clearly identify and define boundaries around a high-dimensional cluster with only a few plots.

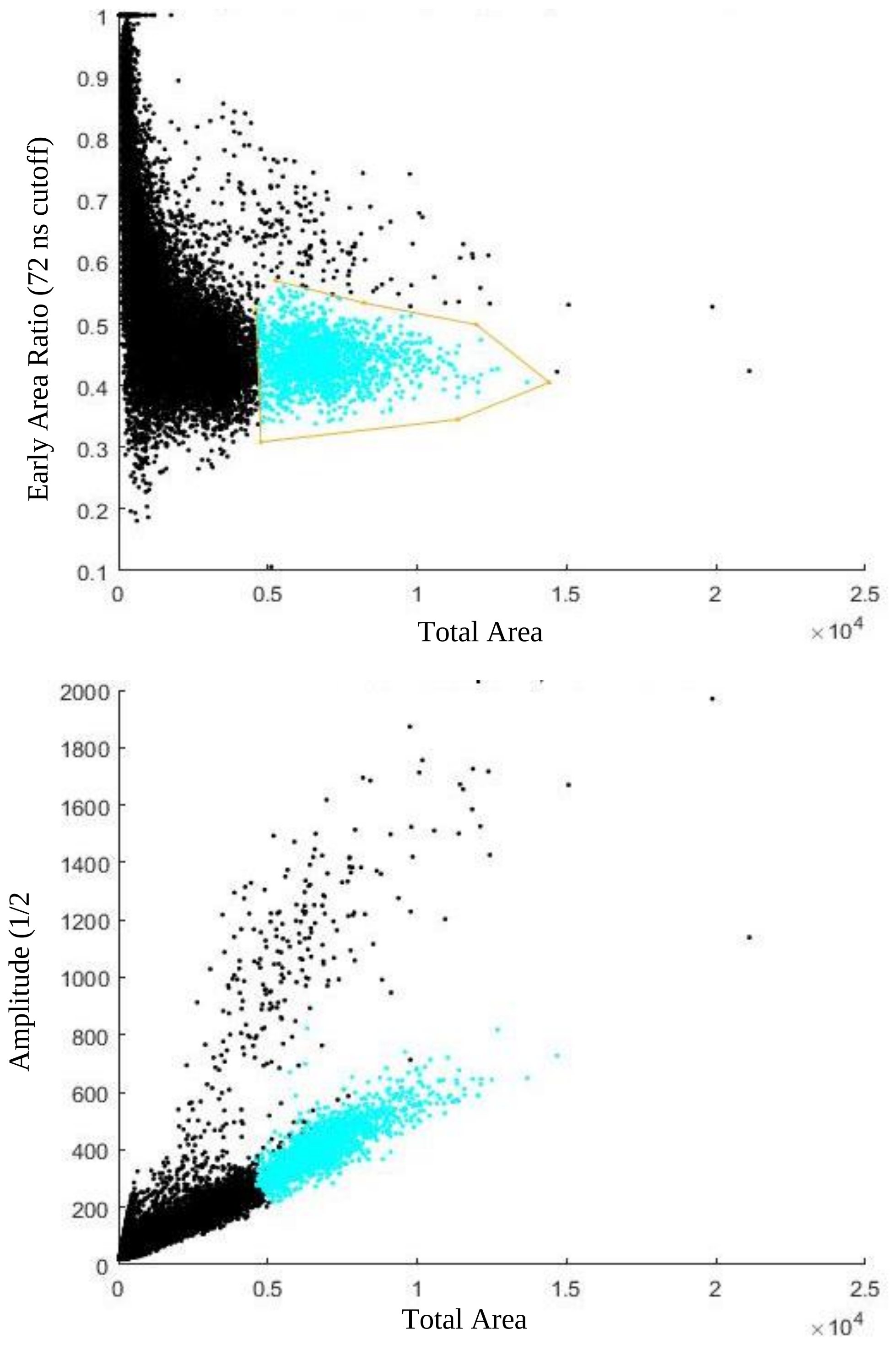

Figure 3.1: Two-Dimensional Projections of the Pulse Attribute Vector Space

This is the core of our classification algorithm. Regions are defined by drawing polygons on a scatterplot. Several tools have been created allowing the investigation of each region created. Each region is assigned a classification and an order. Then pulses are classified by simply assigning them to the type of each region they are contained in, with each region assignment occurring in the order designated. This collection of tools and the associated interface is referred to as the Region Toolbox, which is part of GuiSpec. Subsequent datasets can be used to validate or tweak the regions created, or they can be efficiently classified using the previously defined boundaries.

3.1.5.1 Using the Region Toolbox

As explained above, the region toolbox is used to classify pulses by defining regions around clusters of pulses in two-dimensional slices of the vector space created using pulse attributes such as amplitude, width, or area. Regions are created by drawing polygons around the desired clusters on a two-dimensional scatterplot. The region is assigned a label, such as ‘gamma’, ‘neutron capture’, ‘anomaly’, etc., and pulses inside the region are given that label. By stacking these regions together, a complex non-linear decision tree is created that can achieve much higher accuracy than any individual region. Regions are simple to create, but more difficult to evaluate. Three tools have been created which can help identify where clusters naturally end, and what type of pulse a region contains.

When a region has been created and selected, a button labeled ‘Investigate Region’ becomes active. This button creates two histograms, one for each axis of the region, and randomly selects a pre-defined number of pulses to display one at a time. The two histograms are most useful for detecting where contained clusters naturally end or alerting the user if multiple overlapping clusters are in the region. Given that a natural gaussian-like distribution is expected for each cluster, sharp cutoffs on either histogram would indicate that a portion of the cluster is not contained in the region. This is clearly apparent in the two histograms (**Figure 3.2** below) created for the blue region in **Figure 3.1**.

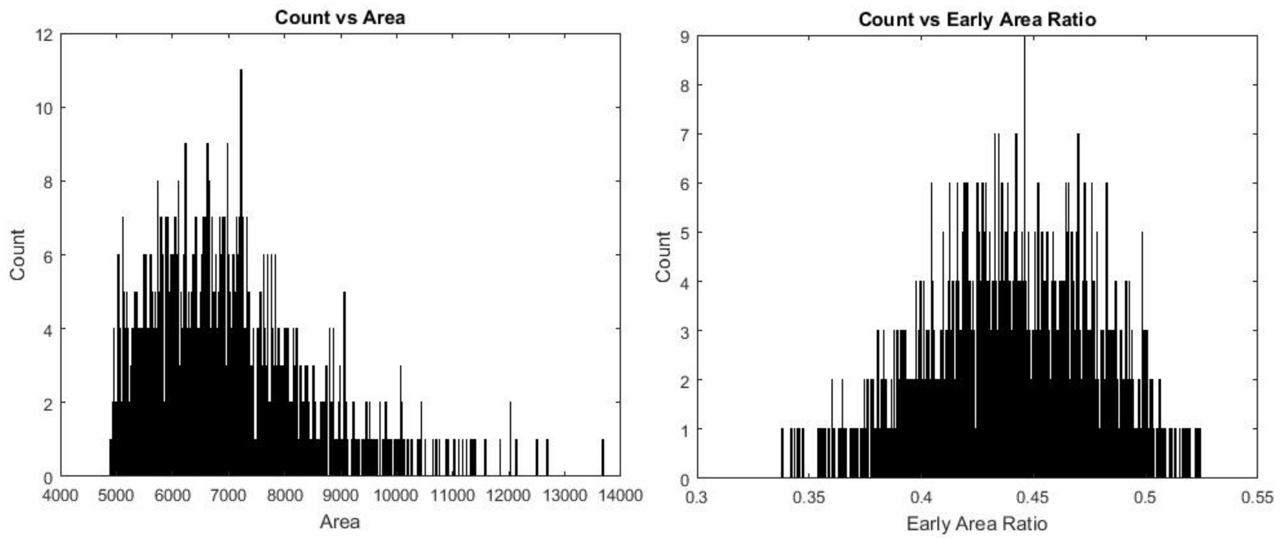

Figure 3.2: Histogram for each axis of the blue region in **Figure 3.1**. Note how the left boundary in the left histogram is sharp. This may indicate that the region contains only part of a natural cluster.

In addition to the histograms generated, the ‘Investigate Region’ button also allows the user to click through a pre-specified number (meta variable “samples”) of randomly sampled pulses from the region. This allows for visual confirmation that the region does or does not contain only one type of pulse, and to determine what kind of pulse is contained.

3.1.6 Pulse Toolbox

The second main toolbox in GuiSpec, called the ‘Pulse Toolbox’, provides additional methods for pulse investigation. This toolbox is designed to allow the user to tune the pulse identification algorithm. The toolbox samples pulses either from the entire group, or from a defined classification (such as ‘neutrons’ or ‘anomalies’), then displays them for the user. The user can then tweak the parameters used in the pulse identification algorithm and view directly how the boundaries of the sampled pulses would be defined differently, and if they now would be classified differently. Used together with the region toolbox this becomes a very powerful tool. Pulses that are misidentified tend to stray from defined clusters and are easily labeled by the user in the region toolbox. Then these can be used to tune the pulse identification parameters to eliminate the anomalous clusters. This is also an effective method for identifying electrical noise and tuning the pulse identification algorithm to ignore it.

3.1.7 Multiple Detector Compatibility

GuiSpec is designed to be used with any number of detectors, including cases where all detectors are recorded by the same digitizer, or where multiple digitizers are used in sequence. The region and pulse toolboxes therefore are also designed to be compatible with multiple detectors. The region toolbox includes a checkbox used to enable or disable each detector. Pulses are displayed in the toolbox only from detectors that are enabled, and when a region is created, it remembers which detectors were enabled when it was created. Regions are only used to classify pulses from detectors that were enabled when it was created. In this way a distinct classification structure can be generated for each detector individually where needed, but each detector can share any individual region.

The pulse toolbox also changes pulse identification parameters individually for each detector, so the pulse identification algorithm can easily be different for separate detectors if needed. When multiple detectors are enabled in the region toolbox, the toolbox displays the full waveform for each enabled detector when a pulse is sampled and updates the parameters for each detector enabled together. However, if the classification structure is distinct for two detectors viewed together in the toolbox, pulse labels are displayed correctly according to the separate structures even when changes are made to pulse boundaries that moves a pulse across regions. Any number of detectors may be used that are operating on the same trigger. The only limitation upon the number of detectors is the graphic interface, which may become over encumbered.

Automated Classification

When the user is satisfied with the pulse identification algorithm and classification structure they have generated, all of these parameters can be saved in a small file. Then subsequent datasets taken using the same detectors can be processed automatically by GuiSpec using this file. Alternatively, subsequent datasets can be viewed in the pulse and region toolboxes using these settings to check if the settings still work appropriately. This is an excellent way to identify new sources of electrical noise or drift that have occurred since the experiment began. Small adjustments to the classification structure or pulse identification parameters can easily be made to this file using the two toolboxes.

Conclusion

It is difficult to adequately explain all of the functionality of GuiSpec without demonstrating in real time, but I hope that this paper has given sufficient introduction that students following after me can quickly learn the diverse capabilities that it offers. GuiSpec satisfies each requirement defined in Ch. 1, namely efficiency, transparency, and adaptability.

Since the classification algorithm operates exclusively on pulse attributes, and not on the waveforms themselves, the classification algorithm is extremely fast. The classification algorithm itself is much faster than pulse identification and calculation of the needed attributes. Those processes have also been optimized as much as possible.

Because the classification structures are generated using graphics, it is not difficult to display exactly how pulses are classified. This means that in the act of generating the classification structure, a researcher also generates the means by which they can justify the process. The act of creating the structure also forces the user to understand how and why they can separate pulses as they have, making the process a white-box algorithm, and completely transparent.

GuiSpec is also extremely adaptable to changes in experimental setup, electrical equipment, or detector material. Once the user is familiar with the program and the types of events they are classifying, generating classification structures is not a time-consuming process. After the structure is generated it can automatically classify any number of pulses so long as the clusters remain stable. Checking if the regions defined still classify correctly is also streamlined, and if clusters have drifted, the containing regions can easily be adjusted.

The graphic nature of GuiSpec leads to a much more comprehensive understanding of the classification process and will help both the professional researcher and new student avoid mistakes that could taint later conclusions. GuiSpec has already dramatically streamlined each new experiment started in the nuclear research group at BYU since it was designed and helped identify mistakes made in previous experiments as well. GuiSpec also represents a dramatic improvement in classification accuracy and efficiency over the processes designed by prior students and faculty here at the university. We hope that it will also benefit other universities and groups to which it is distributed.

3.1.8 Future Work

The motivation in creating GuiSpec was to improve undergraduate experience and improve the quality of research within the Nuclear Research Group at BYU, as well as to enable the group to begin a new data-intensive research project, namely that described briefly in Ch. 1.

The purpose of this research project is to determine whether all neutrons emitted from heavy fission events are emitted via the boil-off model, or whether some portion of them may be emitted via some other method. Other models for neutron emission, such as the scission model, were prevalent before the widespread adoption of the boil-off model. However, recent research indicates that some fraction of neutrons emitted in heavy fission have properties not expected of boil-off neutrons[3,4]. The purpose of the project would be to measure the prevalence of these properties and evaluate whether some other model may explain the discrepancies in this small portion of neutron emissions.

If some portion of neutrons are emitted via a different process than boil-off, this may have large implications in nuclear physics. While we presume boil-off neutrons contain no information about the state they occupied within the nucleus prior to emission, other potential methods of emission (such as scission), may be influenced by the neutron's state in the nucleus. If this is the case, nuclear physicists will have a new channel through which they can study the internal mechanisms of nuclei prior to fission.

We suspect that some other mechanism may be at work for some small portion of neutrons emitted in heavy fission events because prior BYU student KaeCee Terry[3] found that pairs of neutrons from the same nuclear fission could be detected in a single detector at a much higher rate than would expected according to the boil-off model. See her thesis[3] for details. This suggests that there may be some other mechanism by which neutrons are emitted in a non-isotropic fashion. Recent computational studies find that neutron emission via scission is a likely candidate[5].

Future studies could meticulously measure the discrepancy in the number of close-angle neutrons using two high-efficiency neutron detectors. These should be placed far enough away from the source as to obtain a high resolution in the angle between detected pairs of neutrons. Further confirmation that some close-angle neutrons are correlated should be sought by comparing the mean difference in kinetic energy between neutron pairs detected at small angles to that projected by the boil-off model. Large amounts of data will need to be required to obtain statistical significance at each angle measured since two neutrons must be detected, and data should be recorded if ever one neutron is seen, requiring a high filtering capacity (which GuiSpec has). Furthermore, very high efficiency neutron detectors are also desirable.

Appendix A: Documentation of Code

Summary of input variables

The following tables list all input variables (found at the top of the main file). They are divided into three categories. Table 1 summarizes all variables relating to the format of input data and settings used with the digitizer. Table 2 contains variables relating to how the graphic interface and pulse classification algorithm operate. Finally, table 3 contains all variables relating to the pulse identification algorithm.

Table 0.A: GuiSpec Input Data Settings (i.e. mostly digitizer output settings found in the digitizer configuration file).

Variable Name	Description	Possible Values
splitChannels	This variable designates which channels are aligned with which detector. It should be an ordered array listing the first channel occupied by each detector.	Ordered arrays of integers > 1. E.g. [1, 3, 5, 7].
headersEnabled	Set true if header is enabled in the digitizer configuration.	0, 1
channelsEnabled	The total number of channels used on the digitizer. This is used in concert with **splitChannels** to determine how the raw data aligns across detectors.	> 0
samplesPerEventPerChannel	The number of samples recorded by the digitizer each time a trigger occurs.	Integer powers of two (e.g. 4096)
maxEventsPerFile	This variable should be the same size or larger than the variable of the same name in the configuration file.	Integer > 0
digitizerOutputFileExtension	This should match the file extension of the raw data files. This is set in the digitizer config file.	String, period followed by three letters e.g. '.dat'
milivoltsPerChannel	The number of millivolts between vertical samples.	Float > 0. Typically .5.

nsPerChannel	The number of nanoseconds between samples. In other words, the resolution of the digitizer.	Integer > 0. Typically 4.

Table 0.B GuiSpec Region Toolbox and pre-processing variables

earlyAreaCutoff	The default cutoff used to calculate the early area ratio pulse attribute.	Float > 0 Measured in nanoseconds
earlyAreaMax	The maximum allowed value for the earlyAreaCutoff as set by the slider in the Region Toolbox.	Float > earlyAreaCutoff Measured in nanoseconds
fixedAreaCutoff	The cutoff used in calculating the pulse attribute fixedArea. The fixed area is the area for a fixed amount of time after the beginning of the pulse. This ignores any pulse designations, meaning it will keep integrating for that amount of time unless it meets the end of the waveform.	Float > 0 Measured in nanoseconds
bins	The number of bins used in region histograms.	Integer > 0
samples	The number of pulses to sample from a region when the "Investigate Region" button is pushed.	Integer > 0
regions	A list of the labels that can be used as pulse classifications.	e.g. `["start", "stop", "capture"]`
regionColors	A list of colors to be used for each class label. Same length as **regions** variable.	Valid scatterplot marker codes e.g. `{'g.', 'k.', 'b.'}`

pulseProperties	The list of pulse properties which should be calculated in pre-processing.	`e.g.` `{'timingLoc', 'peakHeight', 'width', 'area', 'earlyAreaRatio', 'fixedTimeArea'}`
plotLabels	The axis labels which should be used corresponding to the **pulseProperties**.	`Same length as` **`pulseProperties`** `e.g.` `{'timing location (ns)', 'amplitude (mV)', 'width (ns)','area (mv*ns)', 'early area ratio', 'fixed time area'}`

Table 0.C Pulse Toolbox Variables

These are used in the pulse identification algorithm in pre-processing stage but can be fine-tuned in the Pulse Toolbox. All variables are lists of floats e.g. [46, 52, 5]. Pulses will be identified using thresholds according to the greatest index less than that of the detector being use, i.e. detector three will use the thresholds in each list at index three unless the length of the list is shorter than three, in which case it will use the last value.

thresh	Any peak higher than this will be considered a valid pulse. The beginning of each pulse is defined **riseOffset** indices prior to where this threshold is crossed.
dropThresh	When a valid peak drops below this threshold the pulse end is determined. Specifically, the end of the pulse will be designated at **fallOffset** indices after the location this occurs.
riseOffset	The number of values prior to the location a peak first rises above **thresh** to designate as part of that pulse.
fallOffset	The number of values behind the location a peak first falls below **dropThresh** to designate as part of that pulse.
separation	The maximum separation between adjacent peaks before they are considered part of separate pulses.
widthThresh	The minimum distance between the locations the rising and falling thresholds are crossed before a pulse is considered valid.

Index

Bibliography

1 Tatiana Tambouratzis, Dina Chernikova, Imre Pzsit, "Pulse shape discrimination of neutrons and gamma rays using Kohonen artificial neural networks," Journal of Artificial Intelligence and Soft Computing Research 3 (2), 77-88 .

2 Kevin Murphey, Machine learning : A probabilistic perspective, edited by Anonymous 2012), .

3 KaeCee Terry, "Paired Neutron Detection," , (2016).

4 N. Carjan, P. Talou and O. Serot, "Emission of scission neutrons in the sudden approximation," Nuclear Physics, Section A 792 (1), 102-121 (2007).

5 Matthew J. I. Balmer, Kelum A. A. Gamage and Graeme C. Taylor, "Comparative analysis of pulse shape discrimination methods in a ^{6}Li loaded plastic scintillator," Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 788, 146-153 (2015).