EVALUATION OF AMERICIUM-241 ALPHA PARTICLES INCIDENT ON GS20 GLASS AS AN ISOTROPIC NEUTRON SOURCE

by

Jarom Peterson

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

April 2026

BRIGHAM YOUNG UNIVERSITY - IDAHO

DEPARTMENT APPROVAL

of a senior thesis submitted by

Jarom Peterson

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

Date	Kevin Kelley, Advisor
Date	Lance Nelson, Committee Member
Date	David Oliphant, Committee Member
Date	R. Todd Lines, Thesis Coordinator
Date	Evan Hansen, Department Chair

ABSTRACT

EVALUATION OF AMERICIUM-241 ALPHA PARTICLES INCIDENT ON GS20 GLASS AS AN ISOTROPIC NEUTRON SOURCE

Jarom Peterson

Department of Physics

Bachelor of Science

A Type 20 Glass Scintillator (GS20) doped with Americium-241 was evaluated inside the BYU Isotropic Fission Chamber (IFC) to see if it is a viable neutron source. The sample, originally manufactured in 1985, was assessed for radiation damage and found to be in good condition. Neutrons, which resulted from ^{241}Am alpha emissions interacting with the chemical elements of GS20, were optically flagged and observed using the IFC in conjunction with the LGB-PVT Neutron Pseudo Spectrometer Detector. Data of these interactions between ^{241}Am and GS20 were recorded via a CAEN desktop digitizer. The time difference for a neutron interacting with then being absorbed into the neutron detector indicated there were twenty-two neutrons found across over one-hundred thousand detections.

ACKNOWLEDGMENTS

I want to thank my mentor, John Ellsworth, for his support and advice while doing this research. I also thank Isaac Wilden for teaching me how to use Octave so that analysis of the data was possible. I thank the many BYU-Idaho faculty who helped me develop the skills, mindset, and patience to become a scientist; without their support, I would not have been capable of research. I also thank my fellow students who encouraged me when I didn't think I could continue as a physicist. Finally, I acknowledge that this research is supported by NSF grant #2348770.

Contents

Table of Contents vii

List of Figures viii

1 Introduction 1

2 Methods 3

3 Analysis and Results 11

4 Conclusions 15

Bibliography 16

A Event Sorting Code 18

List of Figures

1.1 ^{241}Am doped GS20 sample 2

2.1 IFC components 4
2.2 Hardware for GS20 spectrum 4
2.3 1985 spectrum vs. 2025 spectrum 5
2.4 Experimental components used in the detector system. 5
2.5 Interaction detected by IFC 6
2.6 Neutron collision visualization 8
2.7 Thermal and capture gate 8
2.8 Process of ^{241}Am alphas to neutrons 9
2.9 Hardware arrangement for detecting neutrons. 10

3.1 Flowchart of the code logic. 12
3.2 All flagged events. 12

Chapter 1

Introduction

Neutron sources can be produced through a variety of methods, many of which require specialized equipment, such as particle accelerators and spark chambers[1]. For example, The Los Alamos Neutron Science Center uses an 800 MeV proton beam to produce neutrons for its research facilities[2]. Building and maintaining such facilities can require millions of dollars, making them inaccessible to institutions with low funding. As a result, few institutions in the United States provide training in neutron science. To make neutron science in academic settings more accessible, a more affordable option is necessary. Americium-241 is one of the more cost-effective radioactive isotopes and can produce neutrons when alpha particles from its decay interact with suitable materials. In 1985, glass scintillator type 20 (GS20) was doped with ^{241}Am (figure 1.1), as a potential low-cost neutron source. A scintillator is a material that will flouresce, or give off light, when exposed to ionizing radiation[3]. Type 2 - 0 (with 20 being the shorthand) refers to the isotopic enrichment of Lithium-6 and total Li by percent weight respectively. Type 20 then is meant to inform that the total weight of Li in the sample is 6.6% and 95% of the Li is the ^{6}Li isotope[4]. When speaking of GS20 then, I refer to a scintillator made of glass with 6.6% total weight

being Li and 95% of the Li being the ^{6}Li isotope. GS20 contains isotopes that produce neutrons when exposed to the 5.49 MeV alpha particles emitted from the ^{241}Am [5]. The objective of this research is to assess the feasibility of using a ^{241}Am doped GS20 sample as an isotopic neutron source. If successful, this approach would reduce costs and provide training opportunities in modestly funded institutions for students in a nuclear physics area that is often overlooked, namely neutron physics[6]. An Isotropic Fission Chamber (IFC) and neutron detector together can test if neutrons were emitted from interactions between the ^{241}Am alphas and the GS20 sample.

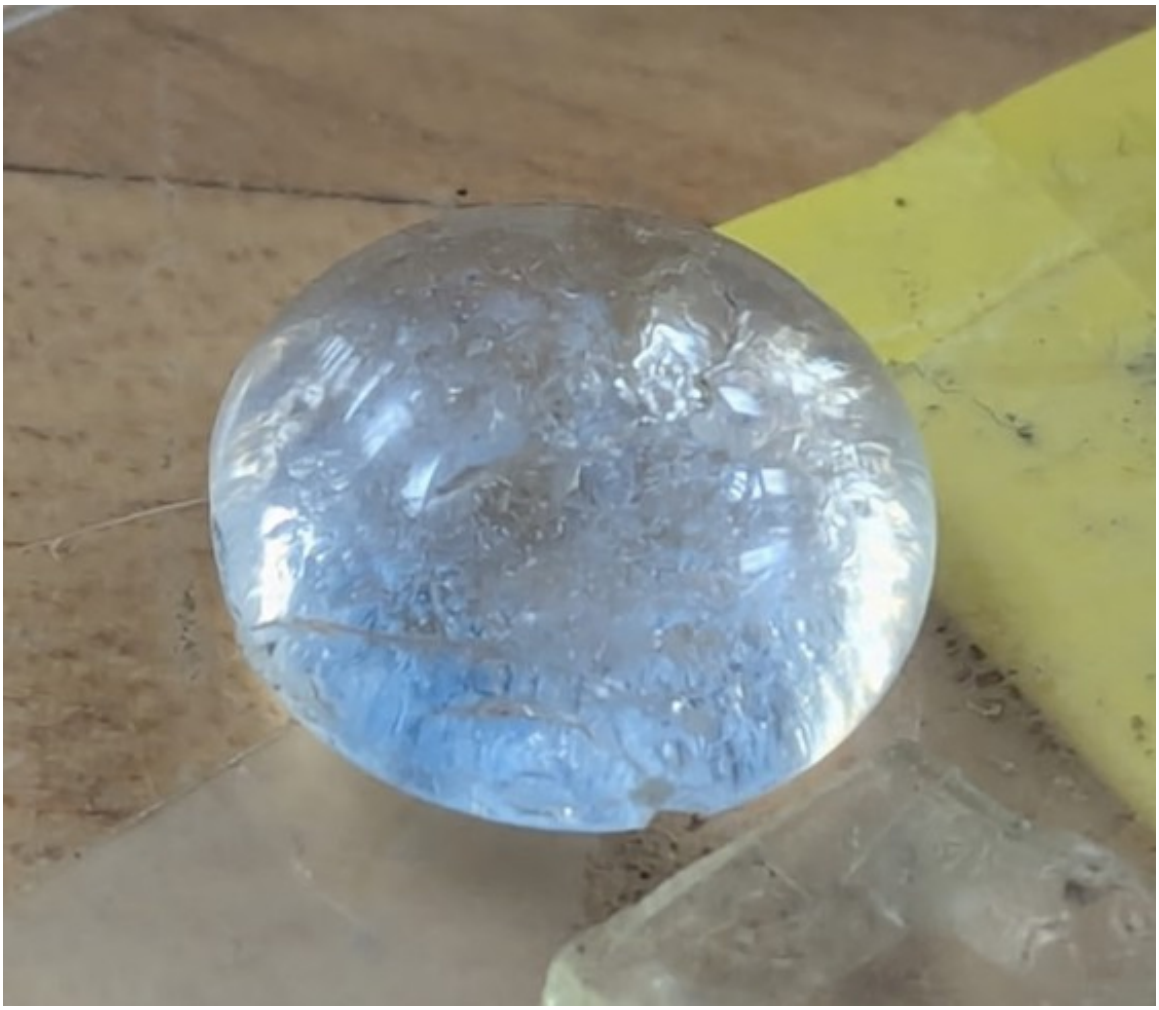

Figure 1.1 ^{241}Am doped GS20 sample

Chapter 2

Methods

The ^{241}Am doped GS20 sample was manufactured in 1985, so we did a spectral assessment of the GS20 sample to ensure it was still in condition for testing. After assessing the GS20 sample, we collected neutron detection data from the sample.

Spectral assessment of GS20

Assessment of ^{241}Am doped GS20 used the BYU Isotropic Fission Chamber (IFC). The IFC is a cylindrical chamber designed to house a variety of scintillators with a photomultiplier tube (PMT) placed below the scintillator within the tube[7]. See figure 2.1 for a full description of the IFC components. The PMT was powered with 2 kV from an Ortec 556 power supply. The sample was optically adhered to the PMT lens using optics gel to prevent scratching the PMT lens. A reflector mirror was then placed over the sample to enhance light collection. Finally, a protective sleeve was placed over the PMT, sample, and reflector to prevent light pollution interfering with data collection. The PMT signal passed through an Ortec 474 amplifier, which was used to make the signal easier to distinguish from background noise, and into an

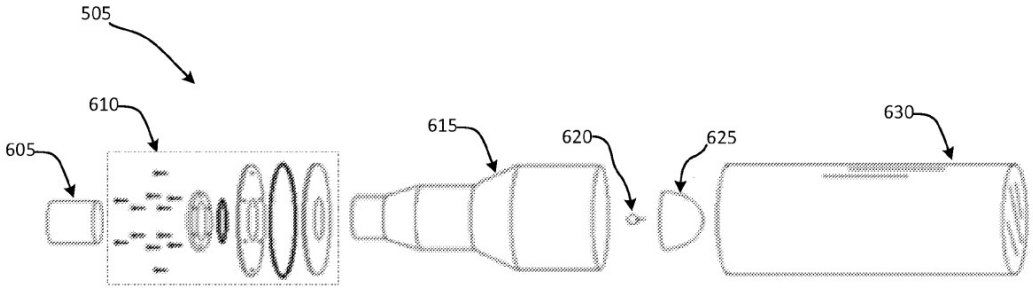

Figure 2.1 The components which make up the IFC. Item 505 is referring to the entire configuration of the IFC as a whole. Item 605 refers to a voltage divider to feed power to the photomultiplier tube. Item 610 is a sealing system to keep all materials within the chamber and to prevent light pollution from entering the chamber. Item 615 is a PMT. Item 620 is a scintillator. Item 625 is a reflector to maximize the optics of the photomultiplier tube. Item 630 is a hollowed out cylinder acting as a faraday cage[7].

Ortec Easy-Multi-Channel-Analyzer (MCA), which digitized the data and generated the GS20 spectrum. The hardware setup is depicted in figure 2.2.

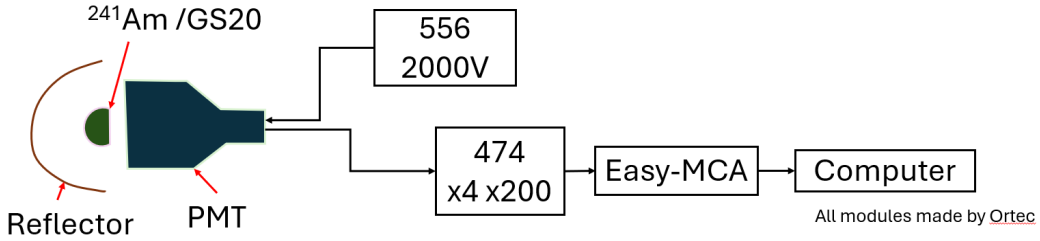

Figure 2.2 Hardware arrangement for retrieving the GS20 spectrum.

After collecting data for the spectrum of GS20, the spectrum was compared with a spectrum taken when the sample was first manufactured. There is no record of how the spectrum was taken when the GS20 was manufactured, so the comparison must be qualitative. Since the spectra have similar shapes, as shown in figure 2.3, the GS20 is deemed to be in operating condition for testing.

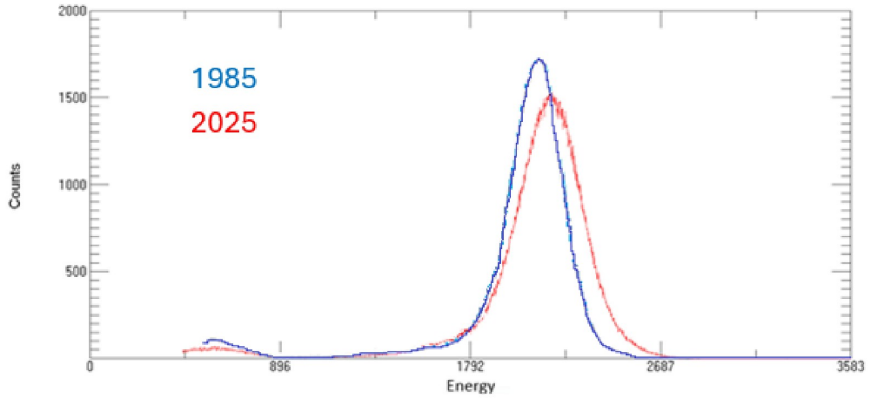

Figure 2.3 The 1985 spectrum superimposed onto the 2025 spectrum. The x-axis represents energy in keV.

Detecting Neutrons

The sample was evaluated for neutron emissions using the BYU IFC and a lithium-gadolinium-borate polyvinyltoluene (LGB-PVT) Neutron Pseudospectrometer Detector, hereafter referred to as the "neutron detector". The neutron detector has LGB crystals dispersed in a PVT disk, which fluoresces when exposed to ionizing radiation, such as neutrons. A separate PMT from the IFC detects each instance of fluorescence. The separate PMT is inside the neutron detector.

(a) Isotropic Fission Chamber **(b)** Neutron Detector

Figure 2.4 Experimental components used in the detector system.

Using the IFC, the sample is again optically adhered to the PMT lens. Once active, the chamber can detect the photons that result from interactions between ^{241}Am and the GS20. The most common decay of ^{241}Am is alpha emission, with an 84.8% chance the alpha emitted has 5.49 MeV of energy[5]. Since the GS20 is a scintillator, it will fluoresce when exposed to ionizing radiation. The fluorescence is recorded by the PMT, sending a signal to the Costruzioni Apparecchiature Elettroniche Nucleari (CAEN) desktop digitizer for computer-based analysis. The fluorescence in the data would be depicted as a sharp pulse (see figure 2.5).

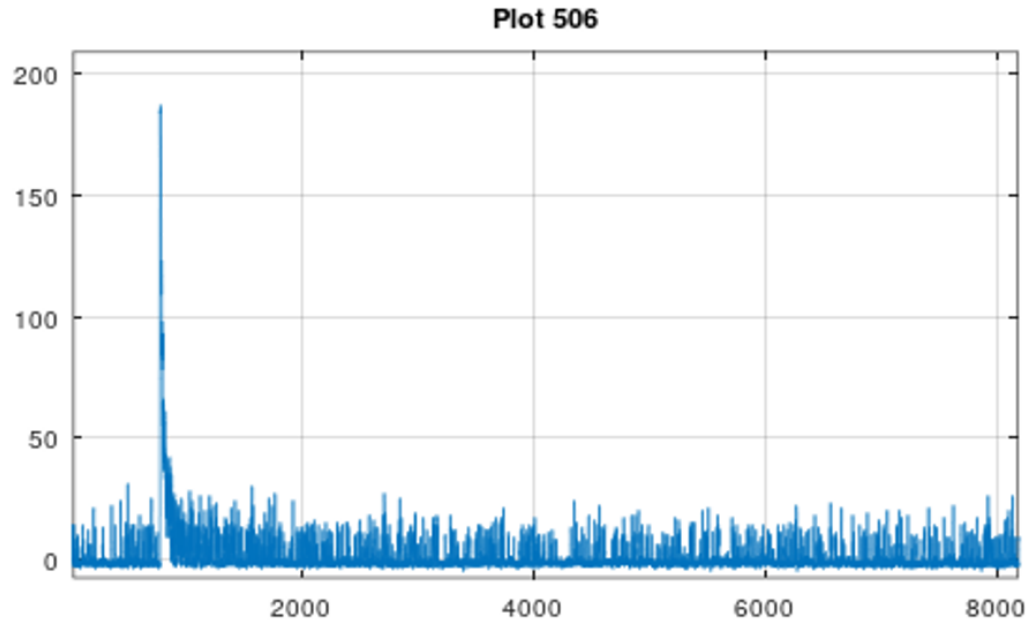

Figure 2.5 An interaction detected by the IFC. The x-axis represents time in 4ns while the y-axis is in 1/2047V.

The production of a neutron heavily depends on what GS20 isotope the ^{241}Am alpha interacts with. GS20 is comprised of the elements oxygen, silicon, magnesium, aluminum, cerium, and lithium, so an alpha from the ^{241}Am will interact with any one of these elements[8]. Table 2.1 contains information concerning elemental isotopes of GS20, the cross section of an alpha producing a neutron with the isotope, and the

Q-value (or energy release/absorption) of the (α,n) reaction using information taken from; the Nuclear Energy Agency JANIS book on alpha interactions, and the Q-value calculator and Experimental Nuclear Reaction Data from the National Nuclear Science Center respectively[9][10][11]. The isotopes used within the table were assumed to be present within the sample using their respective natural abundances.

Table 2.1 (α,n) Cross-sections and Q-values for various GS20 isotopes

Incident isotope	**Cross-section**	**Q-value**
O-17	187.5mb – 234mb	0.587MeV
O-18	379mb – 491mb	-1.53MeV
Si-29	67.5mb	-0.697MeV
Si-30	78mb	-3.49MeV
Mg-23	NA	-3.98MeV
Mg-25	150mb	2.65MeV
Mg-26	196mb	0.034MeV
Al-27	35mb	-2.64MeV
Ce-142	78mb	-3.49MeV
Li-6	7×10^{-7}fb	NA
Li-7	24mb – 145mb	-2.789MeV

The compiled table revealed that ^{17}O, ^{25}Mg, and ^{26}Mg are the most likely isotopes to release energy from the interaction with an ^{241}Am alpha. Due to low interaction probability however, large quantities of data are necessary to identify neutron events.

When neutrons interact with the LGB-PVT scintillator disk within the neutron detector by colliding with a nucleus, the neutron recoils and is then absorbed by the scintillator material. The recoil of the neutron after colliding is referred to as a thermal event, while the absorption into the scintillator constitutes a capture gate event (see figure 2.6). These collisions release photons, which are detected by the

PMT within the neutron detector. If a waveform sample from the IFC shows a pulse and shortly afterward the neutron detector finds a thermal followed by a capture gate, as in figure 2.7, it confirms a neutron was emitted from the GS20 glass.

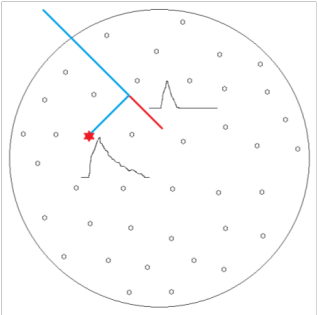

Figure 2.6 Visualization of a neutron colliding with LGB crystals and being absorbed into the scintillator.

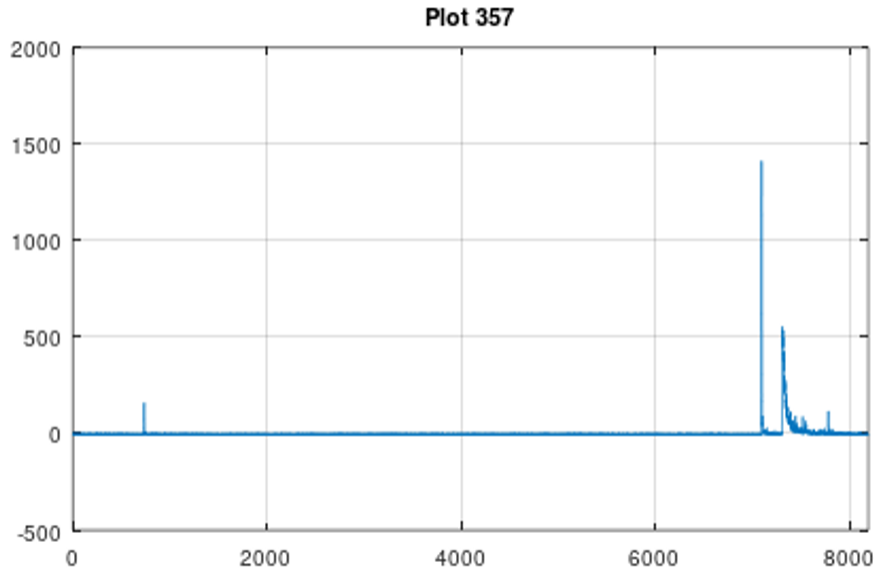

Figure 2.7 A sample of a thermal and capture gate while using ^{252}Cf. The x-axis is in 4ns and the y-axis is in 1/2047 V

To summarize the chain of events, an alpha particle from the decay of ^{241}Am interacts with a an isotope of ^{17}O, ^{25}Mg, or ^{26}Mg in the GS20, producing a neutron and a photon. The IFC will record that it saw a photon, and thus, an interaction, but doesn't know what interaction it saw. The neutron interacts with a nucleus in

the LGB-PVT disk, causing a photon to emit from the initial collision and when the neutron is absorbed into the disk. These photons are recorded by the neutron detector as a thermal event and capture gate respectively (see figure 2.7 for a visual reference of the thermal and capture gate). The full process of ^{241}Am alphas eventually causing neutron emissions to be detected is shown in figure 2.8.

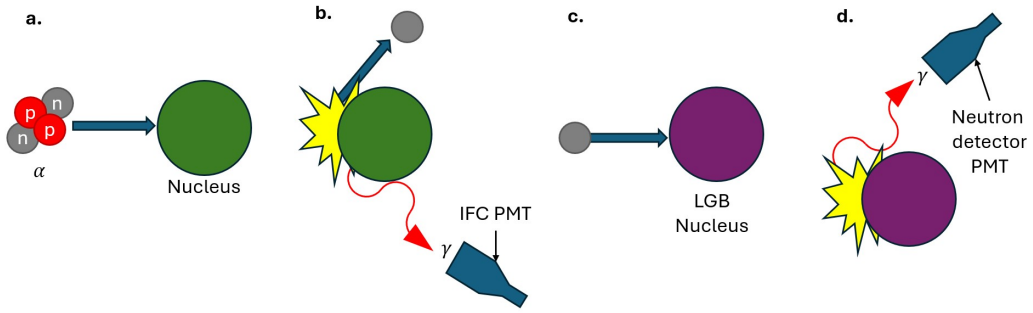

Figure 2.8 a.) An alpha particle from ^{241}Am is incident on the nucleus of an isotope in the GS20. b.) The interaction causes a neutron and photon to emit, the latter of which is detected by the IFC's PMT. c.) The scattered neutron interacts with an LGB-PVT scintillator nucleus. d.) The neutron scatters off the nucleus but is then absorbed by the LGB-PVT scintillator, emitting photons in both cases which are detected by the neutron detector PMT. Note that the schematic representation is not to scale.

The time difference between a thermal event and a capture gate could be anywhere between zero and 32 microseconds, so a histogram will be used to show the most common time differences between the thermal event and a capture gate.

The hardware setup for neutron detection is similar to the spectral analysis, but the primary difference is that the IFC no longer uses an external amplifier, and the neutron detector is now in use. The neutron detector was powered by a separate Ortec 556 and connected to the CAEN by a series of attenuators, filters, and voltage limiters as seen in figure 2.9. Once the hardware was setup and passed initial tests to ensure it was working properly, data collection started, recording all interactions over the course of a week.

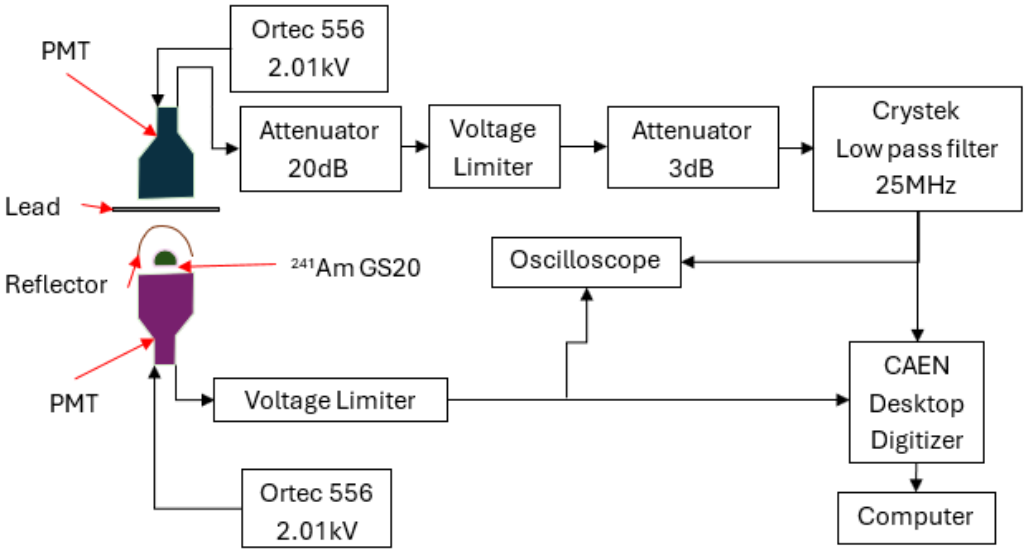

Figure 2.9 Hardware arrangement for detecting neutrons.

Chapter 3

Analysis and Results

Data analysis was performed using a custom program written in Octave, an open-source data analyzer similar to MATLAB. The code iterated through each index of time, checking for voltages higher than 0.0146 V since background radiation is unlikely to go above this threshold. Upon finding a signal with energy above the 0.0146 V threshold, the pulse width was recorded (in number of indices) along with the peak voltage. Pulses were classified using a ratio made by dividing the full width half maximum by the peak value (refer back to figure 2.7). Thermal events typically have ratios lower than 0.02 while capture gates have ratios greater than or equal to 0.02. Capture gates also needed to be at least 75 indices wide to distinguish them from noise. A simplified flowchart of the program is shown in figure 3.1, but the full program can be found in Appendix A.

After processing over one hundred thousand data points with the set criteria mentioned above, a histogram was created, showing that twenty two events were flagged as having both a thermal and a capture gate (see figure 3.2).

The "dt" in figure 3.2 represents the time difference of 4 ns between a thermal event and a capture gate. The time difference was at first recorded so analysis concerning

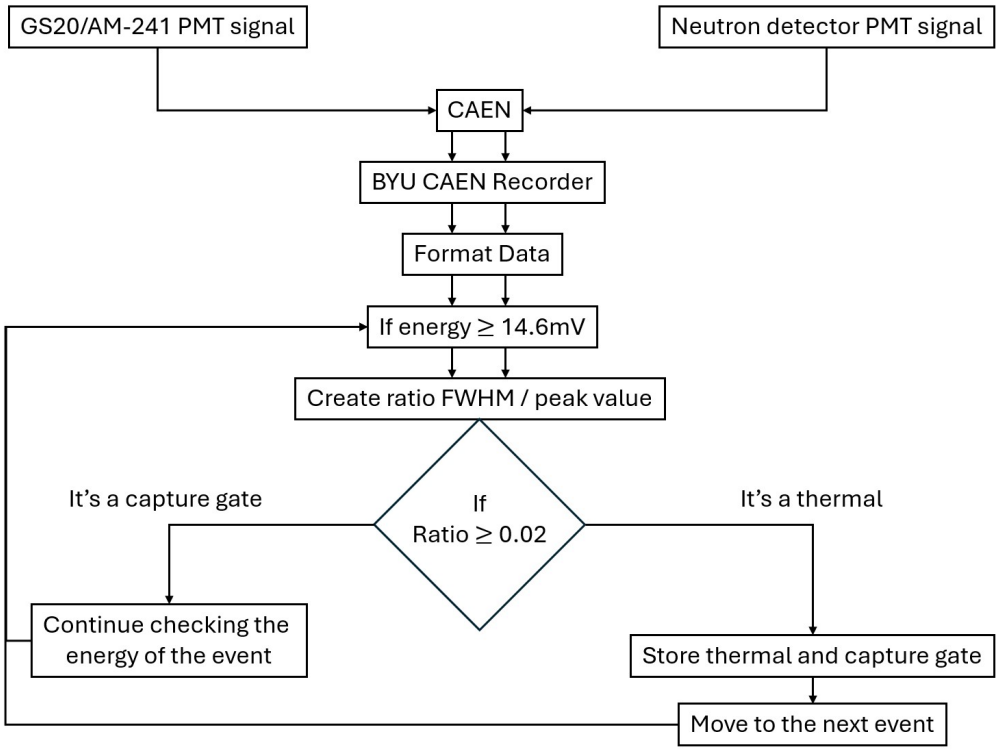

Figure 3.1 Flowchart of the code logic.

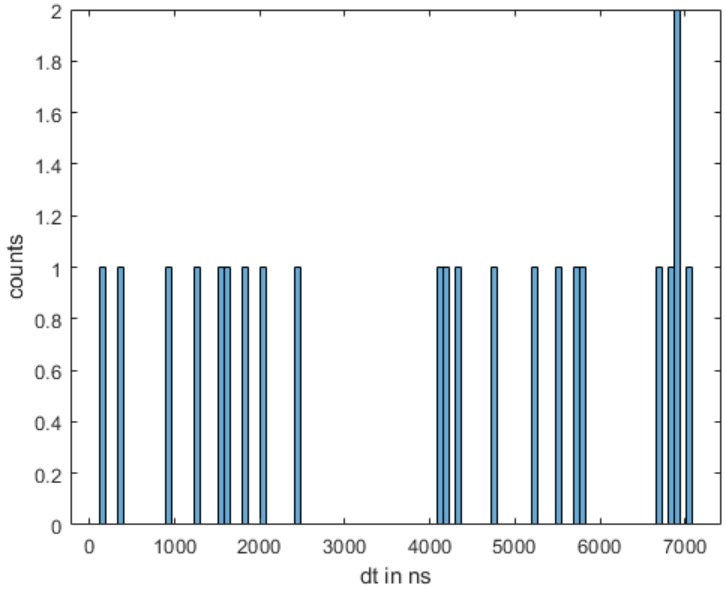

Figure 3.2 All flagged events.

the mean time difference between thermals and capture gates could be done, but with so few detected neutrons that analysis was abandoned. The twenty two events could still be used, however, to calculate the measured isotropic emission rate. With the measured isotropic emission rate, a comparison could be done with the theoretical isotropic emission rate to see if the measured value matches the predicted value. The measured isotropic emission rate is calculated as,

$$R_m = \frac{A_m}{\Omega\epsilon} \tag{3.1}$$

where R_m represents the measured isotropic emission rate, A_m is the measured activity, which is the twenty two neutrons over the 93.5 hours of measuring time, ϵ is the detector efficiency, being 5%, and Ω is the fraction of the solid angle of the detector, which is calculated as,

$$\Omega = \frac{A_d}{4\pi d^2} \tag{3.2}$$

where A_d is the detector area, which is 126.7cm^2 and d is the distance from the sample to the neutron detector (6.2cm). The theoretical isotropic emission rate is calculated as

$$R_t = A_t \sum \frac{n_i}{n_{i\text{tot}}} \frac{\sigma_i}{\sigma_{i\text{tot}}} \tag{3.3}$$

The A_t is the source activity, which uses the emission rate stated on the packaging of the sample, that being $\frac{0.125}{8}\mu$Ci. The variables n_i and $n_{i\text{tot}}$ represent the number density for a given isotope and the total number density respectively. The total number density is $2.496{\cdot}10^{22}$cm^{-3}. The cross sections σ_i and $\sigma_{i\text{tot}}$ are the (α,n) cross section and (α,tot) cross section respectively for a given isotope. The (α,tot) cross section was estimated using

$$\sigma = \pi r^2, r = (1.2\text{fm})A^{1/3} \tag{3.4}$$

The values I used for the number densities and cross sections while calculating can be found in table 3.1.

Table 3.1 Number densities and cross sections for ^{17}O, ^{25}Mg, and ^{26}Mg in GS20

GS20 Isotope	n_i	σ_i	σ_{itot}
O-17	$1.681{\cdot}10^{19}\text{cm}^{-3}$	200mb	299mb
Mg-25	$1.454{\cdot}10^{20}\text{cm}^{-3}$	150mb	387mb
Mg-26	$1.538{\cdot}10^{20}\text{cm}^{-3}$	207mb	397mb

After gathering all necessary information for the calculations, the values for the measured and theoretical isotropic emission rates were respectively,

$$R_m = 0.00498\text{Bq} \tag{3.5}$$

$$R_t = 3.42\text{Bq} \tag{3.6}$$

The measured rate differs from the theoretical rate by three orders of magnitude, making it unclear if something is wrong with our calculations or if something was wrong with the experimental set up. Regardless of which value is correct, it does cast doubt on ^{241}Am doped GS20 being a useful neutron source, as experimental neutron sources are typically on the order of 10^4-10^9 Bq for scattering and imaging.

Chapter 4

Conclusions

To summarize, the GS20 glass doped with ^{241}Am was tested using the IFC and neutron detector as a potentially more affordable isotropic neutron source. When the IFC and neutron detector are used together, it is possible to see if an interaction in the IFC is in coincidence with a detection in the neutron detector. Presently the code used to calculate the measured and theoretical isotropic emission rates continues to be updated as information from the experimental set up is recieved. The sample is likely not a good neutron source for the same reason we wouldn't use a banana for positron emission tomography, the event rate is too low for practical applications or experiments[12]. Further development of the isotropic emission rate code will reveal if the radioactivity of our sample is accurate or not.

Bibliography

[1] N.A., https://www.nuclear-power.com/nuclear-power/reactor-physics/atomic-nuclear-physics/fundamental-particles/neutron/neutron-sources/, 2021-10-13.

[2] S. F. Nowicki, S. A. Wender, and M. Mocko, "The Los Alamos Neutron Science Center Spallation Neutron Sources Physics Procedia," Physics Procedia **90,** 374–380 (2017).

[3] I. S. Anderson, R. L. McGreevy, and H. Z. Bilheux, *Neutron Imaging and Applications* (Springer, 2009).

[4] Scintacor, https://scintacor.com/technologies/neutrons/#:~:text=An%20Introduction%20to%20Lithium%2D6,7.5%25, 2006.

[5] IAEA, https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html, 2019.

[6] S. Grimes (unpublished).

[7] J. Czirr, J. Ellsworth, and L. Rees, "Isotropic Fission Chamber,", patent WO2016077591.

[8] Y. Song, J. Conner, X. Zhang, and J. P. Hayward, "Monte Carlo simulation of a very high resolution thermal neutron detector composed of glass scintillator microfibers," Applied Radiation and Isotopes **108,** 100–107 (2016).

[9] O. N. E. Agency, https://www.oecd-nea.org/jcms/pl_44624/janis-books, n.d.

[10] N. N. D. Center, https://www.nndc.bnl.gov/qcalc/, n.d.

[11] N. N. D. Center, https://www-nds.iaea.org/exfor/, n.d.

[12] D. W. Ball, "How Radioactive is your banana," Journal of Chemical Education **81,** 1440 (2004).

Appendix A

Event Sorting Code

The code required four files. The first file, `Neutronv3.m`, takes an array of data and analyzes which ones have both thermals and capture gates:

```
% The code that follows is meant to take in data collected
   from the BYU Neutron Detector
% This is done by iterating through time and energy on each
   event and taking the full width
% half max of each peak to distinguish between thermals (
   narrow and thin peaks) and capture
% gates (an exponential decay). This code will first try to
   find a capture gate, then a thermal
% Don't forget that each graph that shows both a thermal and a
    capture gate means you likely
% detected a neutron. Ai was used to help write and correct
   some parts of the code.
%
% Author: Jarom Peterson, with help (and some code borrowed
   from) Isaac Wilden and ChatGPT
```

```
% Authors email: jaysworld0604@gmail.com
%
% Last edit: 8th Augest 2025

% Close all figures and clear all variables and load in the
   parallel package
delete(gcp('nocreate'));
clear all; close all; clc;

% Variables
leng = 8192; % Length of each event
minAmp = 30; % Minimum peak to consider when processing
channeL1 = 1; % Channel number 0 on the CAEN
channeL2 = 2; % Channel number 1 on the CAEN
number_of_thermals = 0; % This tells us the number of thermals
    the code found. Used to trim some arrays later for
   analysis
number_of_CG = 0; % This tells us the number of capture gates
   the code found. Used to trim some arrays later for analysis
max_peak_width = 200; % None of the thermals or CG's should be
    larger than 200 indicies
coincidence_energy_threshold = 50; % Minimum peak height to be
    considered a coincidence
look = 2000; % number of indicies to check around the thermal
number_of_differences = 0; % Used to trim the dt_values array
   later
```

```
% Open and read the file
event = readFileCopy(); % Every waveform is considered an
    event
ii = length(event); % The number of events
evnt = zeros(ii,leng); % Create an array of number of events (
    see variable ii) as rows and the length of time each event
    is recorded (see variable leng)
for j = 1:ii
  evnt(j,:) = -(event(j).ch(:,channeL1))';
end

% Arrays
thermal_ratio = zeros(1,ii); % The ratios of full width half
    max divided by the max value of thermal are stored here for
     analysis
thermal_time = zeros(1,ii); % The indicies  for the tail of a
    thermal is stored here
capture_gate_values = zeros(1,ii); % The ratios of full width
    half max divided by the max value of capture gates are
    stored here for analysis
capture_gate_times = zeros(1,ii); % The indicies  for the tail
     of a capture gate is stored here
thermals = zeros(1,ii); % Stores which samples had both a
    capture gate and thermal
peak_indexes = zeros(1,ii); % Stores the index of the left
    half max for thermal peaks
```

```
thermals_with_coincidence = []; % Used for trimming an array
    later
results(ii) = struct('has_thermal', false, 'has_CG', false, '
    thermal_ratio', 0,...
                     'thermal_time', 0, 'CG_ratio', 0, '
                        CG_time', 0, 'peak_index', 0); % The
                        structure here is what stores
                        information about an event. You can
                        think of it as a class using c++ terms
                        % The information stored is whether or
                            not an even has a thermal and
                            capture gate, what their ratios
                            were, and where they were
                        % It is important to remember this is
                            an array (list in python terms) of
                            structures (or an array containing
                            objects of classes)



%%

tic; % Start tracking time

% The samples are all evaluated in this parfor loop. Check the
    backwards_analyze_event file for what evaluate means
% Process each event in parallel. This way the code runs
    faster.
```

```
parpool;%pkg load parallel;
parfor j = 1:ii
  if mod(j, 1000) == 0
    disp([ Sample  , num2str(j)]); % Primarily here to act as
       a loading bar so we can see how much has been processed
        in the command window
  end
  results(j) = Backwards_analyze_event_v3(evnt(j,:), minAmp,
     max_peak_width); % Take the jth indicie in this array and
      pass this information into the backwards_analyze_event
     function
                          % The informatio stored into the
                             results variable gets changed as a
                             result of this function
end

% Here is where any event that has a thermal or capture gate
   are saved into their respective arrays
for j = 1:ii
  if results(j).has_thermal
    number_of_thermals = number_of_thermals + 1; % The
       number_of_thermals variable is used to trim some arrays
        later
    thermal_ratio(number_of_thermals) = results(j).
       thermal_ratio; % Save the thermal_ratio from results
       event j into the thermal_ratio array
    thermal_time(number_of_thermals) = results(j).thermal_time
```

```
        ; % Save the thermal_time from results event j into the
         thermal_time array
    thermals(number_of_thermals) = j; % Save which event had a
         thermal
    peak_indexes(number_of_thermals) = results(j).peak_index;
       % Save the index of the left half max on the thermal
       into the left_half_maxes array
    capture_gate_values(number_of_thermals) = results(j).
       CG_ratio; % Save the CG_ratio from results event j into
         the CG_ratio array
    capture_gate_times(number_of_thermals) = results(j).
       CG_time; % Save the CG_time from results event j into
       the CG_time array
  end
end

% Trim the arrays so that statistics can be done to them (
   because otherwise an error would occur due to there being
   empty spaces in the array)
thermal_ratio = thermal_ratio(1:number_of_thermals);
thermal_time = thermal_time(1:number_of_thermals);
thermals = thermals(1:number_of_thermals);
peak_indexes = peak_indexes(1:number_of_thermals);
capture_gate_values = capture_gate_values(1:number_of_thermals
   );
capture_gate_times = capture_gate_times(1:number_of_thermals);
```

```
% Here is where we start finding the time difference dt
   between a thermal
% and a coincidence peak to do statistics on the dt's

dt_values = nan(1, length(thermals));

for i = 1:length(thermals)
    sample = thermals(i);
    V = -(event(sample).ch(:, channeL2))';
    peak_idx = peak_indexes(i);

    % Search range
    start_idx = max(1, peak_idx - look);
    end_idx = min(length(V), peak_idx + look);

    % Find all indices where V exceeds threshold
    candidate_idxs = find(V(start_idx:end_idx) >=
       coincidence_energy_threshold);

    if ~isempty(candidate_idxs)
        % Convert to absolute indices
        candidate_idxs = candidate_idxs + start_idx - 1;
        % Find the one closest to peak_idx
        [~, min_pos] = min(abs(candidate_idxs - peak_idx));
        closest_idx = candidate_idxs(min_pos);

        % Store time difference in samples
```

```
        dt_values(i) = closest_idx - peak_idx;
        number_of_differences = number_of_differences + 1;
        thermals_with_coincidence(end+1) = sample;
    end

    %{
    % Calculate the starting index, making sure  i t s  >= 1
    start_idx = max(1, index_here - look);

    % Extract the segment 25 indices before the half max
    segment = V(start_idx:index_here);

    while max(segment) >= coincidence_energy_threshold

    %}
    %{
    % Check if there is any peak >= threshold
    if any(segment >= coincidence_energy_threshold)
        % Time difference is thermal half max index minus
           coincidence index
        [~, idx_rel] = max(segment); % Get location of max in
           the segment
        idx_abs = start_idx + idx_rel - 1; % Convert to
           absolute index
        dt = left_index - idx_abs; % Difference in indices
        dt_values(i) = dt;
        number_of_differences = number_of_differences + 1;
```

```
        thermals_with_coincidence(end+1) = sample;
    end
    %}

    %{
    Get the segment in the window V(left-look:right+look)

    while there are segments greater than the energy threshold
        get the index and value of the max(segment)
        save in a temporary variable (tempIndex)
        save the difference to a temporary array
        truncate the segment: segment = segment(tempIndex+5:)

     this should loop until there are no more peaks. Then find
         the smallest
     difference and save that in a permanent array

    %}
end

% Trim the dt_value array for statistical analysis
% dt_values = dt_values(1:number_of_differences);

% These lines just show what samples have thermals
disp( There are coincident peaks on )
disp(thermals_with_coincidence)
```

```
% Below here is where all the plotting takes place
startIndex = 1; % Change this number to an indicie you wish to
    plot and the next eight after it
endIndex = min((startIndex+8), length(thermals));  % Ensure
   you don't go past the end of the array

% Plotting and synching the x-axis on two figures
% The two figures show 9 samples to easily view multiple at
   once
nPlots = endIndex - startIndex + 1; % This tells the for loops
    used for graphing how many plots we want

% Plot Neutron Detector
figure;
ax1 = gobjects(1, nPlots); % Preallocate for axes handles
for i = 1:nPlots
    sample = thermals(startIndex + i - 1); % This retrieves
       the sample number
    V = -(event(sample).ch(:,channeL1))'; % Energy in 1/2047 V
    x = 1:length(V); % Time in 4ns (so 2 would be 8ns, 3 is 12
       ns, etc.)

    ax1(i) = subplot(3,3,i); % Arranges the plots so that
       multiple can display
    plot(x, V, 'o-', 'MarkerSize', 3, 'LineWidth', 1);
    xlabel( Time in 4ns );
    ylabel( Energy in 1/2047 V );
```

```
    xlim([0, length(V)]); % Cut off the x axis at length(V1)
    title([ Neutron detector plot  , num2str(sample)]);
    grid on; % Enable grid
end

% Plot Isotropic Fission Chamber
figure;
ax2 = gobjects(1, nPlots); % Preallocate for axes handles
for i = 1:nPlots
    sample = thermals(startIndex + i - 1); % This retrieves
       the sample number
    V = -(event(sample).ch(:,channeL2))'; % Energy in 1/2047 V
    x = 1:length(V); % Time in ns

    ax2(i) = subplot(3,3,i); % Arranges the plots so that
       multiple can display
    plot(x, V, 'o-', 'MarkerSize', 3, 'LineWidth', 1);
    xlabel( Time in 4ns );
    ylabel( Energy in 1/2047 V );
    xlim([0, length(V)]); % Cut off the x axis at length(V1)
    title([ Isotropic Fission Chamber plot  , num2str(sample)
       ]);
    grid on; % Enable grid
end

% Link the two figures
linkaxes([ax1, ax2], 'x');
```

```
%{
% Number of events to plot at once
nPlots = 1;  % Each event takes 2 rows

figure;
ax = gobjects(2, nPlots); % store axes handles for linking

% Plot and sync the x-axis, but the two device measurements
   are on the same
% figure.
for i = 1:nPlots
    sample = thermals(startIndex + i - 1); % This retrieves
       the sample number

    % This variable helps position things in the subplot
    rowOffset = (i - 1) * 2;

    % --- Neutron detector ---
    V1 = -(event(sample).ch(:, channeL1))'; % Energy in 1/2047
        V
    x = 1:length(V1); % Time in 4ns (so 2 would be 8ns, 3 is
       12ns, etc.)
    ax(1,i) = subplot(nPlots * 2, 1, rowOffset + 1); %
       Arranges the plots so that multiple can display
    plot(x, V1, 'o-', 'MarkerSize', 3, 'LineWidth', 1);
    xlabel( Time in 4ns );
```

```
    ylabel( Energy (1/2047 V) );
    xlim([0, length(V1)]); % Cut off the x axis at length(V1)
    grid on; % Enable grid
    title(sprintf( Neutron Detector - Event %d , sample));

    % --- Isotropic fission chamber ---
    V2 = -(event(sample).ch(:, channeL2))'; % Energy in 1/2047
        V
    ax(2,i) = subplot(nPlots * 2, 1, rowOffset + 2); %
       Arranges the plots so that multiple can display
    plot(x, V2, 'o-', 'MarkerSize', 3, 'LineWidth', 1);
    xlabel( Time in 4ns );
    ylabel( Energy (1/2047 V) );
    xlim([0, length(V2)]); % Cut off the x axis at length(V2)
    grid on; % Enable grid
    title(sprintf( Isotropic Fission Chamber - Event %d ,
       sample));
end

% Link all subplots' x-axes
linkaxes(ax(:), 'x');
%}

figure;
histogram(dt_values.*4,100)
xlabel( dt in ns )
ylabel( counts )
```

```
toc; % Stop tracking time and print how long it took to
   execute code between tic and toc
```

Listing A.1 Neutronv3.mstyle

The second file, `Backwards_analyze_event_v3.m`, defines the struct (essentially a class in c++ or python terms):

```
% This function is meant to be used in tandem with
   processNeutron_BackwardsOptimize_Max.m
% The code below will take events stored in the results array
   from the above mentioned file
% whereupon it iterates through each moment in time, checking
   the energy at each instance.
% Once the energy is above minAmp, the indicies are recorded
   so that it can take the full
% width half max of the peaks it finds. After getting the
   value of FWHM it is divided by the
% value of the peak to make a ratio. The ratio is what reveals
    if it is a thermal or capture
% gate. The information is then saved and returned into the
   results(j) structure of the
% processNeutron_BackwardsOptimize_Max.m file
%
% Author: Jarom Peterson
% Authors email: jaysworld0604@gmail.com
%
% Last edit: 7th Augest 2025
```

```
function result = Backwards_analyze_event_v3(ev, minAmp,
   max_peak_width)
  % The following varaibles are part of a local structure here
      in this file
  % When the code reaches the endfunction statement at the
     bottom of the
  % document, the values of these variables are what get
     returned to the
  % results(j) structure.
  result.has_thermal = false; % used to say if a thermal is
     found
  result.has_CG = false; % Used to say if a capture gate is
     found
  result.thermal_ratio = 0; % value of the ratio for a thermal
  result.thermal_time = 0; % value of the last time step the
     thermal was found
  result.CG_ratio = 0; % value of the ratio for the capture
     gate
  result.CG_time = 0; % value of the last time step the
     capture gate was found
  result.peak_index = 0;

  % Check to ensure the data is not just noise
  if ev(1) >= minAmp
    return;
  end
```

```
% Initialize a few more local variables
t = 7400; % length of time in 4ns (so 3 would represent 12ns
   ). This used to be equal to leng, but after seeing some
   data, this value should be fine
CG_found = false; % This variable will be used for some
   logic later
thermal_found = false; % This will be used for logic later

% Analysis of the event begins here
while t >= 1 && ~thermal_found % while time is greater than
   or equal to 1 and no thermal was found
  if ev(t) >= minAmp % Check if the energy is above the
     minimum amplitude
    tcount = 1; % tcount will be used to figure out how wide
        the peak is
    thermalcheck = zeros(1, max_peak_width); % An array used
        to hold the time stamps of a peak

    while t >= 1 && ev(t) >= minAmp && tcount <
       max_peak_width
      thermalcheck(tcount) = ev(t); % add this time into the
          thermalcheck array
      t = t - 1; % move onto and check the next instance of
         time
      tcount = tcount + 1; % increment the number if
         indicies this peak has
```

```
end

tcount = tcount - 1; % step back one step in time so
   that t is now equal to the last recorded value
if tcount < 1 % if there are no recorded indicies, skip
   this peak
  continue;
end

% The process of finding FWHM starts here
segment = thermalcheck(1:tcount); % The array is now
   made into a shorter array called segment so analysis
   can be done on it
[peak, peak_idx] = max(segment); % Find the highest
   value of the peak that was found
half_max = peak / 2; % Divide the max value of the peak
   by 2

% With half max found, the code will now iterate through
    the left and right sides of the peak until below
   half the peak value
left = peak_idx;
while left > 1 && segment(left) > half_max
  left = left - 1;
end

right = peak_idx;
```

```
while right < tcount && segment(right) > half_max
  right = right + 1;
end

% The width of the peak is found here then divided by
   the value of the peak to make a ratio
FWHM = right - left;
ratio = FWHM / peak;

% Here is where we check for capture gates and thermals
if ratio >= 0.02 && tcount >= 75 % If the ratio is
   larger than or equal to 0.02 and has at least 75
   indicies in its width, its a cpature gate
  result.has_CG = true;
  result.CG_ratio = ratio;
  result.CG_time = t;
  CG_found = true;
elseif ratio < 0.02 && tcount > 4 && tcount < 75 &&
   CG_found % If the ratio is less than 0.02, a capture
   gate was found, and the indicies are between 4- 75,
   it must be a thermal
  result.has_thermal = true;
  result.thermal_ratio = ratio;
  result.thermal_time = t;
  thermal_found = true;
  result.peak_index = peak_idx;
end
```

```
    else
      t = t - 1;
    end
  end
end
```

Listing A.2 Backwards_analyze_event_v3.m

The third file, `readFileCopy.m`, reads files to store data in an array:

```
function event = readFileCopy()

% Open the folder and find the files. Select all the files you
    want to use
dirs = uigetdir('','Click␣on␣directory␣so␣file␣names␣show');
files = cellstr(uigetfile([dirs,'\*.*'],'Highlight␣all␣files␣
   of␣interest', 'MultiSelect', 'on'));
ii=0;

% Reads the files and stores the data into the event array
for fCntr = 1 : length(files)  %get the number of data files
   and arrange them into a vector
    fileName = [dirs, '/', char(files(fCntr))];  %create an
       array of data file names
    fid = fopen(fileName,'r'); % Open the file for reading

    fseek(fid,0,'eof'); % Selects everything from the start to
        the end of the file
    fileLength = ftell(fid); % Finds the total file size in
       bytes based on everything selected
```

```
    fseek(fid,0,'bof'); % Resets back to the start of the file


    while ftell(fid) < fileLength - 1
        ii = ii+1;
        event(ii) = fEventRead8Copy(fid); % Read one event at
           a time
    end


end
fclose('all'); % Close all open files


end
```

Listing A.3 readFileCopy.m

The fourth file, `fEventRead8Copy.m`, handles event file parsing:

```
function event = fEventRead8Copy(fid)
%capture CAEN header values  per UM3244-DT5720 User Manual rev
   .10 p27
event.header = fread(fid, 4, 'uint32');
    %CAEN DATA EVENT format, Standar Mode, Normal Format (4
       32-bit words):
    % 31302928 27262524 23222120 19181716 15141312 1110 9 8  7
        6 5 4  3 2 1 0
    %  1 0 1 0  * * * *  * * * *  * *  EVENT SIZE   * * * *  *
        * * *  * * * *
    %    RESERVED     0          RESERVED                      *
        CHANNEL MASK *
    % RESERVED           * * * *  * EVENT COUNTER   * * * *  *
```

```
        * * *  * * * *
    %  * * * *  * * * *  * * * TRIGGER TIME TAG  *  * * * * *
        * * *  * * * *
event.patA = bitand(event.header(1),4026531840);   %1111 0000
   0000 0000 0000 0000 0000 0000
event.Size = bitand(event.header(1),268435455);    %0000 1111
   1111 1111 1111 1111 1111 1111
event.BoardID = bitand(event.header(2),4160749568);%1111 1000
   0000 0000 0000 0000 0000 0000
event.ChannelMask = bitand(event.header(2),255);   %0000 0000
   0000 0000 0000 0000 1111 1111
event.Counter = bitand(event.header(3),16777215);  %0000 0000
   1111 1111 1111 1111 1111 1111
event.TrigTime = event.header(4);                  % all 32
   bits

%Determine channel count
ChCnt = 0;
if bitand(event.ChannelMask,1) == 1
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,2) == 2
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,4) == 4
    ChCnt = ChCnt+1;
end
```

```
if bitand(event.ChannelMask,8) == 8
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,16) == 16
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,32) == 32
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,64) == 64
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,128) == 128
    ChCnt = ChCnt+1;
end

%Determine waveform length in 16 bit (12 bits used) words
length = ((event.Size - 4)/ChCnt)  *2;
%length = 1248

%read channel data enabled in channel mask, leaving no empty
   waveform channels
for i = 1:8  %4
  if(bitand(event.ChannelMask,(2^(i-1))) ~= 0) %only used
     channels
    %event.ch(:,i) = fread(fid, length, 'uint16');
    event.ch(:,i) = fread(fid, length, 'uint16') - 2047;
```

```
  end
end
end
```

Listing A.4 fEventRead8Copy.m