INVESTIGATING CORRELATED NEUTRONS FROM PULSED PHOTONUCLEAR INTERROGATION FOR TREATY VERIFICATION APPLICATIONS

A Thesis

by

SCOTT LAWRENCE STEWART

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	William S. Charlton
Committee Members,	Craig M. Marianno
	Don T. Phillips
Head of Department,	Yassin Hassan

August 2013

Major Subject: Nuclear Engineering

ABSTRACT

The treaty verification field is of renewed importance as continued nuclear weapons disarmament is prioritized nationally in partnership with other nuclear weapons states. This interest has led to research and development on technologies that could support future U.S. verification missions. A technology employing pulses of high-energy photons from an electron linear accelerator is one technique under consideration. High-energy photons are advantageous as an interrogation source because they penetrate thick shielding and can generate neutrons inside a measurement object. The neutrons would then multiply when presented with an object containing fissile material and allow for detection in a time domain immediately after the pulse. The purpose of this work was to develop an understanding of neutron behavior following a high-energy photon pulse and then develop a tool set to analyze data from this region to determine if a measurement object contains multiplying material, the mass of that material if present, and the moderation in the measurement object. Results indicate the tool sets developed were able to determine multiplication was present accurately in 3 out of 4 realistic verification objects. Additionally the state of the moderation in each object was able to be determined, and the mass could potentially be determined by calibrating to representative samples.

DEDICATION

I am thankful for the support of members of my faith community, my family, and my friends for their support of me over the years. I know I can always count on these groups of people to provide words of encouragement when needed. I would especially like to thank my parents for instilling in me a desire to seek out a field that allows me to serve others, for teaching me to work hard, and for providing encouragement and advice constantly. I have also been fortunate to have many individuals encourage my continued participation in safeguards science and technology. Thanks to Greg Sheppard for introducing me to the field and helping me to find a home for my interests when I was searching and to Martyn Swinhoe for being an excellent mentor over the years.

ACKNOWLEDGEMENTS

I would especially like to thank all the members of NEN-1 at Los Alamos National Laboratory for their input and ideas throughout this project. Jonathan Thron and Martyn Swinhoe in particular were very supportive of these efforts. I would also like to thank my committee chair, Dr. Charlton, for his continued input and ideas on project direction as well as his advice and suggestions throughout the graduate school process.

NOMENCLATURE

A Gate	Accidentals Gate
D/S	Doubles Divided by Singles
DU	Depleted Uranium
DU+p	Depleted Uranium with Polyethylene
FORTRAN	Formula Translation
He-3	Helium-3
HEU	Highly Enriched Uranium
HEU+p	Highly Enriched Uranium with Polyethylene
INL	Idaho National Laboratory
IO	Inspection Object
LANL	Los Alamos National Laboratory
LINAC	Linear Accelerator
LSF	Least Squares Fitting
MCNP	Monte Carlo Neutral Particle Transport
NAVI	Nuclear Arms Verification Instrument
NMIS	Nuclear Material Identification System
PANDA	Passive Nondestructive Assay of Nuclear Materials
PND	Photonuclear Neutron Detector
PSR	Pulsed Shift Register
Pu	Plutonium

Pu+p	Plutonium with Polyethylene
R+A Gate	Reals plus Accidentals Gate
R&D	Research and Development
SPNS	Simple Neutron Simulation
START	Strategic Arms Reduction Treaty
TTL	Transistor-Transistor Logic
U-235	Uranium-235
UK	United Kingdom
US	United States of America
W	Tungsten
W+p	Tungsten with Polyethylene

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

I.A. Motivation

The current administration under President Obama has made it an objective to reduce and eliminate nuclear weapons throughout the world. This objective has been pursued through a new STrategic Arms Reduction Treaty (START) with Russia that was signed on April 8, 2010 and entered into force on February 5, 2011. The START treaty is a bilateral agreement that limits the numbers of strategic weapons held by both countries. The Obama administration has also been discussing world-wide participation in a Fissile Material Cutoff Treaty (FMCT) that would freeze the current global stockpiles of nuclear materials for nuclear weapons by prohibiting the production of more material.[1]

A recent US Nuclear Posture review released on April 6, 2010 has committed the US to developing treaty verification and arms control programs to strengthen the R&D efforts in these fields in order to support future US treaty obligations[1]. A pulsed high-energy photon accelerator coupled with neutron detection capability is one potential technology under consideration for future treaty verification applications. This technique has shown promise in previous work in detecting shielded nuclear weapons objects, which is ideal since many nuclear weapons packages are shielded by material within the device as well as by materials within the delivery vehicle[2].

I.B. Treaty Verification Background

The impetus for treaty verification technologies is to be able to certify the disarmament of nuclear weapons in line with treaty obligations[1]. The exact procedure of verifying disarmament is always negotiated on a per treaty basis between the various nations involved in the treaty writing process. In order to facilitate this negotiation, it helps to have a number of viable verification technologies in reserve should any one technology prove unpalatable to any of the negotiators.

Since all armed nuclear weapons have a radiological signature, this signature is often used to acknowledge its disarmament. Until very recently, most significant development in arms control and treaty verification technology occurred in the 1990s[1]. Most of the notable development during this period was tied into various verification exercises that occurred at the Francis E. Warren Air Force Base as well as at the Los Alamos Simulation Facility in Technical Area 18 at Los Alamos National Laboratory (LANL)[3].

One of the early prominent nuclear verification devices was the Nuclear Arms Verification Instrument (NAVI). This instrument employed both singles neutron and gamma counting to verify the disarmament of weapons devices. For neutron counting, the device relied on He-3 detectors with polyethylene moderation, and it used a low resolution gamma detector for gamma spectral analysis. Gamma spectroscopy in particular has a high probability of generating classified data, so the detection equipment was coupled with a simplified operator interface and a yes/no light to indicate whether an inspection object was a weapon or not. This simplification essentially served as an

information barrier to prevent unauthorized individuals from accessing classified information. Later technologies continued to utilize the basic concepts incorporated into the NAVI design, although several other techniques were investigated, such as scintillation technology for neutron detection, infrared imaging systems, and 2D gamma imaging technology.[3]

More modern efforts have again returned to singles neutron and gamma counting techniques, but have more specifically focused on adapting portal monitoring technology to fill roles in potential treaty verification regimes[4]. Another research effort that has been ongoing for the past 20 years and is being examined for potential treaty verification applications is the Nuclear Material Identification System (NMIS) which uses neutron transmission imaging technology [5,6].

I.C. Research Objectives

The goal of this thesis is to ascertain the viability of neutron analysis in the pulsed region of data generated from a pulsed high-energy photon source for use in treaty verification activities. In order to achieve this goal, the physics occurring in the pulsed region was examined to correlate measurement object behaviors to perturbations in analysis parameters. To assist in this process, a point model Monte Carlo code was developed and coupled with Monte Carlo Neutral Particle Transport (MCNP) analysis. Once these investigative tools provided greater insight into the physics occurring, a shift register analysis tool was developed to analyze experimental neutron data acquired at Idaho National Laboratory (INL) as well as simulated neutron data from the point model Monte Carlo tool. Both of these tools were then used to determine if neutron data from

the pulsed region could determine whether or not there was fissile material present in a measurement object. If fissile material was present, then the tools were used to ascertain the mass and moderation state of the fissile material. The capabilities of these analysis tools as well as the relative state of the technology contributed to the recommendations on the viability of this method in the conclusions of the thesis.

CHAPTER II

EXPERIMENTAL SETUP

Data for this research comes from two different measurement campaigns at INL. The first set of data comes from a joint US and UK measurement campaign from 20-23 September 2010. The methodology and setup for these experiments are detailed extensively[7,8], and the data was provided to assist in the work covered in this thesis. The second set of data used in this analysis comes from a measurement campaign undertaken from the 25-27 June 2012. The experimental setup for the second measurement campaign is detailed in the rest of this section.

The pulse in this experiment was provided by an electron linear accelerator (LINAC) with an S-Band Radio Frequency standing wave design that uses a Varian L-3000 waveguide. The LINAC is depicted in Figure 1. In the figure, some of the cooling equipment can be seen towards the edge of the image. The LINAC's waveguide is enclosed within the yellow box labeled "Varitron." The two aluminum boxes on the front of the Varitron contain two calibrated red lasers that are used to ensure that samples are in line vertically and horizontally with the Varitron before data acquisition begins. The very end of the collimator can be seen just behind the lead bricks in the figure. This LINAC design is capable of producing electrons between 2 and 12 MeV, and was operated to produce 10 MeV electrons for all data included in this work. The LINAC was characterized as part of the US-UK measurement campaign, and the beam current of the LINAC was determined to be around 3 microamperes at 125 Hz. This

produces around 1.5 x 10^{11} electrons per pulse and 125 pulses per second[7]. The LINAC was operated in the same regime for the set of measurements in the June 2012 measurement campaign, and so the electrons produced in each pulse were comparable to the value from the US-UK measurement campaign.

Figure 1. The Varitron electron LINAC[2]

After production and acceleration, the electrons from the LINAC are sent into a tungsten converter to produce Bremsstrahlung photons. The Bremsstrahlung process produces gammas as a result of the electric field surrounding an atomic nucleus

redirecting the electrons from the LINAC. As the electrons are redirected, they lose kinetic energy through photon radiation[9]. This leads to a photon spectrum as seen in Figure 2, where the highest possible photon energy is based on the energy of the electrons from the LINAC. In this figure, the 0^{o}-5^{o} mark indicates production of photons that are in the same line as the incident electrons.

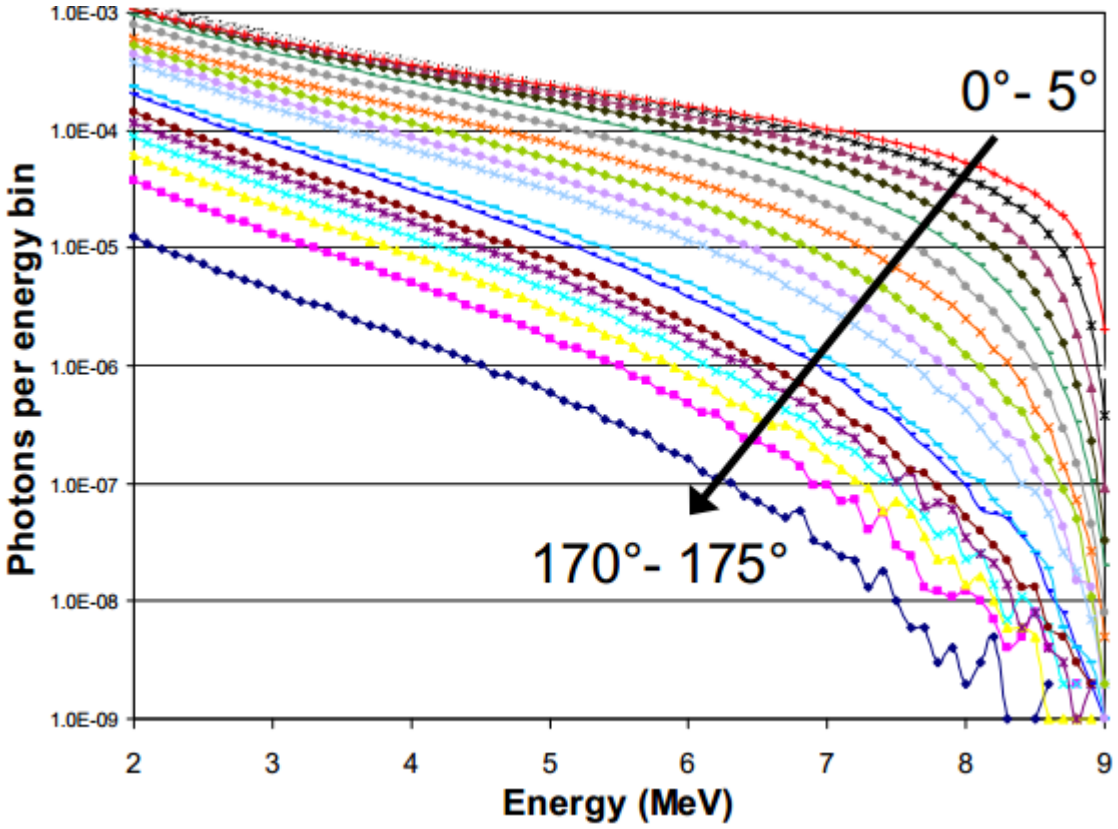

Figure 2. Bremsstrahlung spectra demonstrating the angular dependence of the energy of photons given a 9MeV electron beam incident on tungsten[10]

The photons from the tungsten converter are then collimated toward the measurement object. The energy distribution from the photons out of the collimator is not uniform. Instead the highest energy photons are preferentially located in the line of

the LINAC to the measurement object, and, on average, the photon energy decreases as they diverge from this line. Figure 2 also shows the non-uniformity of the photon flux from a Bremsstrahlung source depending on the angle.

The collimator of the LINAC is 1 m from the measurement object. After traveling the distance between the LINAC and the measurement object, photons generate neutrons in the object through photoneutron and photofission interactions. A 10 MeV LINAC electron beam was selected to provide up to 10 MeV photons capable of penetrating any shielding surrounding potential nuclear material in a measurement object. Another advantage of the 10 MeV beam is that it generates photons that are above the photoneutron generation threshold energy in most potential shielding and nuclear material[11], which ensures that neutrons are created in whatever measurement object is placed in front of the LINAC.

The neutrons from the pulsed photons then interact in different ways depending on the composition of the object. If shielding is present in the object, then neutrons will tend to thermalize or be absorbed. The thermal neutrons then might travel to nuclear material in the measurement object and induce a fission chain reaction. Neutrons are also capable of causing a fast fission in the nuclear material in a measurement object, although this is significantly less likely than a thermal neutron returning and inducing a fission. Some of the neutrons generated from the initial photon burst as well as others produced from both prompt and delayed neutrons from induced fission reactions will then escape the measurement object and potentially be detected by the detection system.

Figure 3 is a schematic of the overall experimental setup. There are a total of 12 He-3 INL designed neutron detectors in the experiment. The INL neutron detectors are called Photonuclear Neutron Detectors (PNDs) and are specifically designed to recover from the strong gamma flash from the LINAC. A more detailed report on their design is included in References 12 and 13. Six of them are stationed 50 cm to either side of the line from the LINAC to the measurement object. The numbering on the PNDs in the figure indicates their channel number in the electronics.

Figure 4 shows the inside of the PNDs in more detail. The PNDs are 33" long, and the He-3 inside the tubes is at 2 atm. The He-3 tube is surrounded by high-density polyethylene, boroflex, and cadmium. The cadmium layer ensures that only fast neutrons enter into the detector. The boroflex thermalizes and absorbs epithermal neutrons. The high-density polyethylene thermalizes the fast neutrons that entered the detector and escaped absorption by the cadmium and boroflex such that they can interact in the He-3. The PND is designed to limit the number of neutrons reaching the detector so that the He-3 tube is able to recover from the pulse quickly.

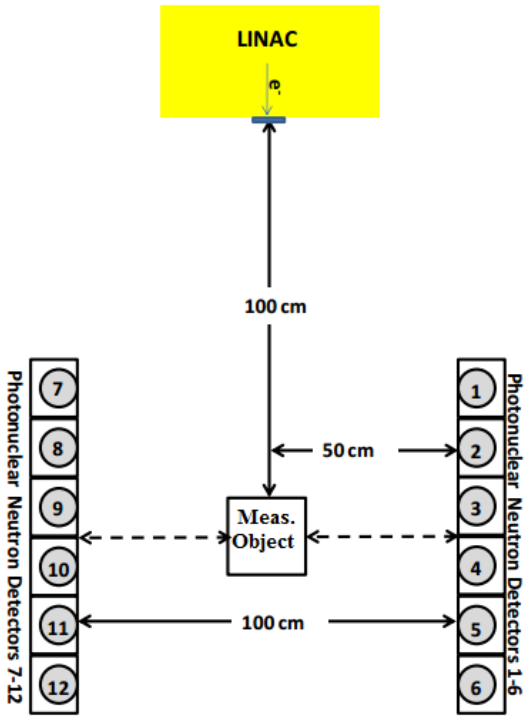

Figure 3. The experimental setup for both the US-UK measurement campaign and the June 2012 measurement campaign[7]

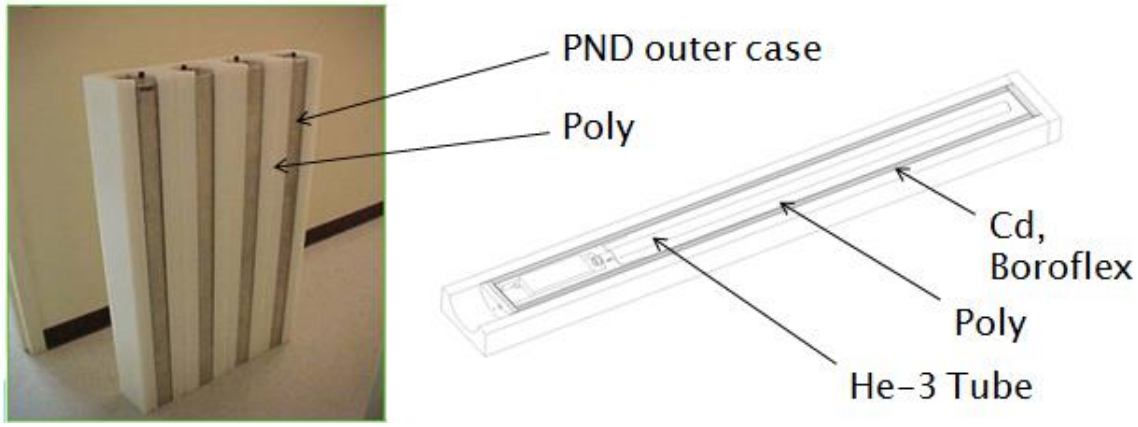

Figure 4. Exterior and interior of the PND detectors [12]

He-3 proportional counters have He-3 as a fill gas and are held at a high voltage when in operation. Neutrons entering the tubes have a chance of interacting with the He-3 in an (n,p) reaction. This reaction leads to a charge avalanche that results in a pulse. Each individual PND contains an electronic package that allows for Transistor-Transistor Logic (TTL) output. These signals are then collected by a Los Alamos National Laboratory (LANL) designed converter box which switches them from TTL to a differential signal. These signals are then fed via a ribbon cable into a LANL list-mode data acquisition module[12]. A picture of the LANL modules is included in Figure 5. In the figure, the box on the left is the LANL-designed list-mode data acquisition module, while the blue box on the right is the LANL-designed TTL-to-differential signal converter

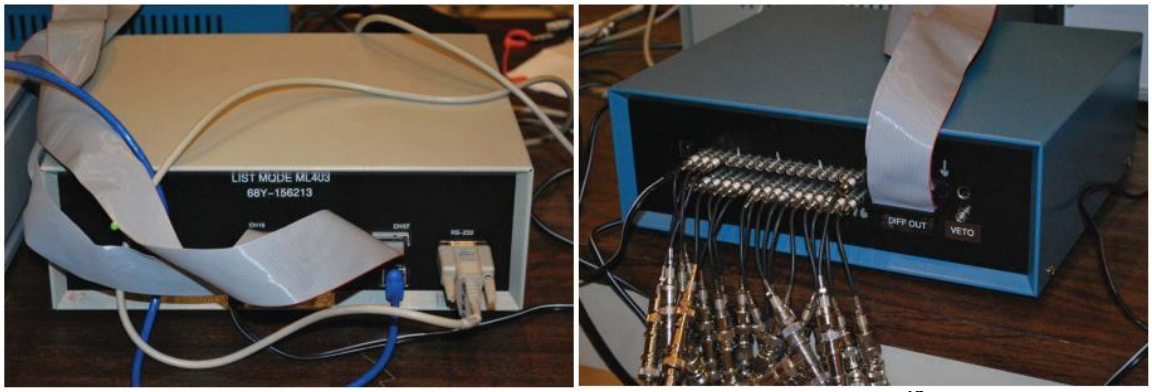

Figure 5. LANL list-mode data acquisition module[12]

List-mode data is then sent to a personal computer where it is recorded in binary files that preserve the channel of the detector and the time of the neutron detection. Preserving the information in this way allows for the original data stream to be recorded

so that any form of analysis can be performed on the measurement objects at any point after the data has been taken. In addition, the LINAC sends a pulse to the list-mode system in the last channel whenever the LINAC fires, which allows for data to be analyzed based on the time of the pulse. The list-mode data files of the measurement objects from the US-UK measurement campaign and the June 2012 measurement campaign are used for all analysis discussed in this thesis.

CHAPTER III

ANALYSIS METHODS

Three different analysis programs were employed in this project to help understand the pulsed region of the data from the measurement objects. These programs are the Pulsed Shift Register (PSR) Program, the SimPle Neutron Simulation Program (SPNS), and Monte Carlo Neutral Particle Transport (MCNP) code. All of these analysis programs utilized pulsed histograms of one form or another to help understand the measurement object data.

III.A. Pulsed Histograms

Figure 6 is a singles pulsed histogram included to demonstrate the time domains in the pulsed experiment. Figure 6 is produced by summing all of the counts in a series of time bins following each pulse. The first 1000 μs after a pulse is called the pulsed region. The neutrons in this region come from photoneutron and photofission interactions in the measurement object as well as the neutrons from fission chains induced by these initial neutrons. The neutron population in this region is dependent on time since most of the neutron creation results from fission chains due to the pulse. This means the region exhibits non-steady-state behavior, which in turn reduces the applicability of traditional coincidence analysis without special considerations being made. The work in this thesis focuses mostly on analyzing the pulsed region. The time immediately following the pulse (near t=0) has very few counts due to the gamma flash generating dead-time in the detectors. The region from 1000 to 8000 μs exhibits quasi-

steady-state behavior and is dominated by delayed neutrons produced from β-decay of fission fragments. Traditional neutron coincidence analysis techniques can be used to analyze the data in this region since the neutron background is at a steady rate and the effects of the pulse have died away.

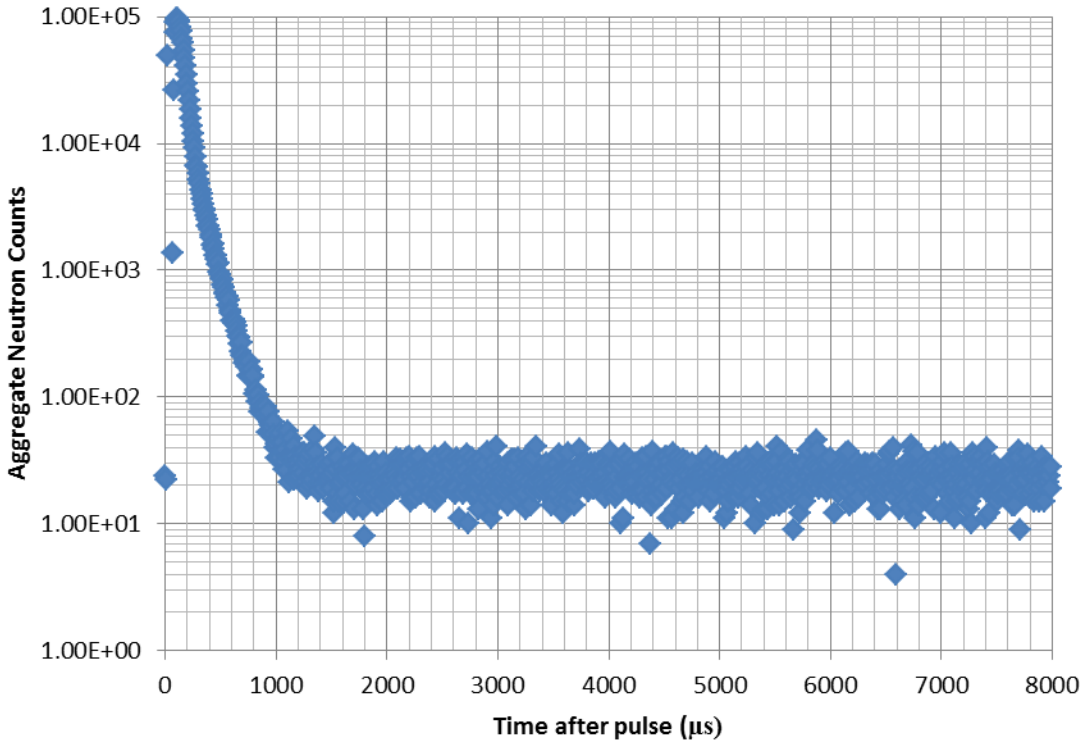

Figure 6. Log-linear singles pulsed histogram of plutonium for a 60 second total measurement time with a pulse rate of 125 Hz

The x-axis in all pulsed histograms is time after a pulse where 0 μs is the beginning of a LINAC pulse and 8000 μs is the time right before another pulse occurs. The x-axis of the pulsed histogram is broken up into bins of a set size. For most pulsed histograms in this thesis, the x-axis is broken up into 1000 bins each 8 μs wide. The y-

axis of a pulsed histogram represents singles counts (S), doubles counts (D), or D/S values per bin depending on the pulsed histogram type. Figure 6 is a singles pulsed histogram, where 'singles' refers to the total number of neutrons counted within a particular bin for the entire list mode data file. A 'doubles' count in the pulsed histogram represents the number of neutrons counted that are correlated in time to a neutron count that appears within a particular bin over the course of the entire list mode data file. The concept of a 'doubles' count is explained further in the Pulsed Shift Register Analysis section. The values in the bins of pulsed histograms represent total counts over a data file, and are not count rates. Therefore in order to accurately compare histograms from different measurement objects, the measurement time for both objects must be the same. Unless otherwise noted, pulsed histograms in this thesis are all created from 60 second measurement files.

III.B. Pulsed Shift Register

The PSR is a post-processing program coded in C++ that can accept list-mode binary files as an input and will then perform shift register operations on the data before outputting the singles and doubles rates for a file as well as pulsed histograms. A flow diagram of the PSR program is included in Figure 7.

The PSR program first looks for a user input file in the same directory location as the executable program. The user input file allows users to specify a pre-delay value, a gate-width, a long delay, a trigger offset, a window, and a veto. All of the input parameters modify how the doubles analysis and pulsed histogram analysis is performed by the PSR program. The input file also allows users to specify the names of all of the

list-mode data files they want processed using these parameters. The program will rerun for each list-mode data file included in the input file without any additional interaction from the user. This allows for the user to process a large number of list-mode files efficiently using the same parameters.

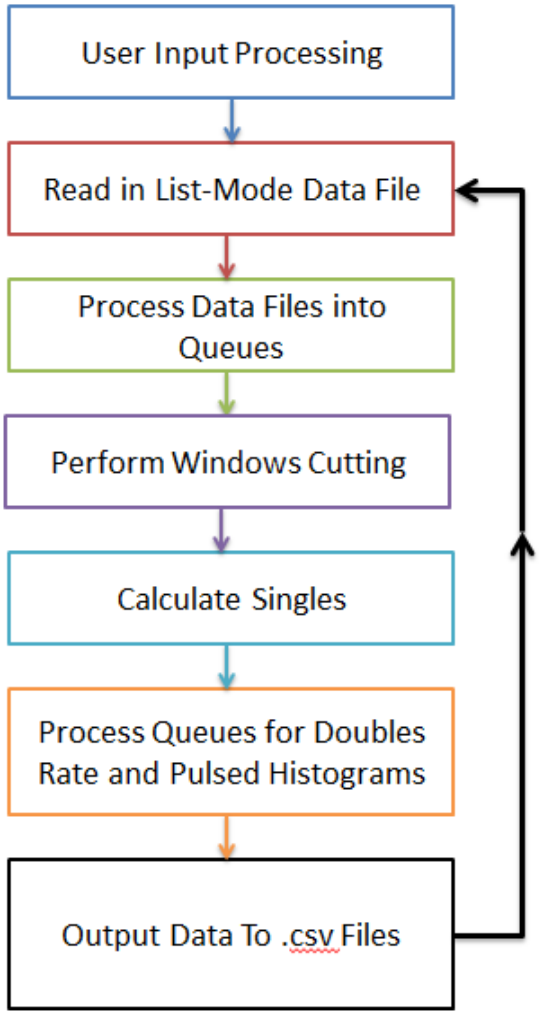

Figure 7. A flow diagram of the Pulsed Shift Register Program

After processing the input file, the program will find and read in the first binary list-mode data file. There is not a standardized list-mode data format, so the pulsed shift-register is programed to read in the LANL list-mode data format in order to couple with the LANL list-mode electronics used in the experiment. LANL list-mode electronics store channel information in 4 bytes and time information in 4 bytes for each detected neutron event, and are recorded in binary files by a computer in the order in which they are detected. There are 32 bits in the 4 bytes of channel information, and each bit corresponds to one detection channel. A nonzero bit indicates the channel or channels that detected a neutron. Time information is stored in big endian format where the first byte is the most significant and the last byte is the least significant, and in units of 100 nanosecond bins. The PNDs each have channels assigned according to Figure 3. In addition, trigger pulses, which indicate when the LINAC is pulsing, are fed to the computer for output in the same list-mode data file. Trigger pulses are assigned to channel number 31, and are all recorded with an expected time delay after a LINAC pulse in the list-mode file. This offset is one of the user inputs and is corrected for as the binary files are parsed in the PSR program.[14]

The PSR program separates the trigger pulses from the neutron pulses as it is processing the binary data files, and stores each signal type in a separate queue data structure. The queue is a first in-first out data structure, which means that elements placed in the queue first are the first elements to come out of the queue. Each element has its own location in memory and points to the next element in the queue. Queues are much more efficient to use for large amounts of data than the more frequently used

arrays because the size of the queue is only limited by the free memory of the computer. Additionally memory is allocated as the queue size increases and can be deallocated as elements are deleted, while for an array the memory required for something of that size is set aside as soon as the array is created and is not deallocated until the end of the program.

PSR also performs some corrections for electronic noise from the PND electronics during data file processing. Occasionally the PND electronics send phantom signals after an actual neutron detection event. The PSR program handles this by allowing the user to input a veto value. The veto value represents the amount of time the PSR program will ignore any neutron events in a specific channel after a detection event in that channel. Additionally, the electronics erroneously record noise on occasion across multiple channels as a pulse. Some number of these pulses could be true pulses, as it is possible for multiple neutron detectors in the system to count neutrons at the same time within the 100 nanosecond bin resolution used by the list-mode electronics. The PSR program assumes that if a pulse is detected across more than 3 neutron detectors at the same time then that data is noise rather than a true neutron signal.

After separating the list-mode data into triggers and neutron pulse queues and performing some electronic noise corrections, the PSR program performs windows subtraction. A window is a period of time after the pulse for which the program removes all the neutron data so that it will not be used in the later analysis steps. Windows subtraction is intended to correct for the neutron detector's recovery time after the gamma flash from the LINAC. This data is removed because the neutron signal during

this time is more indicative of the neutron detector's recovery from the gamma flash than the neutron signatures from the measurement object. The exact size of the window is specified by the user in the input file.

The PSR program also assumes that the time between pulses is 8000 μs and will remove any pulses that appear more than 8000 μs after the most recent trigger. This correction is applied because some measurements from the US-UK measurement campaign had regions of the data where the trigger pulse either was not recorded properly or did not send properly. This left a section of data that could not be window corrected. The program indicates how much of the file it has subtracted due to this correction as one of the final outputs.

Once the PSR program has made these two corrections, the program performs a singles calculation. 'Singles' refers to the total number of neutron counts in the list-mode data file (after the various electronics corrections) divided by the count live time. This value is determined by asking for the number of elements in the neutron pulse queue. The live time is then divided from this total count value before data output later in the program.

After storing the total counts for singles computations, the program begins shift register analysis. In steady-state analysis, shift registers are used to find the doubles, or correlated pairs of neutrons. The PSR still computes standard doubles, but also creates both singles and doubles pulsed histograms so that users can see when events happen in the time after a pulse.

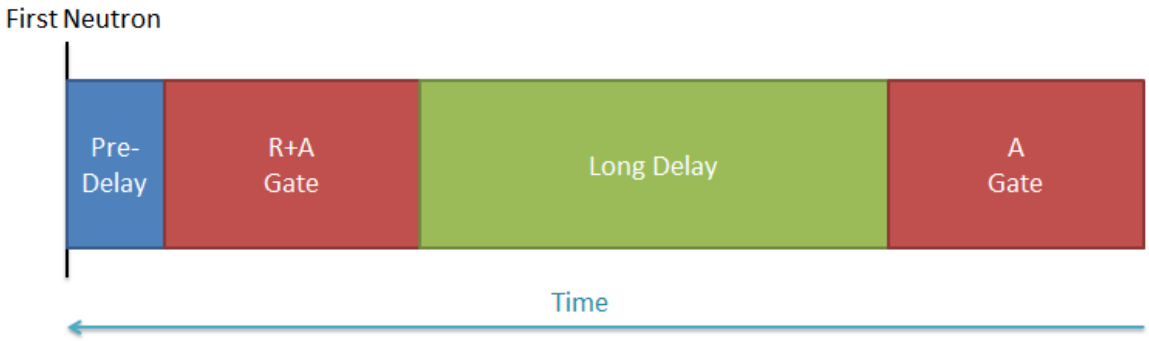

Figure 8. Time domains in a shift register

Figure 8 is included to help explain the various time domains in a shift register. Each of the time domains in Figure 8 is stored in its own queue in the PSR program. For the first shift register operation, each shift register is filled with the number of neutrons counts that span the length, in time, of that particular part of the shift register. For example, the A gate is filled first starting with the first count from the neutron queue until the time of the neutron in the neutron queue is greater than the length in time of the A gate. This next count is then placed into the long delay queue and this procedure repeats until all of the time domains are filled with the number of counts that span their length in time. This operation is performed because in normal shift register operations, each time domain will already be filled when data collection starts. The time it takes to fill these domains is subtracted from the live time in the file.[15]

After the time domains are filled, the count that is next in the neutron queue is pulled out of the queue and designated as the first neutron, as seen in Figure 8. The first neutron is the neutron count that triggers data to be recorded, and the number of neutron counts in each time domain past the first neutron is dependent on when neutrons in the data stream were detected and the length of the time domains. When triggered, the shift

register ignores any neutron counts that appears in the pre-delay time domain to avoid a dead-time effect in the electronics, and the number of neutron counts in the R+A gate queue and the A gate queue are counted. By counting the number of neutron counts in the R+A gate and the A gate, the shift register is tracking the pairs of two that can be made with the first neutron in each gate. The shift register then places the current first neutron in the pre-delay queue, and pulls the next neutron count from the neutron queue to be the new first neutron. The neutron counts in the time domains queues are then adjusted based on the detection time of the new first neutron and the length of the time domains specified by the user. This analysis is repeated until the neutron queue is empty.[15]

In traditional steady-state coincidence analysis, the operation with the R+A and A gates will result in doubles values that correlate to the neutrons produced from fission in a sample. A double is more formally defined as time correlated pairs of neutrons from the same event, typically a fission chain reaction. The R+A minus A gate subtraction is able to determine correlated neutron events based on the typical behavior of fission chains. This behavior is characterized by the Rossi-Alpha distribution, as seen in Figure 9. The Rossi-Alpha distribution shows that neutrons generated due to a fission chain are most likely to be detected close in time, relative to the neutron lifetime in a sample, to an initiating neutron event rather than further away. Based upon this observation, if a gate were opened near to a neutron event and then again several neutron lifetimes away from that neutron event, the possible pairs of neutrons in the first gate would be statistically larger than the possible pairs of neutrons in the second gate if the first or initiating

neutron had a time correlation with some of the detection events in the R+A gate. Therefore if the A gate were subtracted from the R+A gate, it would only leave the "Reals" or true Doubles from fission chains rather than "Accidentals" or incidental coincidences from other types of neutron generation with no time-dependence. As this operation is performed on all neutrons for the duration of the count time, the resulting doubles value is then correlated to the neutrons produced via fission in the measurement object.[15]

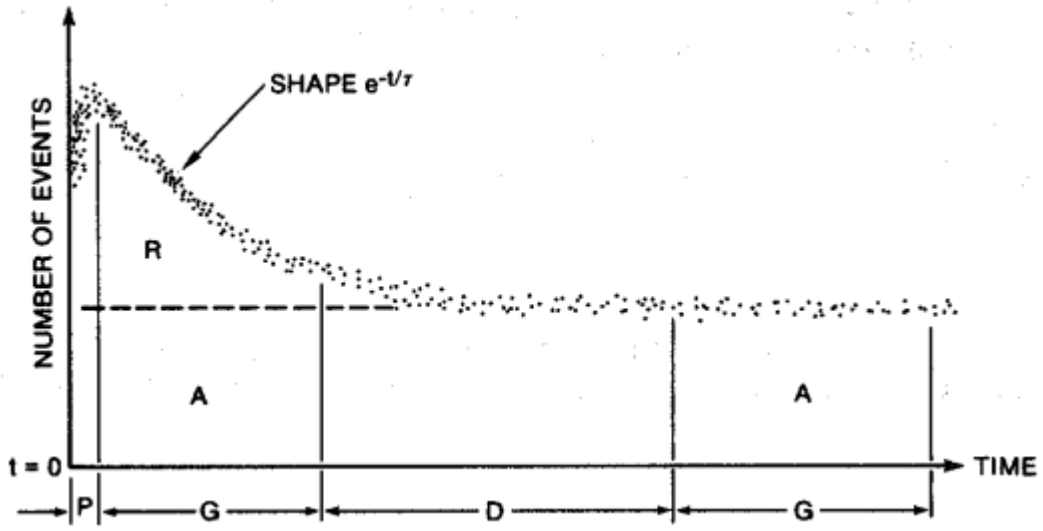

Figure 9. A Rossi-Alpha distribution of neutron events after an induced fission followed by a fission chain over time[16]

This, however, does not work as simply in a pulsed environment because the shape of the Rossi-Alpha distribution is unknown whenever the situation is not steady-state. The main issue in determining a doubles rate in a pulsed environment is that the pulse introduces time dependence to the A gate which is not explicitly known. Since the

shape of the accidentals curve is unknown it becomes very difficult to subtract a proper number of accidentals from the R+A gate in order to determine the true correlated pairs of two neutrons from the measurement object. The way this issue is handled with PSR analysis is to make the length of the long delay such that the A gate opens on the same part of the pulse profile as the R+A gate[17]. This ensures that the singles profile should be around the same in both gates throughout the experiment, which in turn should account for the time dependence in the accidentals and allow for true coincidences to be found in the data.

While performing the shift register analysis, the PSR also populates the singles and doubles pulsed histograms. In the internal workings of the program, the doubles pulsed histogram is tracked separately as an R+A histogram and an A histogram to avoid subtraction error. The program populates pulsed histograms using the first neutrons during the shift register analysis. For both singles, R+A, and A pulsed histograms the length of time between the current first neutron and the most recent pulse is used to determine the appropriate bin in which to place information. For singles pulsed histograms, the bin selected is then incremented by one to indicate that a neutron was present in that time after a pulse. For R+A and A pulsed histograms, the selected bin is incremented by the R+A and A values computed for the current first neutron.

Once the program completes the analysis steps, it outputs the information in two separate .csv files. Comma separated value files were chosen as the output method to facilitate graphing within Microsoft Excel. One .csv file contains the traditional shift

register output, including the singles and doubles values, while the second .csv file contains the pulsed histogram information.

III.C. Simple Neutron Simulation

The SimPle Neutron Simulation (SPNS) program is two neutron energy-group point model Monte Carlo simulation that was written in C++ to model the physics of interactions within a measurement object. This program was first presented in Reference 18, and a flow diagram for the program is shown in Figure 10.

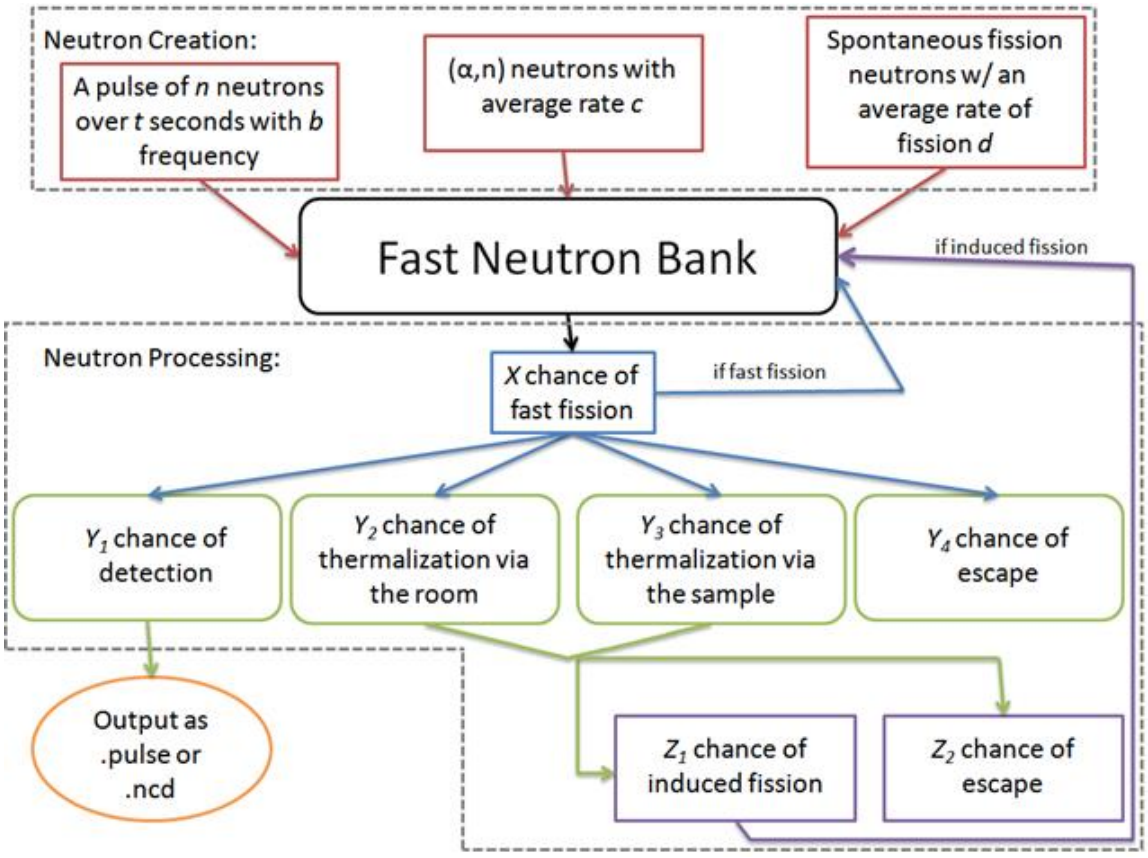

Figure 10. Flow diagram of the SPNS Monte Carlo Program

The first SPNS program operation is to read in input parameters from an input file. The user specified inputs determine how neutrons will interact during the simulation as well as what is output to the user. A user can control both the type and the time length of an output file, as well as the pulsed and steady state neutron creation characteristics such as the length of a pulse, numbers of neutrons in a pulse, and the frequency of the steady state creation events. The user may also adjust how created neutrons interact with a measurement object and are detected in the detection system by modifying the probabilities of detection, thermalization, induced fission, fast fission, and escape as well as the die-away time of thermalization and detection and the channel dead-time value. The user can also specify the average number of neutrons that come from induced, spontaneous, and fast fission in order to represent the material type of a measurement object accurately. This is implemented by using a multiplicity distribution for each fission type. Presently the multiplicity distributions in the input file are set to represent a U-235 sample based on data from the PANDA manual[16]

After reading in input parameters, SPNS creates neutron data files through a multistep process depicted in Figure 10. Neutron creation is the first step in this process. Neutrons can be created either as a group of neutrons in a pulse of a certain width, at a certain rate through a typical uncorrelated 'singles' event such as an (α,n) process, or through spontaneous fission at a certain rate. For the data modeling in this thesis, the uncorrelated 'singles' events and the neutron pulses were the primary methods of neutron creation. Each neutron that is created is then added to a fast neutron bank. Neutrons created by a spontaneous fission event share the same creation time, and

neutrons created as part of a pulse are tagged with times that are randomly distributed over the pulse width.

Once neutrons have been created, they can be removed individually from the fast neutron bank and processed to determine on average how they would interact based on the user input parameters. Upon removal, SPNS rolls on the chance of the neutron undergoing fast fission. If the neutron fast fissions, the resulting neutrons are placed back in the fast neutron bank. The neutrons from fast fission share the same timestamp as the inducing neutron. If a neutron fails to fast fission then it moves into the next loop of interaction chances. In this loop, a neutron has a chance of being detected by the detection system, escaping the sample, or thermalizing and having a chance of inducing fission. If a neutron thermalizes then its thermal diffusion time is simulated by using a stable Poisson distribution:

$$t(a,b) = a * -\log(1-b) \tag{1}$$

In this equation, a is the user specified die-away time and b is a random number from 0 to 1 generated by the random number generator. The stable Poisson distribution is also used to model the detector die-away time when a neutron is detected. In the case of both thermalization and detection, the die-away time is added to the already existing neutron time to simulate the neutron being thermalized.

After being detected, neutrons are passed to a loop to assign a channel number to the neutron. In this thesis work the channel corresponds to the helium-3 detector that

detected the neutron. Dead-time is assumed to be non-paralyzable in SPNS and is tracked on a per channel basis, so once a neutron is detected in a channel, the channel will be unable to detect other neutrons for a fixed period of time. After channel assignment, neutrons are sorted according to detection time before being output in the user specified output format.

The random number generator used for this Monte Carlo program comes from "Numerical Recipes in C" [19]. It is a random number generator of L'Ecuyer with a Bays-Durham shuffle and added safeguards. This allows for a long period of greater than 2×10^{18}. This recursion relationship for a linear congruential generator is used in the code:

$$X_{n+1} = (A\,X_n + C) mod (M) \tag{2}$$

In this equation, X_0 is the seed value, A is known as the multiplier, C is the increment, and M is the modulus. The SPNS random number generator is a variation of a linear congruential generator. The program adds two different random number sequences with different periods, which results in a new sequence whose period is the least common multiple of the two periods. After adding the two sequences, the modulus of the modulus of either of the two series is taken. The combination of two different random number sequences results in a very low incidence of serial correlation in the resulting sequence. To further reduce the chance of serial correlations, an additional shuffle is added to the new random number sequence. All of these processes result in a random number

generator that passes all statistical checks for randomness, which means that the results generated from SPNS are not affected by poor statistical randomness.

III.D. Monte Carlo Neutral Particle Transport

The Monte Carlo Neutral Particle Transport (MCNP) code is a FORTRAN-based radiation transport code that is maintained by LANL[20]. MCNP allows for users to establish a geometry that is representative of their experiment and select the types of initiating particles to create. The code then uses nuclear data cross sections and random sampling to determine on average the history of all particles in that particular geometry. By history, we mean the location, energy, time, and direction of each particle simulated. The user then can ask for different kinds of information from the program using tallies that track how many particles passed through a location in a certain energy or time bin as well as the expected number of interactions in a certain location in a certain energy or time bin.

Since MCNP uses well validated continuous-energy cross-sections, it can provide a very accurate model of the physics of most kinds of experiments including the pulsed photonuclear experiment. The precision of MCNP is due to the number of histories sampled. To build statistically significant results, a large number of particle histories must be sampled. For a pulsed histogram simulation, events that occur in the region past the window subtraction are much lower probability events, so a large number of histories need to be run in order to get statistically relevant answers for this time domain. MCNP was also configured so that all of the histories were started at time 0 rather than as a series of pulses. This means that the delayed neutrons are not modeled

accurately since they never reach an equilibrium state comparable to what is seen in experimental data. MCNP also fails to account for detector electronics effects, and reports answers assuming perfect electronics. These limitations and features restricted the possible relevant uses of MCNP in this thesis work.

CHAPTER IV

PERTURBATION STUDIES AND WINDOW SELECTION

SPNS was used to generate data with known perturbations in order to understand how various changes in pulsed histogram graphs related to physical quantities. The SPNS generated list-mode files were analyzed with the PSR to generate singles and doubles pulsed histograms. These data sets were plotted using Microsoft Excel.

Unless specifically perturbed, several SPNS parameters were fixed for all of the perturbation studies. These parameters included the count time at 60 s, the pulse length at 4 μs, the pulse frequency at 125 Hz, the detector efficiency at 10%, the detector die-away time at 37 μs, the trigger offset at 70 μs, and the detector dead-time at 2.6 μs. In addition, and the channel detection probabilities were set to give each channel an equal chance of seeing a neutron. The induced fission multiplicity diagram was always set to represent Highly Enriched Uranium (HEU) unless specifically perturbed. Also, throughout the perturbation study the lines between data points do not represent a fit of the experimental data and are intended only as visual aids to help discern one specific set of data from another.

The PSR parameters used to analyze this data are the same as the values used to analyze experimental data in order to make the perturbation analysis more comparable to the experimental data analysis. The values used were 3 μs for pre-delay, 96 μs for gate-width, 8000 μs for long delay, 70 μs for the trigger offset, and 2 μs for the veto.

The first set of perturbations studied the effect of changing the number of neutrons generated per LINAC pulse. This can be caused by changes in the amount of nuclear material or changes in the beam current of the LINAC. Figures 11 and 12 are singles and doubles histograms that depict the effects of these perturbations. The curves lengthen in time in both figures as the number of neutrons per pulse is increased due to dead-time in the He-3 tubes. In addition, the curves do not disappear immediately after the detector pulse due to the detector die-away time.

As the neutrons per pulse are increased from 5k to 10k and then again to 20k in Figure 12, the maximum doubles value in the bins increases and then decreases dramatically with 20k. This effect is again due to the dead-time in the He-3 tubes, and is not seen in Figure 11 because singles detection only relies on one detector. In doubles detection, more than one detection event is needed to successfully detect the pair of correlated neutrons. This means that after reaching some critical value of doubles neutrons, increasing the number of neutrons will lead to a decrease in doubles because it is more likely that one of a correlated pair of neutrons will be lost due to channel dead-time. The slightly negative doubles values after the positive peak in Figure 12 are likely due to the length of the long delay only being approximately equal to the time between pulses, which results in a slight time lag between the max A gate bin and the max R+A gate bin. The appearance of a “knee” in Figure 12 for the 20k and 50k graphs is due to the same discrepancy between the max A and max R+A gate bins.

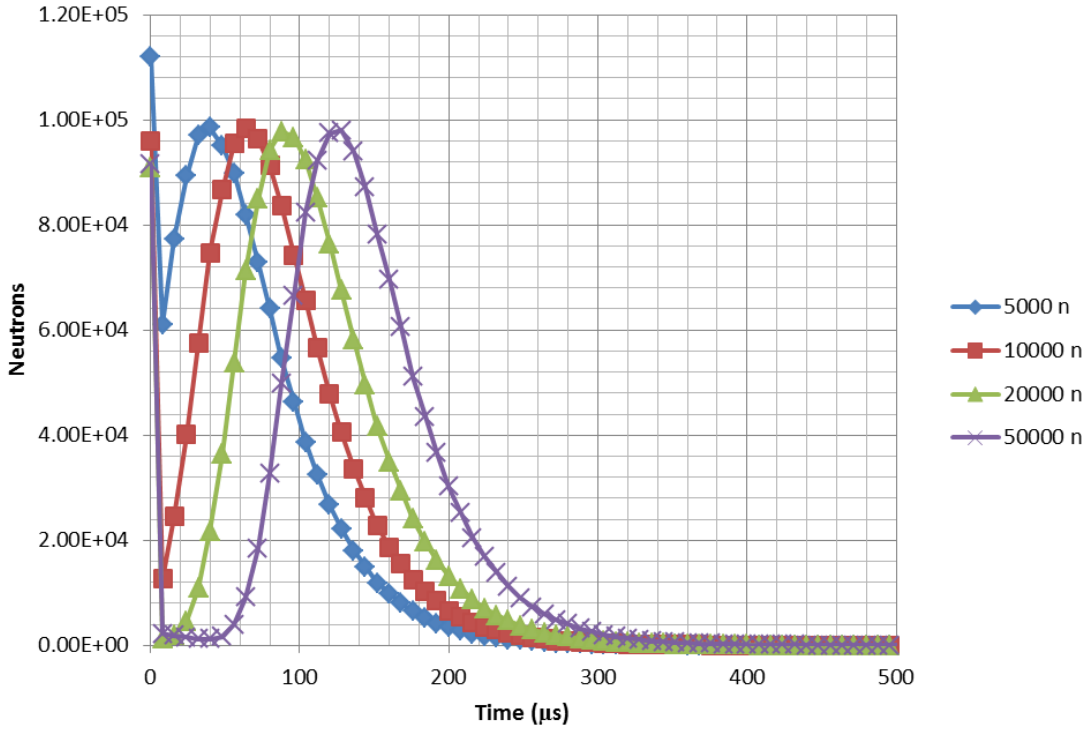

Figure 11. Singles pulsed histogram calculated by SPNS varying the number of neutrons per LINAC pulse

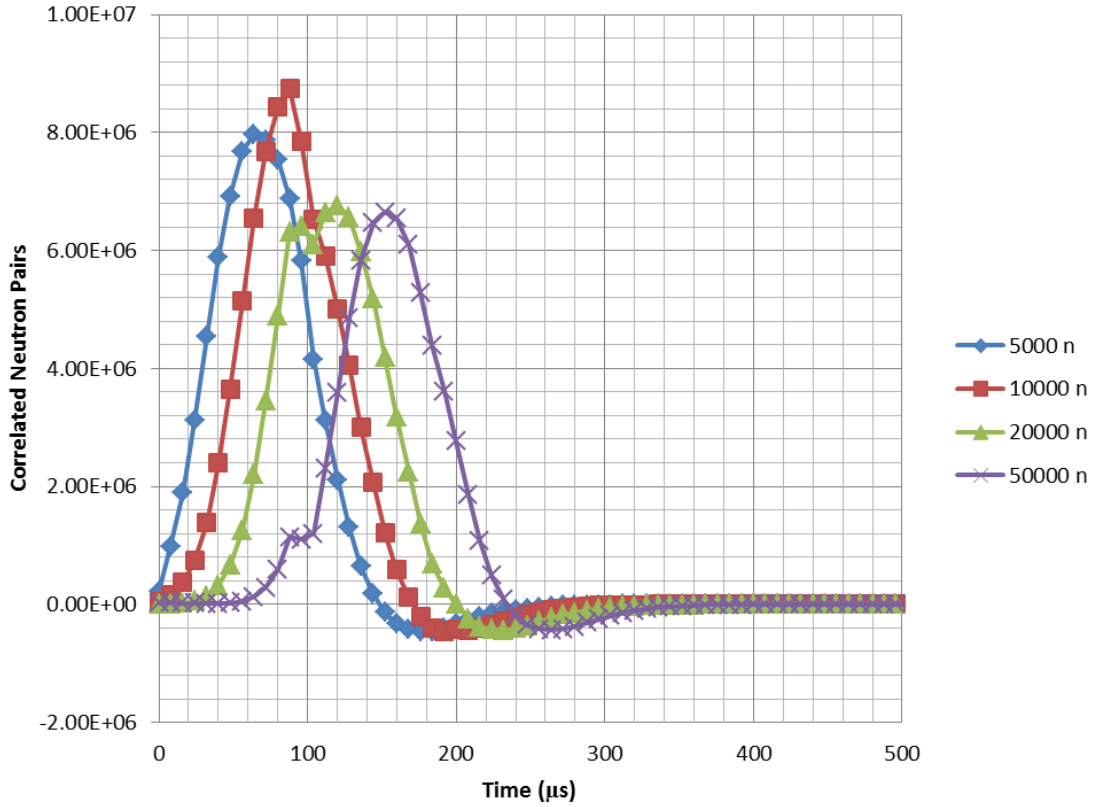

Figure 12. Doubles pulsed histogram calculated by SPNS varying the number of neutrons per LINAC pulse

The second set of perturbations focused on PND detector effects including the detector die-away time in Figures 13 and 14, and the channel dead time in Figures 15 and 16. The number of neutrons per pulse for these perturbations was held constant at 20k. Figure 13 demonstrates that the detector die-away time also plays a role in lengthening the time that it takes before the neutron curve from the neutron pulse returns to zero since as detector die-away increases neutrons are more spread out in time. Figure 15 shows that as channel dead time increases, the singles count rate is depressed. Channel dead time does not impact the time length of the curve.

The effect of channel dead time on doubles is again demonstrated in Figures 14 and 16. In Figure 14, the significant increase in the magnitude of the doubles counts is due to the fact that increasing the detector die-away time has spread the neutrons arrival time at the helium-3 tubes out more, so less of the doubles pairs are lost to channel dead time. In 16, as the channel dead time increases, it reduces the doubles counts at a faster rate than it reduces the singles counts in Figure 15. The rate is depressed so significantly that by 10.4 μs the curve is virtually flat. The "knee" that appears in some of the cases in Figures 14 and16 is likely due to the same effect that was illustrated in Figure 12, and the reappearance of the negative doubles continues to be due to the length of the long delay not be precisely equal to the time between two pulses.

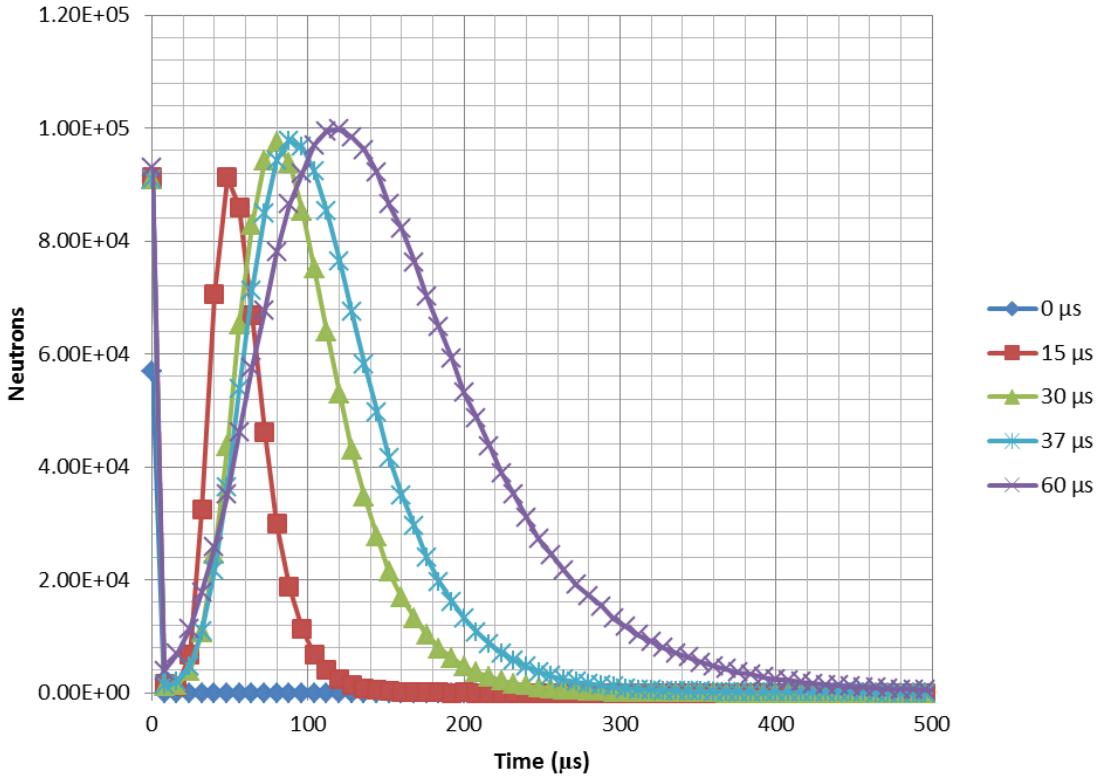

Figure 13. Singles pulsed histogram calculated by SPNS varying the detector die-away time

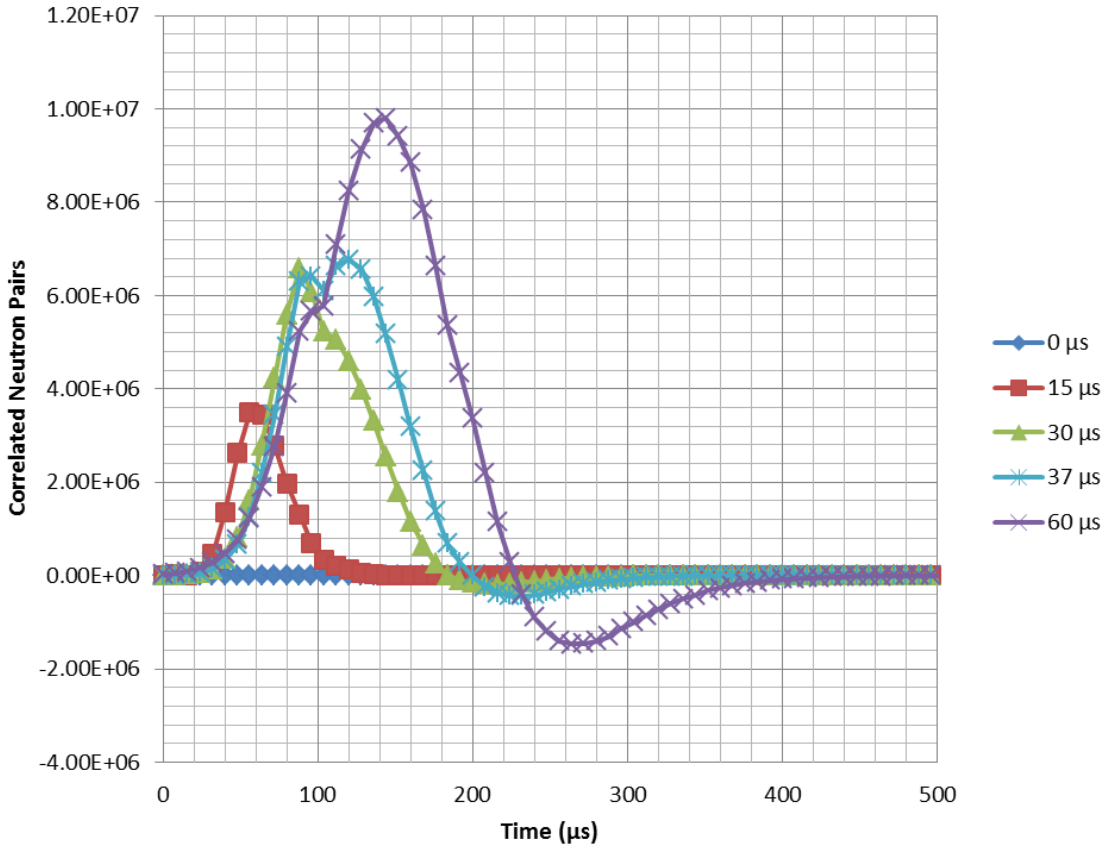

Figure 14. Doubles pulsed histogram calculated by SPNS varying the detector die-away time

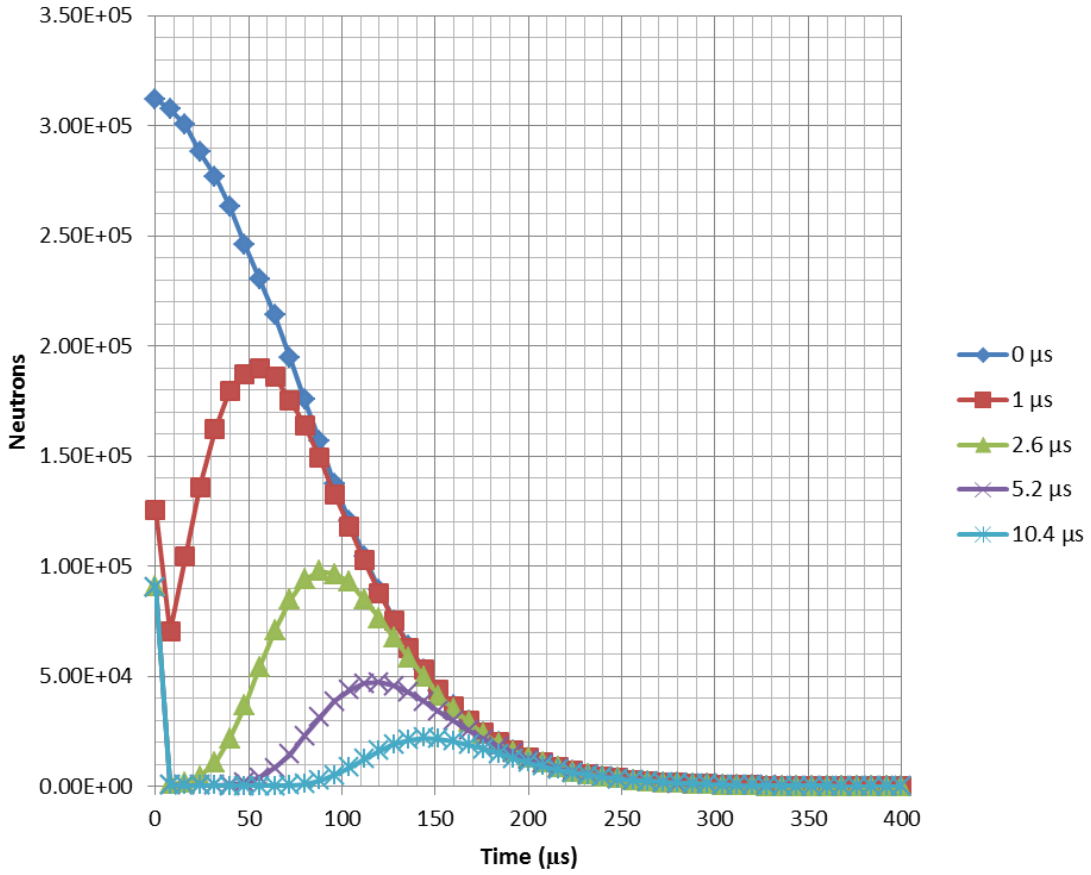

Figure 15: Singles pulsed histogram calculated by SPNS varying the channel dead-time

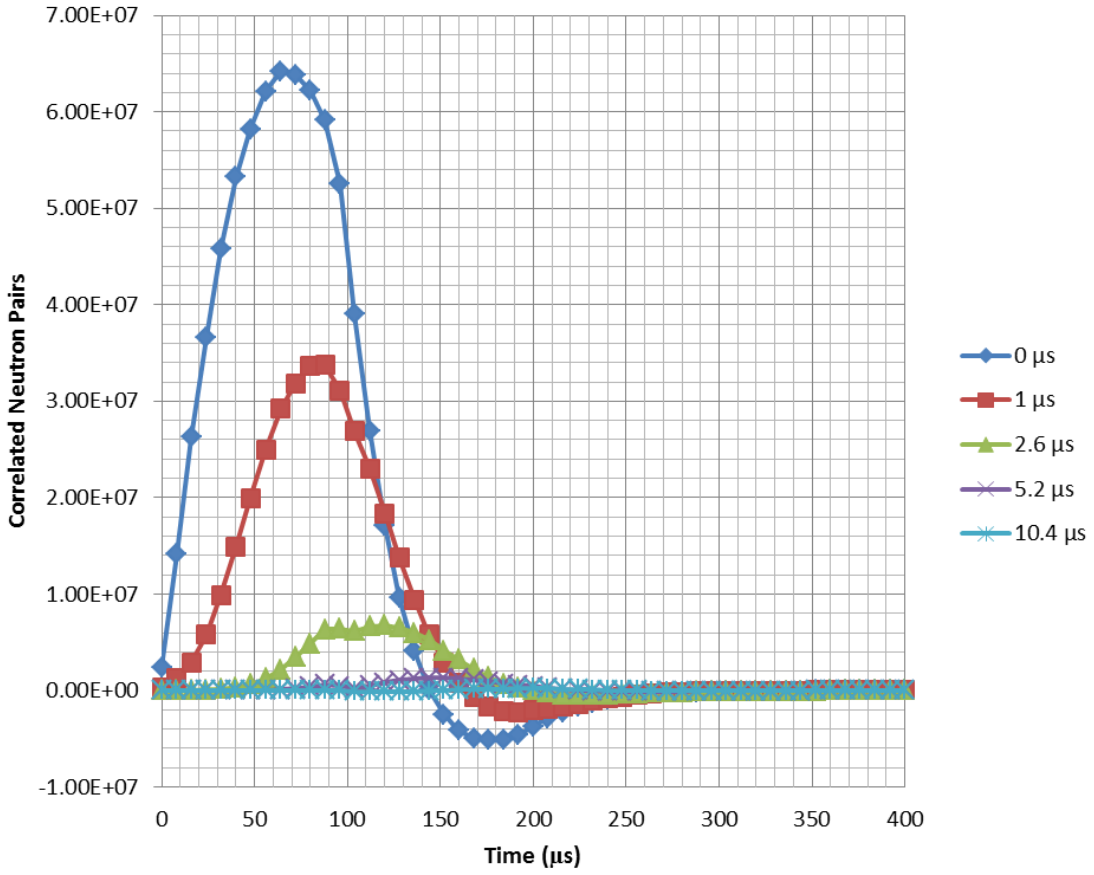

Figure 16. Doubles pulsed histogram calculated by SPNS varying the channel dead-time

Ideally only changes in the measurement object would result in changes in the appearance of the data in the pulsed histograms. Of the SPNS parameters studied so far, only the pulse height could change due to the composition of the measurement object, although even this parameter can be strongly influenced by the detector die-away and the channel dead-times, which are solely effects from the design of the PND. The detector die-away will be constant throughout the experiment since it is an inherent part of the detector design and so it can be corrected for. However, experimental data indicates that the performance of the PND electronics immediately after the pulse does not conform to

the non-paralyzable dead-time assumption made in the SPNS model. In fact both charge deposition and dead-time behavior in the PND electronics immediately after the pulse are not well characterized due to the gamma flash. For this reason, the data in this region needs to be cut out of the analysis in order to get more accurate results. This is performed by using a window in the PSR analysis tool.

An ideal window in the analysis would be located at a point where the dominate effects determining the appearance of the pulsed histograms were from the measurement object rather than the PND detector. MCNP was determined to be the ideal tool to ascertain this point because it has the most accurate physics modeling of any of the analysis tools available for this work and it fails to account for electronics effects in its tallies. As a result, an ideal window could be determined by comparing data generated from an experimental measurement to a representative simulation of that experiment in MCNP.

For the MCNP simulation of this experiment, the quasi-steady-state delayed neutron background was not modeled. Without delayed neutrons, getting enough histories to generate statistically useful data in time domains that were far away from the pulse becomes difficult for most of the measurement objects. One solution to this problem is to choose a measurement object that contains both moderation and fissile material, so HEU moderated with polyethylene selected as the experimental data to model for this study.

Two HEU plates of 5.1x10.2x.27cm were used as material in the measurement object. Each place contained 272 g of uranium at an enrichment of ~93% U-235. These

plates were surrounded by a 1” polyethylene box, and then had an additional 2” of polyethylene on all sides in the plane of the detector. Figure 17 shows the MCNP geometry of the measurement object as well as the detectors. The image is taken looking down on the experimental apparatus so that it is comparable to the experimental setup diagram in Figure 3.

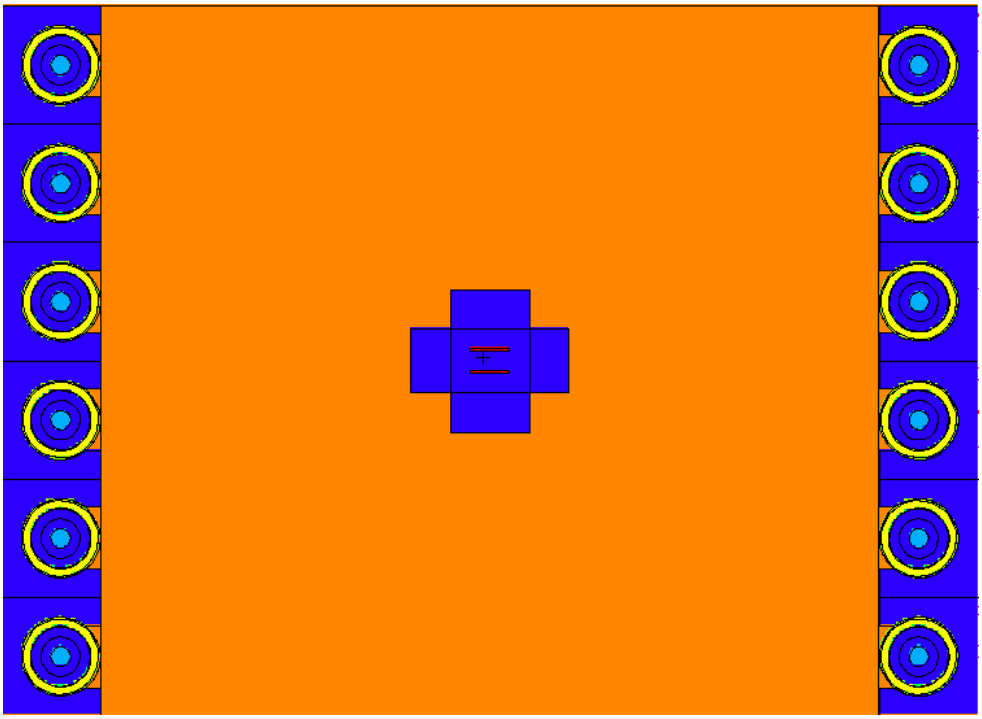

Figure 17. MCNP geometry of the PND detectors and the 544g polyethylene moderated HEU sample

Spontaneous fission neutrons were started in the HEU over the course of 4 μs as the initial source in the MCNP deck. A spontaneous fission neutron tends to be lower in energy than a neutron from a photofission, and a more representative neutron energy spectrum was not included due to a lack of published information on the flux of photons out of the LINAC. In order to generate an accurate neutron energy spectrum, either the

energy flux of photons from the LINAC needs to be known, or the energy spectrum of the electrons and the shape of the tungsten target as well as the collimator must be known. However, no accurate information was available for the electron or photon flux. The resulting error from this approximation should over-predict the reactivity of the sample slightly.

The source strength normalized flux of the neutrons reaching the detectors was determined by using a combined F4 tally for all of the He-3 detectors. A FM card was then used to multiply the F4 tally by the (n,p) interaction cross-section of He-3 to convert the source strength normalized flux into a source strength normalized count rate. The tally was then split into 400 time bins each with a width of 20 μs to try and replicate the pulsed histogram data format.[20]

The resulting data points were then imported into Microsoft Excel to plot on top of a singles pulsed histogram of a one minute experimental data run of the polyethylene moderated HEU sample. The resulting plot can be seen in Figure 18. Another difficulty with the MCNP simulation is that the source strength of the LINAC in terms of neutrons generated in the target is unknown since the variables needed to simulate the electron conversion to Bremsstrahlung photons in the tungsten were never found in literature. To overcome this obstacle, the MCNP points were fit to the experimental data using a constant to transform the MCNP data points along the y-axis. This constant was adjusted by eye until there was maximum agreement between the MCNP and experimental points. This method of fitting the MCNP data to experimental data generated a window cutoff value of 260 μs, which is denoted by a green line in Figure 18.

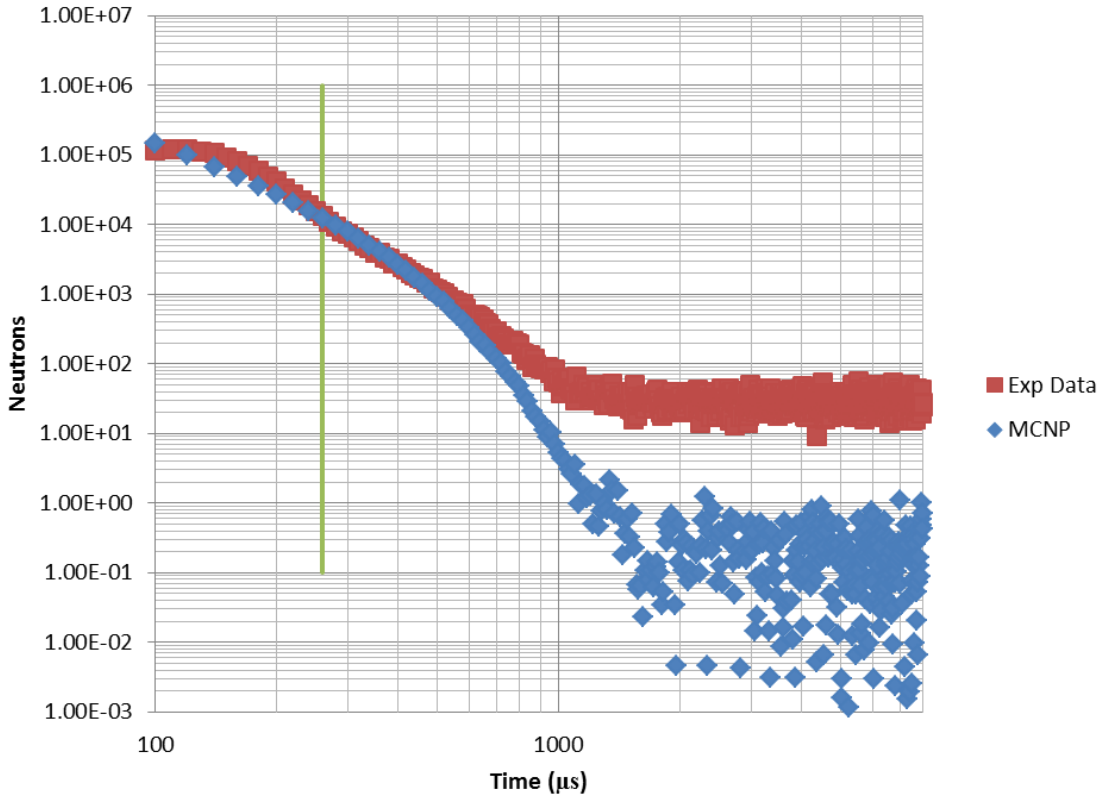

Figure 18. Log-log graph of MCNP data plotted on experimental HEU data with the window value denoted with a green line

The method of window determination is not very precise; however, the window subtraction value does not need to be very precise in order to generate meaningful results. While the best results would be generated with an optimal window, a window that removes most of the effects from the uncharacterized detector behavior will reduce the impact of these effects enough that data analysis results will be indicative of measurement object properties. Two methods were employed to determine whether or not the window was adequate to generate meaningful results. First, the initial perturbation studies for the detector die-away time and the neutrons per pulse were reexamined to ensure that the detectors time dependent effects are minimized. The

second method involved examining measurement object specific effects to ensure that they were predominant when the 260 μs window was used.

The detector dependent effects were reexamined in Figures 19, 20, 21, and 22. Figures 19 and 20 displays the pulsed histogram results from the perturbation of the neutrons per pulse parameter and Figures 21 and 22 shows the resulting pulsed histograms from the perturbation of the detector die-away. Channel dead-time is not reanalyzed because while its effect on the data is more dramatic in regions with higher count rates, it does not have a dominant effect on any particular time region. Figures 19, 20, 21, and 22 demonstrate that a window of 260 μs significantly reduces the magnitude of time dependent detector effects on the analysis in the pulsed region.

The perturbation methodology employed for this study using the SPNS model ensured that the last perturbation of a particular parameter extended past the range of values typically seen when SPNS was used to fit experimental data. With this in mind, the time dependent effects of the detector die-away time should not have a significant impact on measurement object effects with the 260 μs window. The more significant effects to escape elimination are seen in Figures 19 and 20 and result from neutron per pulse perturbations. Since measurement object characteristics also impact the number of neutrons generated per pulse in the SPNS model, the failure of the window to completely eliminate effects based on changing neutrons per pulse values is not a major concern.

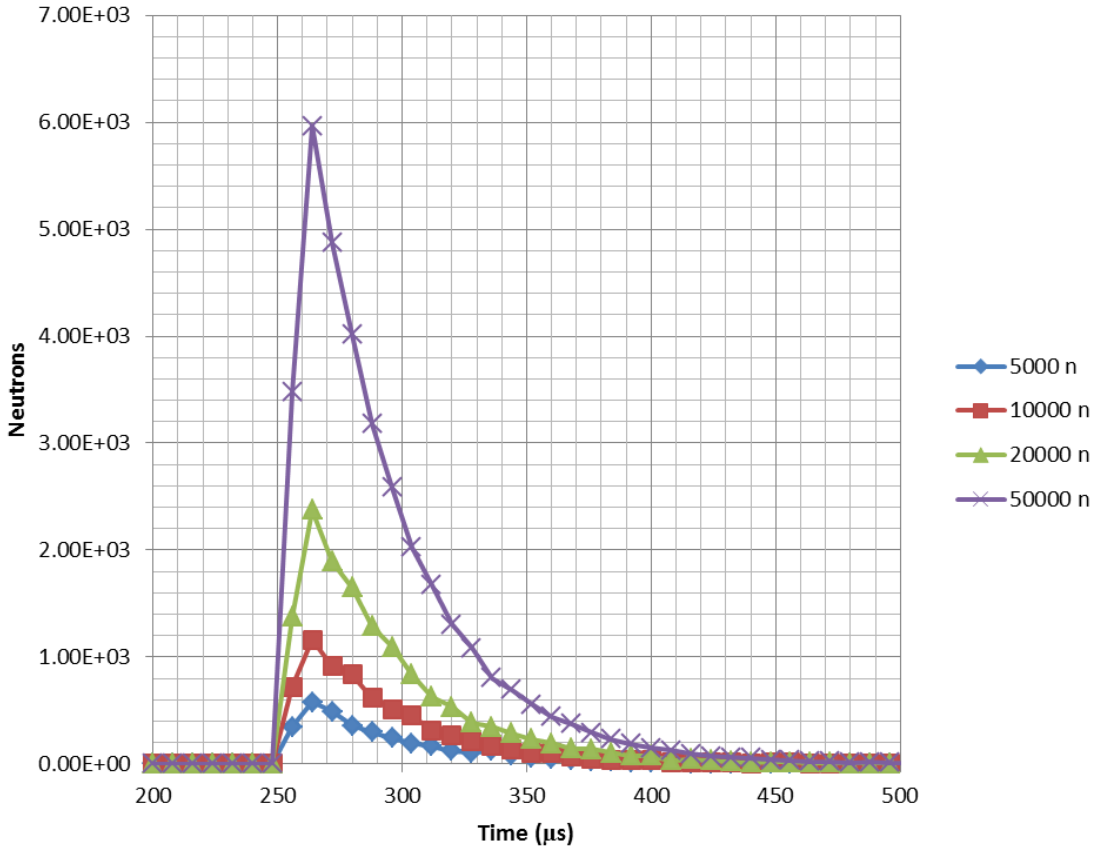

Figure 19. Singles pulsed histogram calculated by SPNS varying the neutrons per pulse with windows

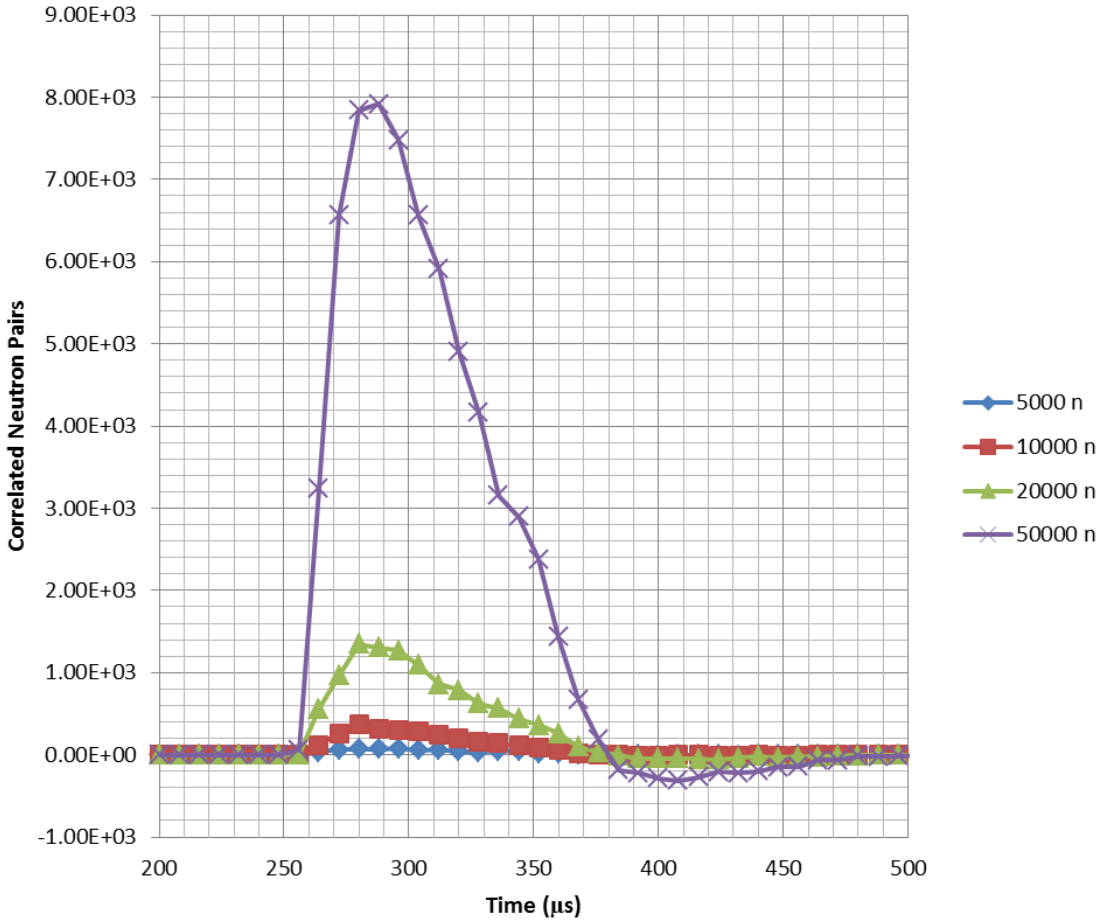

Figure 20. Doubles pulsed histogram calculated by SPNS varying the neutrons per pulse with windows

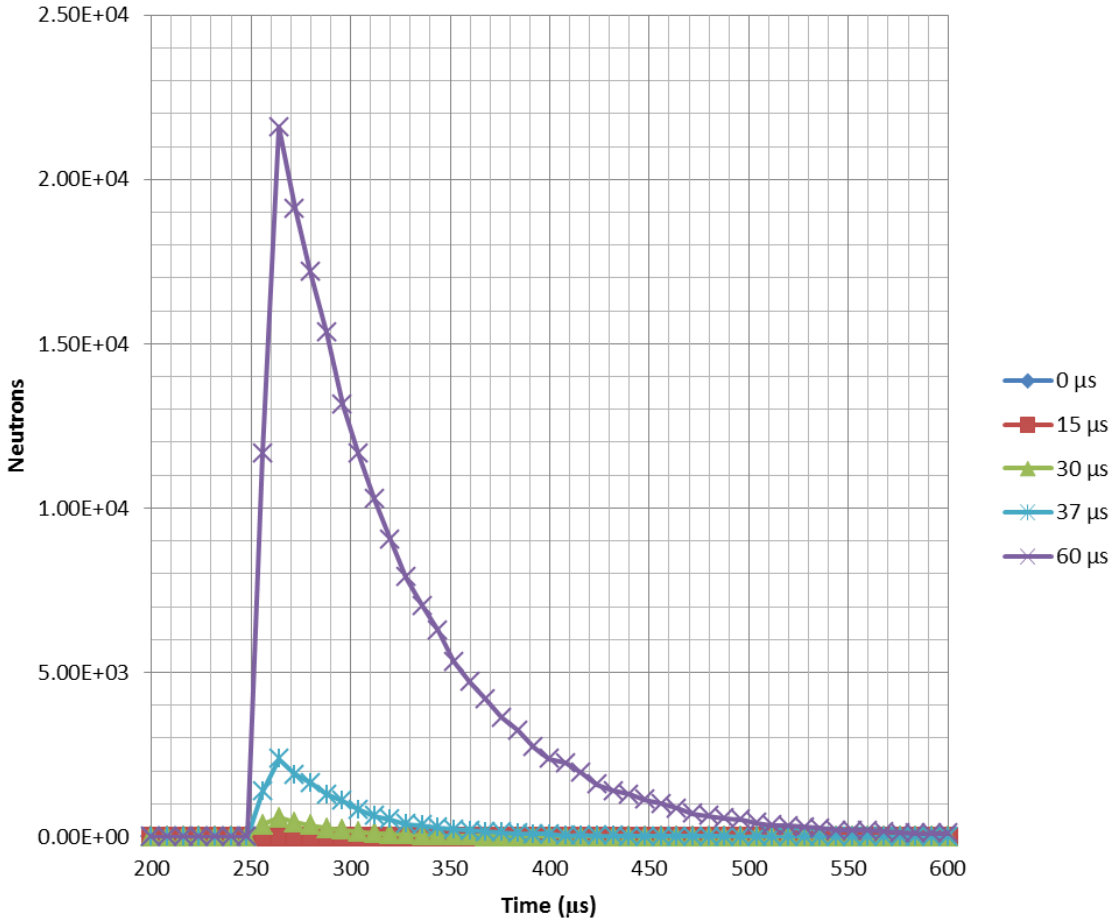

Figure 21. Singles pulsed histogram calculated by SPNS varying the detector die-away with windows

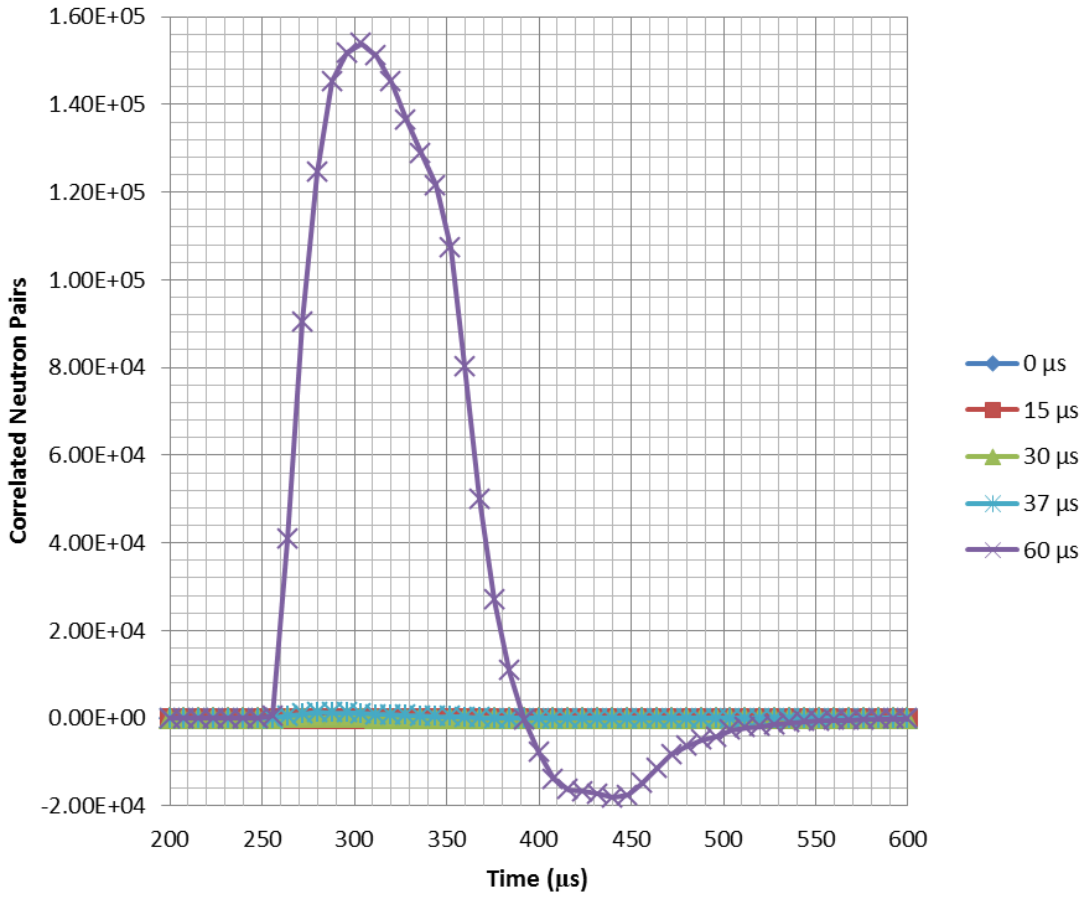

Figure 22. Doubles pulsed histogram calculated by SPNS varying the detector die-away with windows

SPNS model parameters that are impacted by characteristics of the measurement object include the probability of a neutron thermalizing and then undergoing fission and the associated die-away time of this thermalization as well as the induced fission multiplicity distribution. The neutrons generated per pulse were held constant at 20k for the perturbations of these parameters. In addition, whenever not being perturbed the probability of fission was set to 1% and the die-away time was set to 120 µs. The resulting pulsed histograms from these perturbations are Figures 23 and 24 for the

probability of fission, Figures 25 and 26 for the fission die-away time, and Figures 27 and 28 for the induced fission multiplicity distribution.

From these figures, it is clear that measurement object effects are dominant when the 260 μs window is employed, as a trend from the perturbations is evident in all of the graphs. In the singles pulsed histograms from the measurement object perturbations, the figures have a single dominate exponential that is generally translated up as the magnitude of the perturbation effect is increased. These exponentials also last longer in time as they are perturbed. In the doubles pulsed histograms, the increasing magnitude of parameter perturbation generally results in translating the curve up the y-axis.

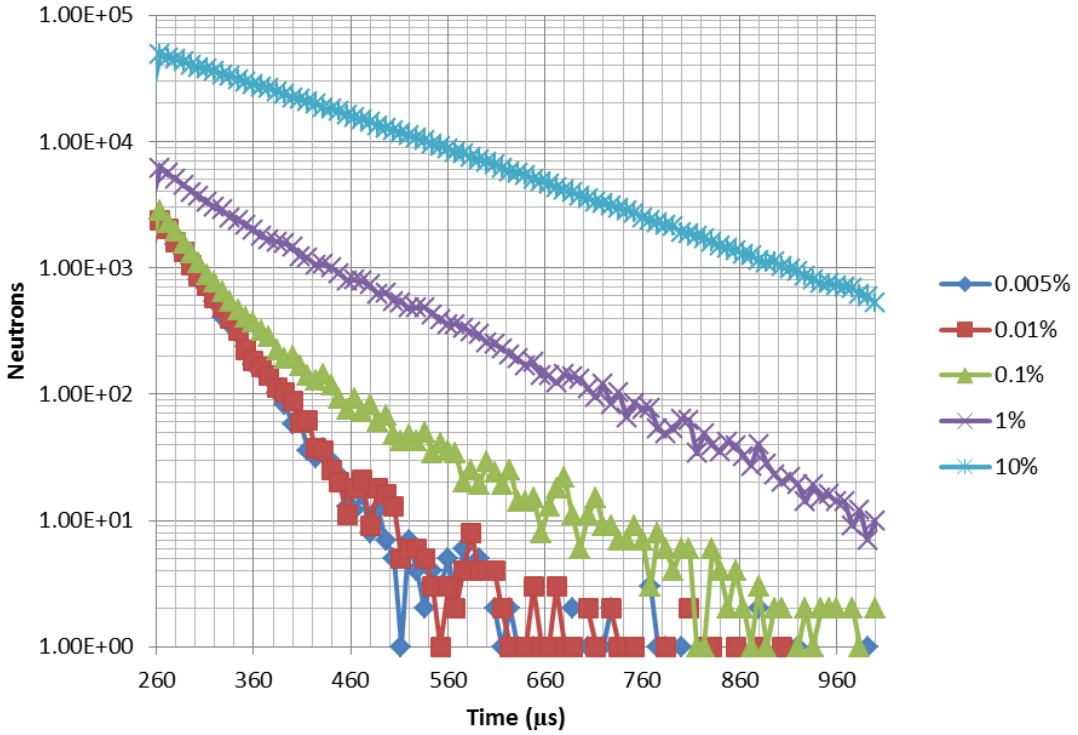

Figure 23. Singles pulsed histogram calculated by SPNS varying the probability of fission with windows

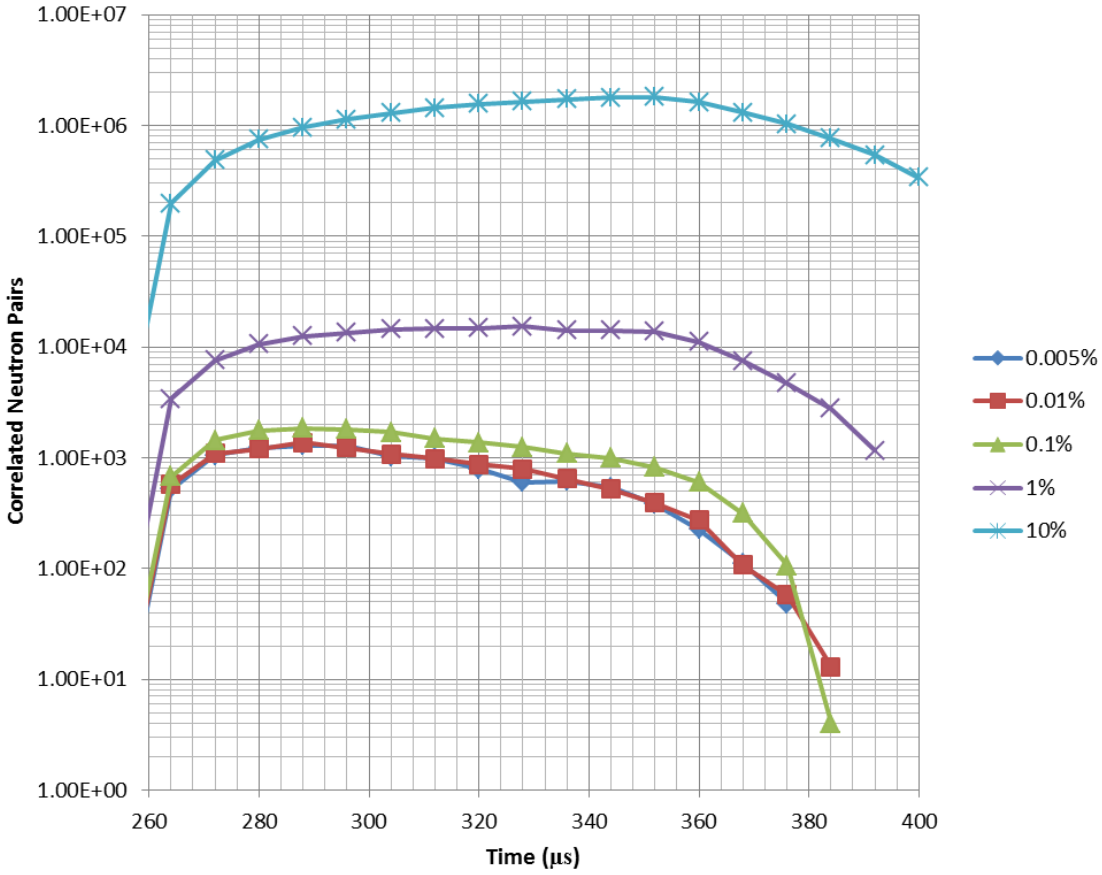

Figure 24. Doubles pulsed histogram calculated by SPNS varying the probability of fission with windows

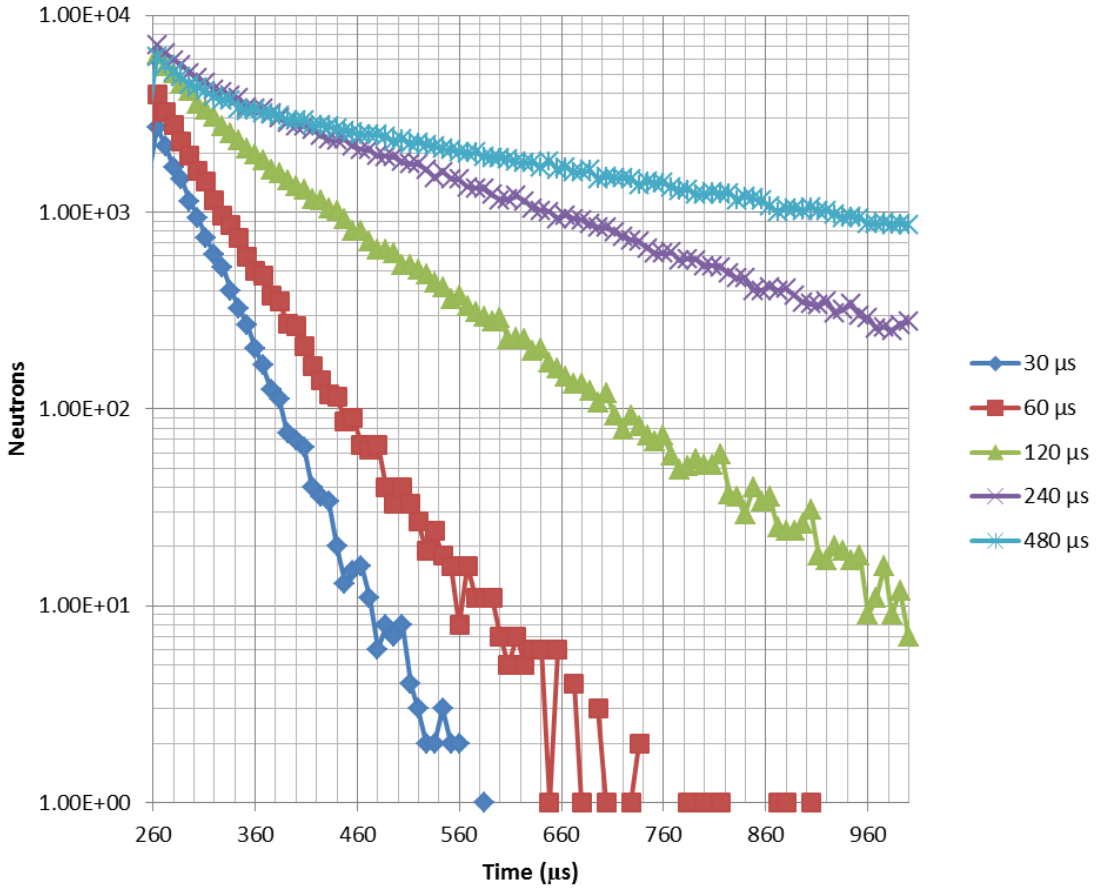

Figure 25. Singles pulsed histogram calculated by SPNS varying the fission die-away with windows

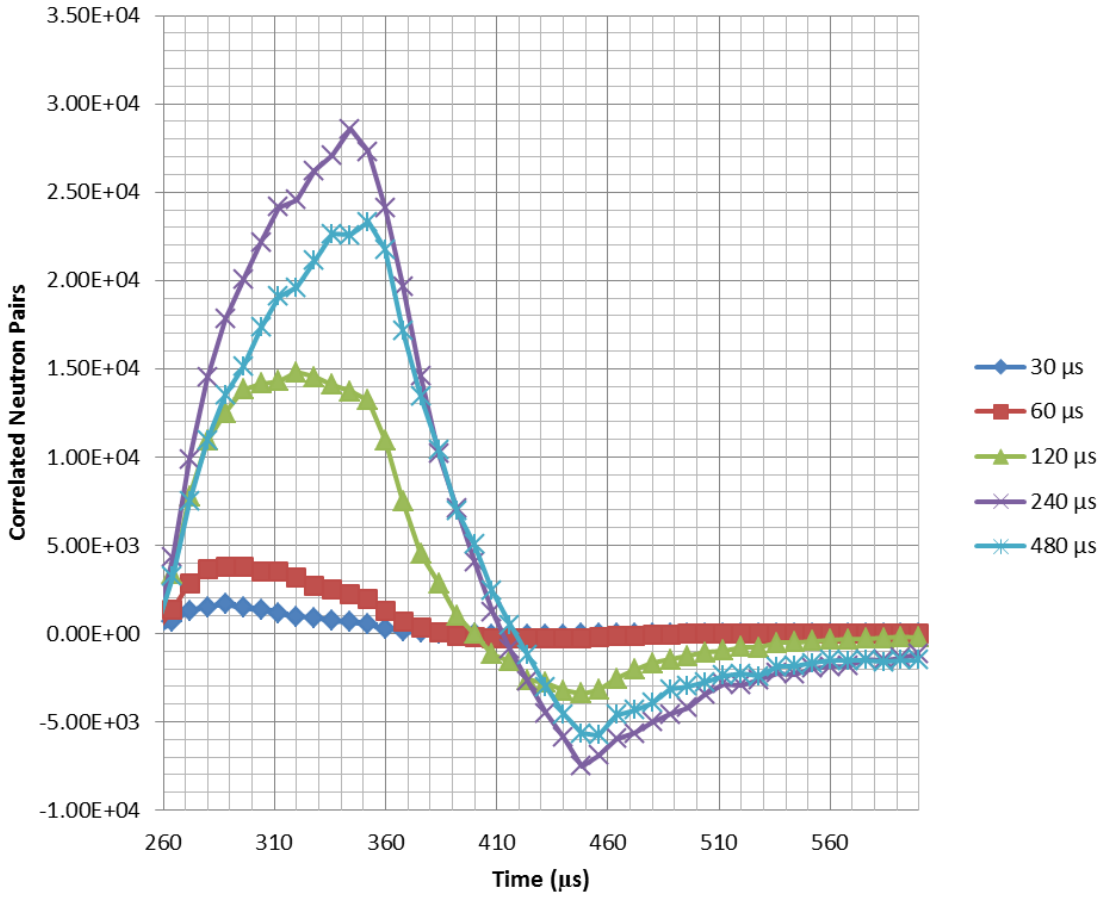

Figure 26. Doubles pulsed histogram calculated by SPNS varying the fission die-away with windows

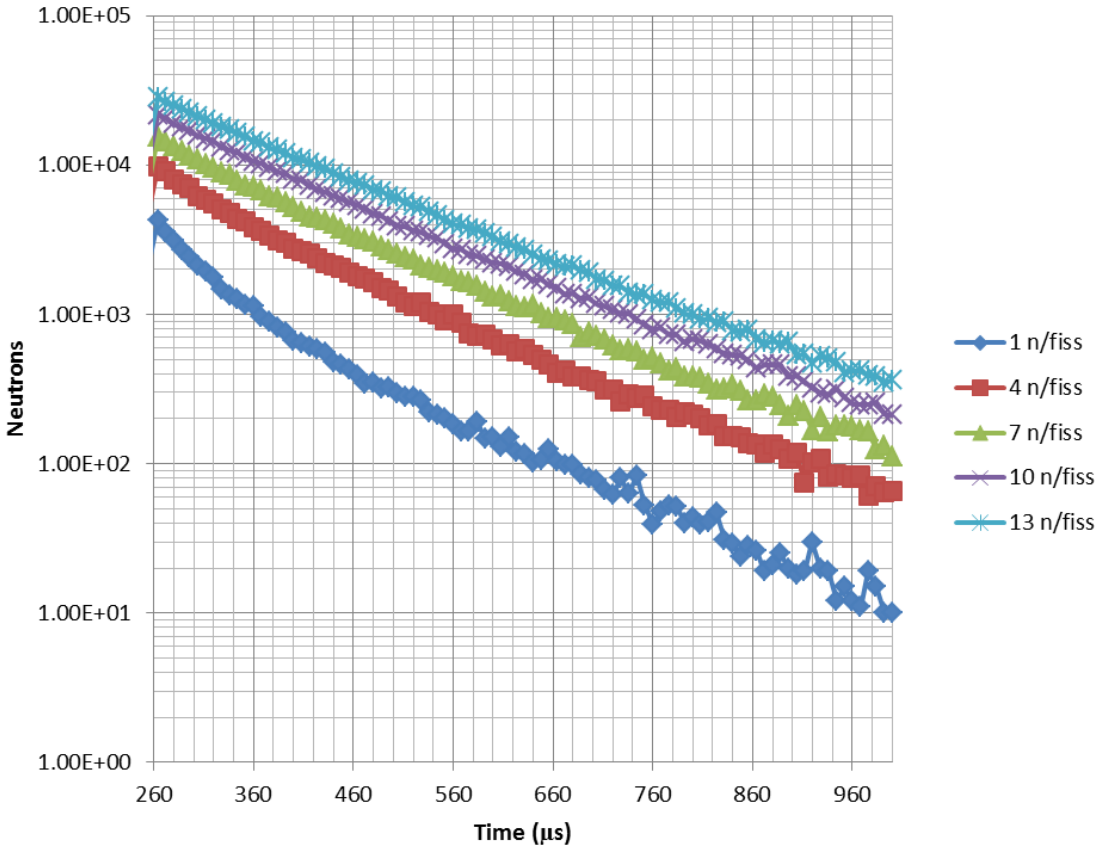

Figure 27. Singles pulsed histogram calculated by SPNS varying the multiplicity distribution with windows

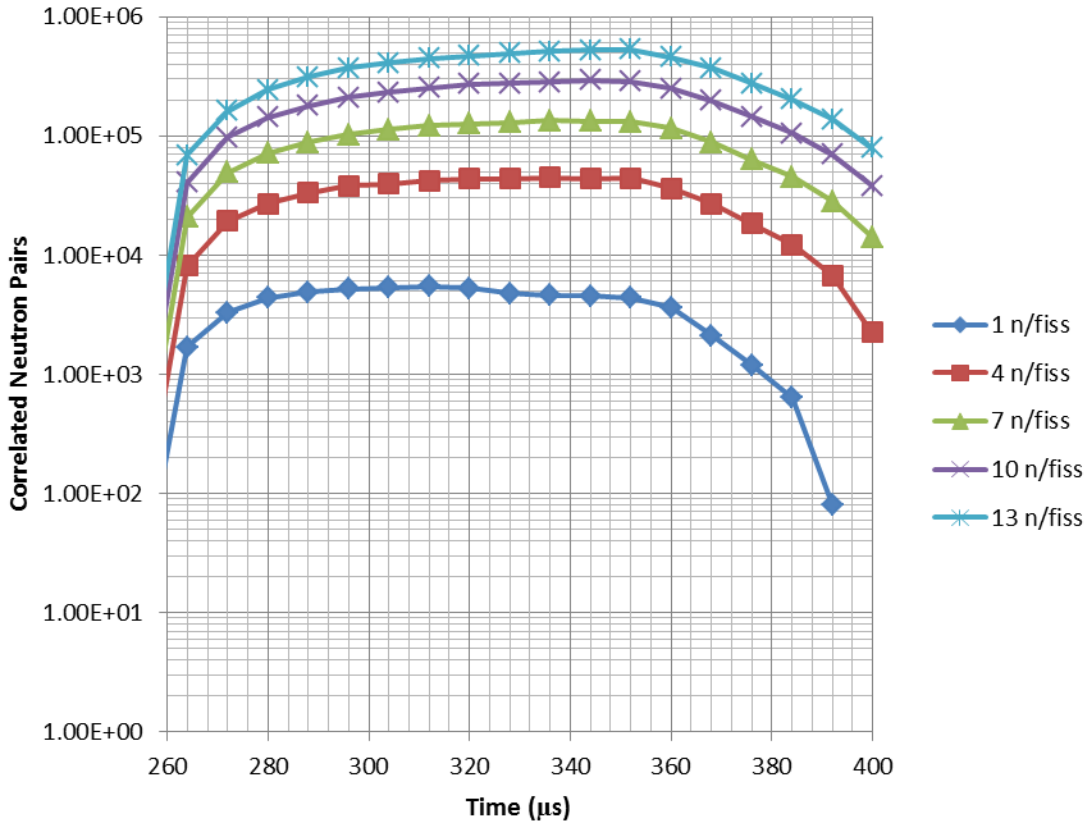

Figure 28. Doubles pulsed histogram calculated by SPNS varying the multiplicity distribution with windows

The dome-like shape in the doubles pulsed histograms in Figures 24, 26, and 28 are an artifact of the windows subtraction. For shift register analysis the first neutron looks back in time, so first neutrons close to the windows subtraction are looking back at no data and so do not record either R+A or A pairs. Since the R+A gate is filled before the A gate in time, this also results in a time lag between R+A values appearing and A values appearing. The lag results in the A values peaking later in time than the R+A values, which exacerbates the previous long delay lagging effect and generates a region of negative doubles in 26 and 28. This region is not shown with the scale of the plot in

Figure 24, but is still present. The time lag is not an issue for typical shift register analysis since this analysis sums the pairs from both gates for the entire data collection file before subtracting the A gate value from the R+A gate value.

As the fission probability is increased in Figure 23, the curves are translated up the y-axis because more neutrons are created in the system from a larger number of fission reactions which means that more neutrons will be detected. Additionally, as the probability of fission increases, the singles curve stretches to the right since more neutrons generated means a greater chance that some will undergo additional fissions and keep the chain alive for a longer period of time. There is negligible distinction between the two lowest fission probability perturbation cases. This is because the neutron production difference between the two perturbations is relatively small when compared to difference between any of the other perturbations. As more data was collected for these two perturbation cases, they would eventually separate from each other more distinctly. The doubles pulsed histogram in Figure 24 also is translated up as the fission probability is increased because an increased fission probability generates more correlated neutron pairs. Again the lowest probability perturbations are hard to distinguish in the doubles pulsed histogram due to the fairly small in the number of fissions they produce relative to the other perturbation cases.

In Figure 25, the increase in the number of singles neutrons detected as the fission die-away increases is due to less channel dead-time loss since the neutrons from fission are more spread out in time. The decrease in neutrons lost to channel dead-time also increases the number of doubles detected as the fission die-away is increased in

Figure 26 up to a certain point. There are more doubles detected in the 260 μs die-away case than the 480 μs die-away case because eventually the larger die-away value will lengthen the time between correlated neutron pairs enough that the gains from less channel dead-time loss will be negated by the loss from correlated neutron pairs falling outside of the gate width from each other.

In Figures 27 and 28, the growth in the number of singles and doubles events are due to more neutrons being produced per fission event. As the number of neutrons from fission increases in the singles pulsed histogram in Figure 27 it will increase the numbers of neutrons being detected and increase the probability that fissions will occur later in time resulting in a positive translation along the y-axis and a lengthening of the pulse curve. In Figure 28, more neutrons per fission results in more correlated neutron pairs being detected in the shift register analysis which in turn translates the curves from the cases positively on the y-axis.

The difference between seeing more doubles counts due to an increase in the number of fission neutrons in the system versus seeing an increase in the doubles counts due to increasing the die-away time can be seen by comparing Figure 28 with Figure 26. In Figure 28, an increase in the doubles via more correlated neutron pairs being generated in the measurement object results in all of the perturbation curves turning from positive to negative at about the same point whereas in Figure 26, the point at which the perturbation curves arrive back at zero gradually shifts to the right as the die-away increases.

CHAPTER V

EXPERIMENT AND ANALYSIS PROCEDURE

After gaining an understanding of some general physics in the pulsed environment and perturbing some pulsed histograms to understand how effects manifest themselves in some of the data analysis techniques used in the thesis, experimental data was examined to determine the viability of this process for treaty verification activities. The experimental procedure for data collection as well as the analysis methodology for each technique is explained below.

V.A. Experimental Procedure

For an experimental run, the experimental hall is cleared and then the LINAC operator turns on the LINAC. Upon activating the LINAC, the operator ensures that it has stabilized at a constant operating beam current. The operating current controls the number of electrons and therefore the number of photons that are produced in a pulse. This needs to be relatively constant during data acquisition to ensure that differences in the neutron data only stem from the differences between measurement objects rather than drastic changes in the number of neutrons that the LINAC generates on average in the measurement object. All data in this thesis was collected with the LINAC at a 125 Hz repeat rate and a constant 3 microamperes current.

For each day of the June 2012 measurement campaign, a background measurement was taken in the morning to both check the room background and ensure that the LINAC and the data collection electronics were functioning correctly. Data

collection only begins once the operator certifies that the beam current from the LINAC is constant. Background is then counted for 10 separate runs of 1 minute each, or 10 minutes of data collection total. After collecting the background a measurement plan was created for the day. This measurement plan included measurement objects with varying masses of depleted and enriched uranium as well as measurement objects with a fixed amount of nuclear material but varying amounts of moderation around the material.

Once the LINAC was turned on and the LINAC operator indicated that the beam current was stable, data collection could begin. For each measurement object, a minimum of 10, 1 minute list-mode files were created. Once data collection was completed, the data taken was analyzed to ensure that it was of adequate quality and that no discernible electronic noise effects had impacted the counts coming from the measurement object. Once the data was examined, a decision was made to either continue on the previously developed plan or a new measurement object was devised that shed light into an effect seen in the previous measurement. This entire process was repeated until the end of the day.

V.B. PSR Analysis Method

The PSR was one of the two analysis tools used to analyze the data taken as part of the US-UK measurement campaign as well as the June 2012 measurement campaign. PSR analyzed each of the 10, 1 minute data files taken per measurement object and then generated a singles and doubles value for these files as well as a singles, reals plus accidentals (R+A), and accidentals (A) pulsed histogram in a .csv file. The PSR parameters used to analyze the data are included in Table I. These individual files were

compiled into one large Excel data file for each measurement object. Excel was then used to generate the average value of the singles, doubles, and individual bins in the pulsed histograms from all the one minute measurements as well as the standard deviation between the values to give some indication of the error in the average. The first two measurement files were excluded from the average and standard deviation calculations because the delayed neutron production was not yet at equilibrium during the acquisition of the first two files. In addition, the doubles value was divided by the singles value both for the entirety of a measurement and for individual bins to give a value that was proportional to the multiplication in a measurement object. The averages and standard deviations were then used to compare different measurement objects.

Table I. PSR parameters used during analysis

Parameter	Value
Pre-Delay	3 μs
Gate Width	96 μs
Long Delay	8000 μs
Trigger Offset	70 μs
Window	260 μs
Veto	2 μs

It was determined that the doubles and D/S pulsed histograms provided the most distinctive information on the moderation of a measurement object as well as the presence of nuclear material in an object since the peaks generated in these graphs are a direct result of fissions neutrons in the sample. Therefore another analysis technique was devised using these pulsed histograms. This process involved finding the maximum bin

value of the curves from the D/S and doubles graphs as well as the average time of the curves after the pulse. Only positive bins were included in the mean time computation, and it was assumed that the curve terminated as doubles went negative. In principle, since the graph was discrete, the time average value was determined by taking leftmost and rightmost sums of the neutron count curve and then finding the time location where the difference between the two sums was minimized. This minimal distance point indicated that the point at which the two sums would be equal would either be the bin to the left of the meeting location or the bin to the right of the meeting location. If the error on the difference was larger than the difference, then it was impossible to determine if the bin on the left or the bin on the right would contain the point at which the two sums would be equal. In these cases, the time bin value at the meeting location was taken to be the time average for that curve, and the error was set to the size of a bin. If the difference between the sums was larger than the error, then it could more specifically indicate whether the bin on the left or right would contain the actual point at which the two sums would be equal. In these cases the midpoint of the identified bin was used as the time average for the curve, and the error was set to half of a bin.

Background was subtracted from the traditional doubles and singles values from shift register analysis as well as each individual bin in the pulsed histogram outputs to account for environmental background as well as room activation from the high-energy photons. Non-background subtracted doubles and singles values were used to compute D/S, before the D/S background value was subtracted to reduce the error in the D/S term.

V.C. SPNS Analysis Method

The SPNS program was also used in conjunction with the PSR program in an attempt to generate data from SPNS parameters that matched the pulsed histogram output of experimental data. The PSR analysis parameters used in conjunction with SPNS fitting are included in Table II. This analysis typically involved guessing initial SPNS parameters based on previous experience with the program, processing the resulting run with PSR, and then graphing the results in Microsoft Excel on top of the experimental data that was to be fit. Various SPNS parameters would then be altered until the program eventually generated a fit that was reasonably close to the experimental data. The SPNS parameters from this fit could then be compared to the SPNS parameters from other experimental data fits in order to analyze the samples. SPNS fits were only performed against the data from one of the one minute runs from a particular measurement object rather than a plot of the average of eight of the measurements used in the PSR analysis. This was to ensure that the statistical variation in the SPNS parameters was comparable to the statistical variation of the experimental data.

Table II. PSR analysis parameters used for SPNS fitting

Parameter	Value
Pre-Delay	3 μs
Gate Width	96 μs
Long Delay	8000 μs
Trigger Offset	70 μs
Window	0 μs
Veto	0 μs

If the parameters from SPNS fitting are going to be compared between different measurement objects, then it would be helpful if each measurement object only could be best fit by one unique set of SPNS parameters. In order to determine if this was the case, a least squares fitting (LSF) analysis was performed with SPNS fitting a non-background subtracted polyethylene moderated HEU object. Background is not typically subtracted with SPNS fits because the background is accounted for in some of the SPNS parameters. The equation for LSF is:

$$\left[\frac{Sim - Exp}{Exp}\right]^2 \tag{3}$$

In this equation *Sim* represents SPNS simulated data and *Exp* represents the experimental data. To use LSF to gauge the uniqueness of SPNS parameters for this data set, the optimal SPNS fitting parameters for the experimental data were determined and then the values were perturbed off of the optimal. The resulting LSF values were then plotted versus the SPNS parameter values. If the SPNS parameters represent a unique fit of the data, then there should be an absolute minimum LSF value corresponding to the SPNS parameter. The initial determination of the uniqueness of SPNS parameters was calculated using singles pulsed histograms only, since at that point the doubles pulsed histograms had not yet been created.

SPNS parameters that model a measurement object tend to be dominant in specific time domains. For instance, the detector die-away time is most likely to impact the LSF value from about 120 μs to 320 μs in a singles pulsed histogram. Table III contains a list of all of the time domains considered in the LSF analysis as well as the parameters expected to most heavily the specific regions. The LSF values in each region

were normalized by the number of bins in that domain to ensure that the LSF values were comparable between regions.

Table III. SPNS parameter time domains used for LSF

Sensitive Parameters	Time Domain
All Bins	120 - 8000 μs
Pulse Region	120 - 1200 μs
(α,n) rate	1200 - 8000 μs
Neutron/Pulse & Detector Die-Away	120- 320 μs
Fast Probability of Fission & Die-Away	200 - 600 μs
Slow Probability of Fission & Die-Away	600 - 1200 μs

The sensitivity of LSF values to the natural fluctuation of the neutrons in each bin for a fixed parameter due to the random number generator in SPNS was also considered since the error bars would bound the ability to call a specific point an absolute minimum. Essentially these error bars represent the precision of the SPNS parameters within the LSF time domains. In order to find this error, 10 SPNS simulations with the same parameter set were run and the LSF values for the specific time domains were computed for each run. The standard deviation of the LSF values was then taken to determine the error. The resulting values are included in Table IV.

Table IV. Error in LSF time domains due to the SPNS random number generator

Time Domain	Error
120 - 8000 μs	6E-04
120 - 1200 μs	3E-04
1200 - 8000 μs	7E-04
120- 320 μs	1E-05
200 - 600 μs	7E-05
600 - 1200 μs	6E-04

In general it was assumed that all SPNS parameters are independent variables, which means that the uniqueness of each parameter can be determined independent of all the other parameters. The only parameters that egregiously violate this assumption are the probabilities of thermal fission coupled with their respective die-away times. In these two cases, SPNS parameters had to be perturbed by both the probability of fission and the die-away to generate a 3D LSF grid. If there was a minimum LSF value evident on this grid, then it could be said that the combination of the fission probability and the die-away had a unique fit for experimental data.

The SPNS parameters studied for this analysis included the neutrons per pulse, the detector die-away time, the two probabilities of induced fission and their associated die-away times, and the (α,n) generation rate. Figure 29 contains the LSF versus parameter modified graphs for the neutrons per pulse, the detector die-away, and the (α,n) generation rate. Error bars are included in the figure but are difficult to see due to the scale of the graph. Additionally, the lines drawn between points do not represent a fit of the points and are only intended to serve as visualization aids.

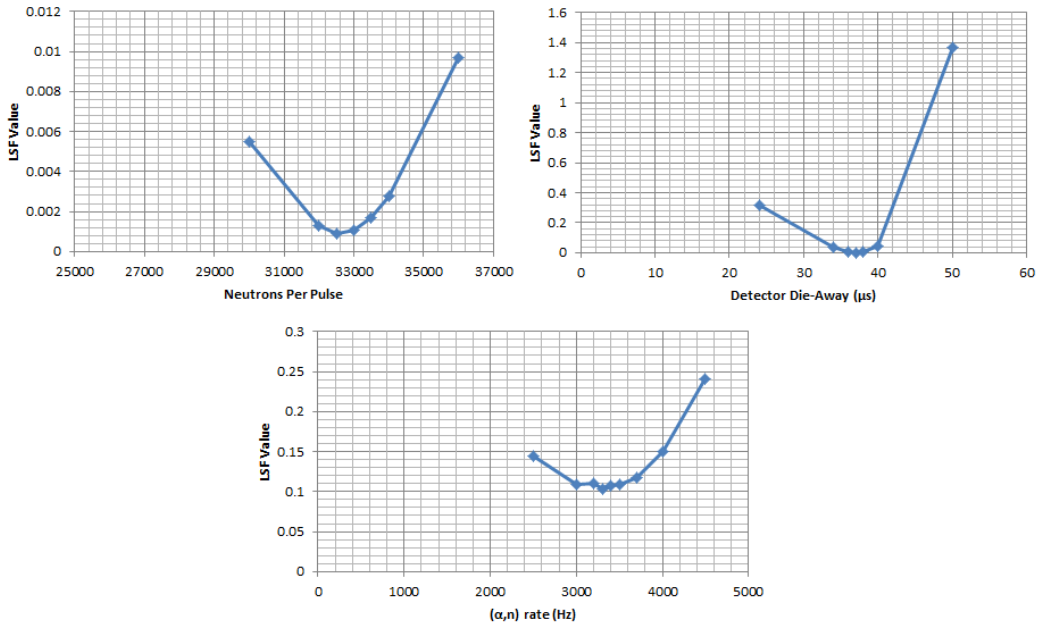

Figure 29. LSF vs. parameter modified graphs for neutrons per pulse (upper left), detector die-away (upper right), and the (α,n) generation rate (bottom)

From Figure 29 it is clear that there is an absolute minimum for the neutrons per pulse, detector die-away, and (α,n) generation rate parameters, so the SPNS fit of these parameters to this experimental data represent a unique solution set. A 3D plot of the fast thermalization fission probability vs. die-away time vs. the LSF value for their region of sensitivity is included in Figure 30. LSF values greater than 0.04 are cutoff in the figure to enable better visualization of the minima region.

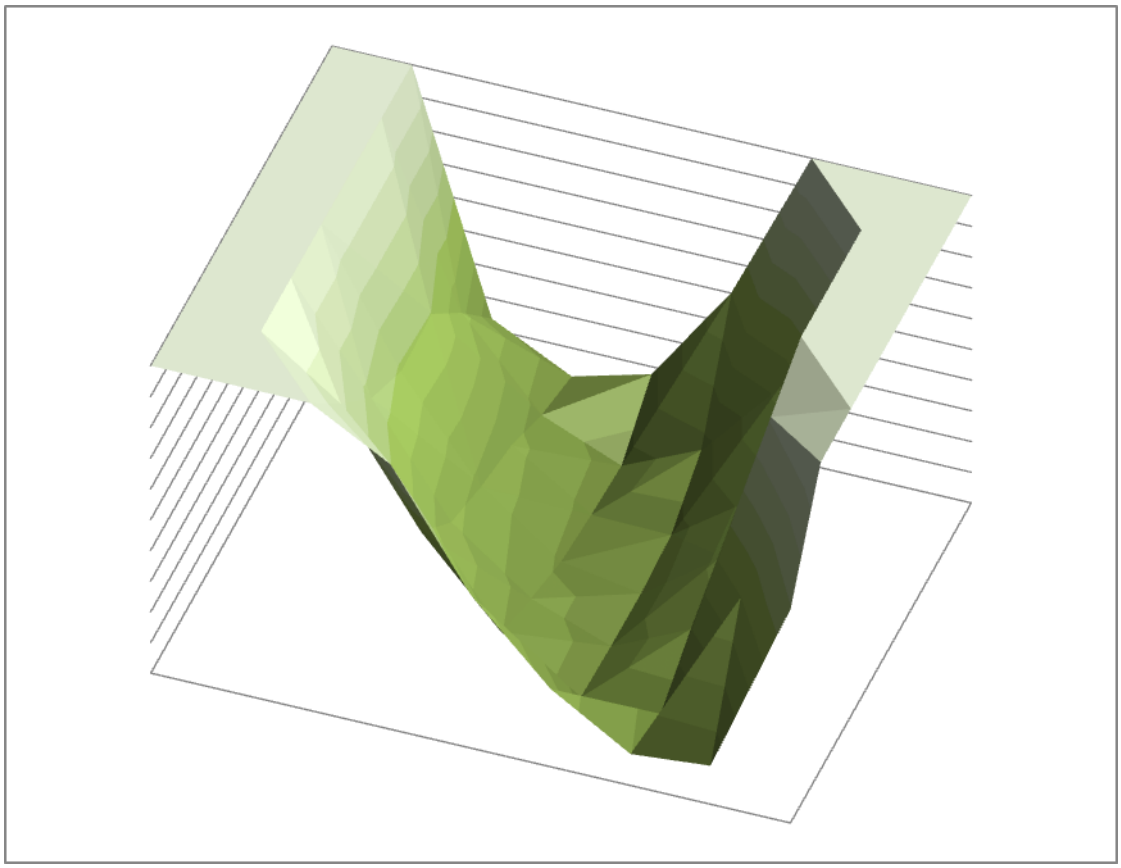

Figure 30. 3D LSF graph with fast thermalization induced fission probability in the y-axis, the die-away (μs) in the x-axis, and the LSF values in the z-axis coming out of the page

Figure 30 makes it difficult to ascertain whether or not the coupled fast thermalization fission probability and die-away time parameters are a unique representation of the measured data, so the numerical LSF values for this region are included in Table V. The random error in the LSF value for this region is 7E-5, so all the values listed in the table are several orders of magnitude above the random error for this region.

Table V. LSF values for the coupled fast thermalization parameters with absolute minimum in bold

	80 µs	90 µs	95 µs	100 µs	105 µs	110 µs	120 µs	130 µs
0.66%	0.1139	0.0652	0.0431	0.0249	0.0127	0.0051	0.0063	0.0348
0.76%	0.0932	0.0412	0.0235	0.0103	0.0026	0.0030	0.0250	0.0850
0.81%	0.0848	0.0334	0.0164	0.0069	0.0022	0.0081	0.0456	0.1289
0.83%	0.0819	0.0292	0.0135	0.0043	**0.0021**	0.0112	0.0532	0.1400
0.85%	0.0768	0.0298	0.0115	0.0032	0.0030	0.0118	0.0582	0.1511
0.90%	0.0639	0.0214	0.0078	0.0026	0.0082	0.0217	0.0882	0.2127
1.00%	0.0518	0.0117	0.0054	0.0088	0.0251	0.0560	0.1553	0.3359

The 3D graph in Figure 30 demonstrates that the coupled fast thermalization parameters exhibit the functional form that is expected in least squares fitting if there is a unique fit, while the specific LSF values in Table V show that there is an absolute minimum for these two coupled parameters. Based on these two pieces of evidence, the coupled fast thermalization probability of fission and die-away SPNS parameters are capable of a unique fit of this experimental data.

Figure 31 and Table VI are included to determine whether or not the coupled slower thermalization fission parameters are unique. Figure 31 is used to demonstrate the shape of the region. The least squares fitting values greater than 0.1 are cutoff in the figure to give a better picture of the data. With a random error of 6E-4, this fitting region has greater statistical fluctuations, which makes it more difficult to see trends in the figure; however the general trend in the slow thermalization values is similar to the fast thermalization values. Both have a valley of parameters that generate markedly better LSF fits, demonstrated with low LSF values. Table VI shows that there is an absolute

minimum outside of the random error from the LSF variation, which when combined with the shape of Figure 31 lends confidence that the coupled slow thermalization parameters are a unique fit of this experimental data.

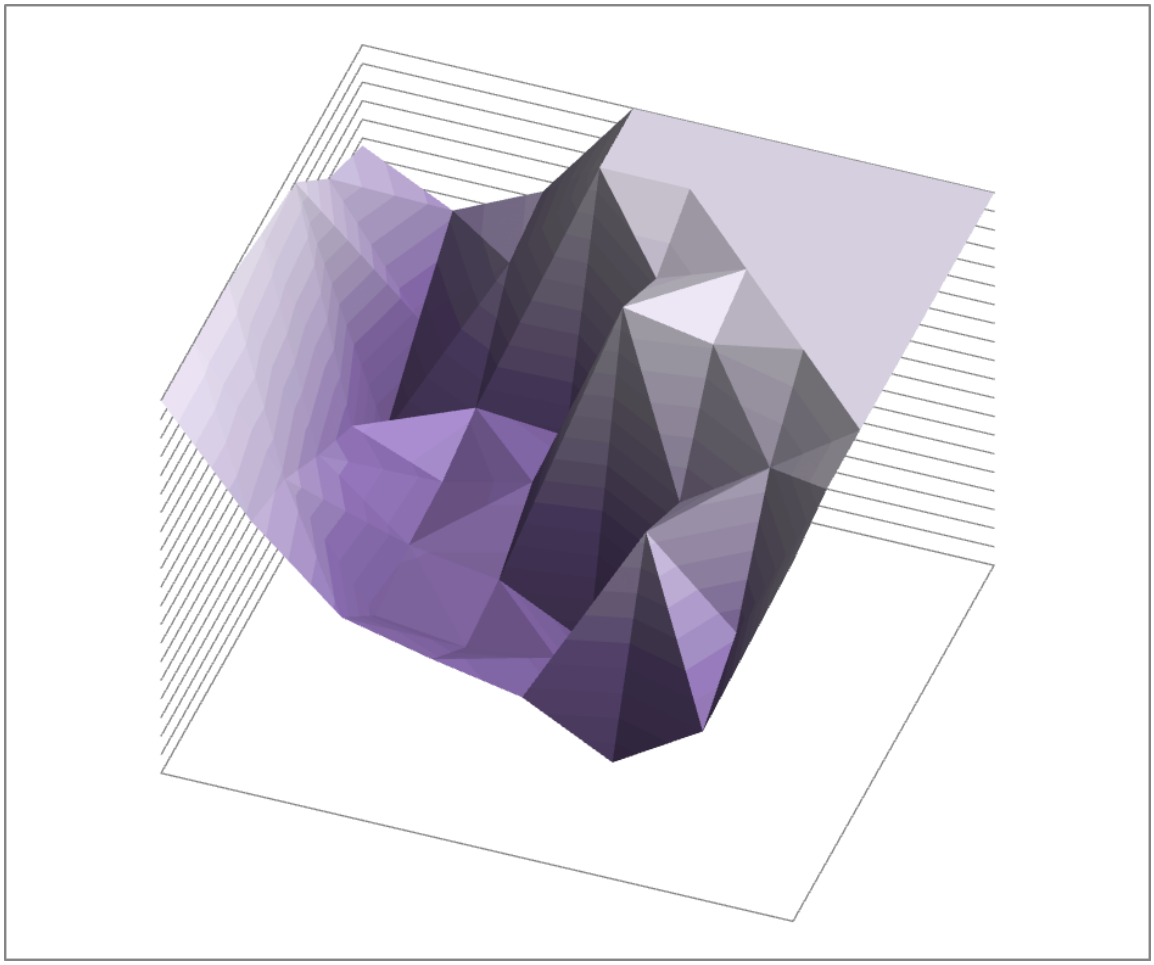

Figure 31. 3D graph of the slow thermalization induced fission probability in the y-axis, the die-away (μs) in the x-axis, and the LSF values in the z-axis coming out of the page

Table VI. LSF values for the probability of fission and die-away parameters of slow thermalization with absolute minimum in bold

	175 µs	185 µs	195 µs	200 µs	205 µs	215 µs
0.05%	0.158	0.122	0.113	0.097	0.095	0.080
0.09%	0.073	0.069	0.059	0.051	0.034	0.048
0.11%	0.053	0.038	0.041	0.042	0.046	0.069
0.13%	0.048	0.035	0.037	0.047	0.044	0.104
0.15%	0.043	0.038	**0.025**	0.103	0.092	0.176
0.17%	0.031	0.077	0.069	0.096	0.143	0.262
0.19%	0.045	0.056	0.084	0.174	0.265	0.363
0.25%	0.096	0.199	0.177	0.457	0.546	0.936

The LSF fitting analysis has shown that all of the SPNS parameters are a unique fit of this experimental data set. The final SPNS fit of the singles pulsed histogram from the experimental data is included in Figure 32. Unfortunately, the ability of SPNS parameters to uniquely fit one set of experimental data does not prove that they are capable of uniquely fitting all sets of experimental data. Some sets of data could violate the point model in a way that invalidates the use of SPNS fitting. In order to truly be certain that all data sets are unique, this analysis would need to be performed for every set of data fit by SPNS.

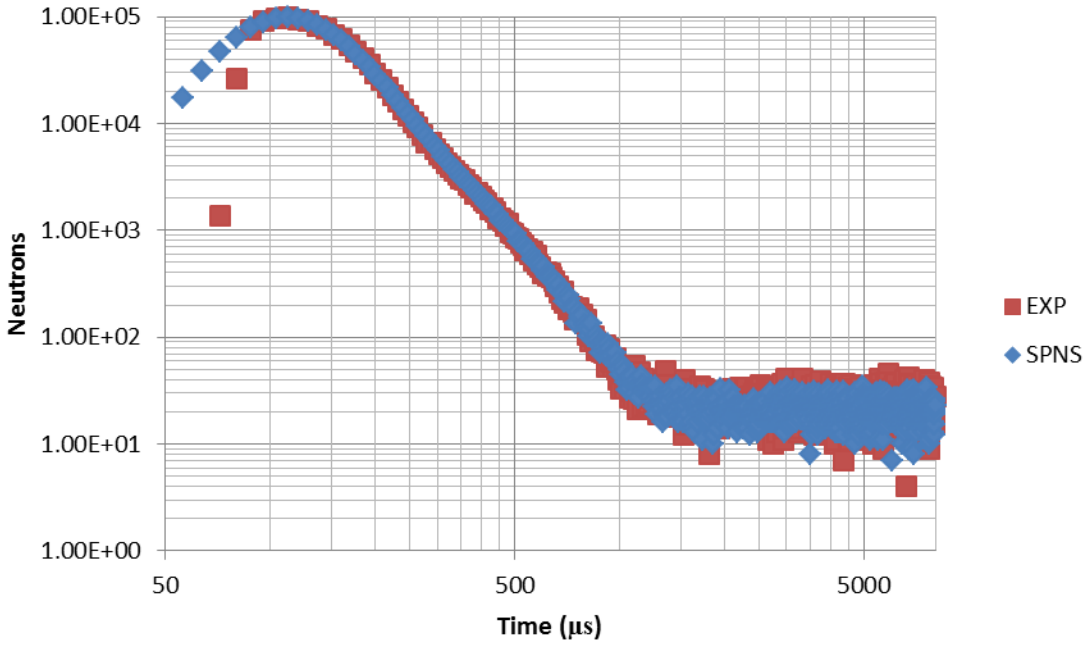

Figure 32. Singles pulsed histogram of the best LSF SPNS fit of experimental data

At present there is not an automatic fitting algorithm coupled with SPNS fitting which generates LSF values, so the process of proving uniqueness is very time intensive. For the purposes of this thesis, SPNS ability to uniquely fit one set of experimental data is taken as evidence that SPNS fits of other experimental data sets have a high probability of also being unique given that other sets of data used in measurement objects would also not likely seriously violate the point model assumption. Additionally, due to the time cost of SPNS fits without an automated fitter, other SPNS fits included in this thesis were fit visually rather than by using LSF values. This is not thought to be a truly restrictive problem since the experimental data fit used as part of this uniqueness study had been previously fit visually, and only minor changes were made to the SPNS fitting parameters due to the LSF values. Also general trends should still be discernible

with the use of visual fitting in SPNS, individual terms may just have a higher error than they would had LSF fitting been employed.

After the introduction of the doubles pulsed histograms, the uniqueness study was revisited to investigate the comparison between SPNS generated doubles and the experimental data doubles. Figure 33 shows this fitting using the same parameters that were previously established in the uniqueness study. There is clearly a significant fitting discrepancy in the doubles pulsed histogram even though the SPNS parameters were shown to be a unique representation of the data in the singles pulsed histogram.

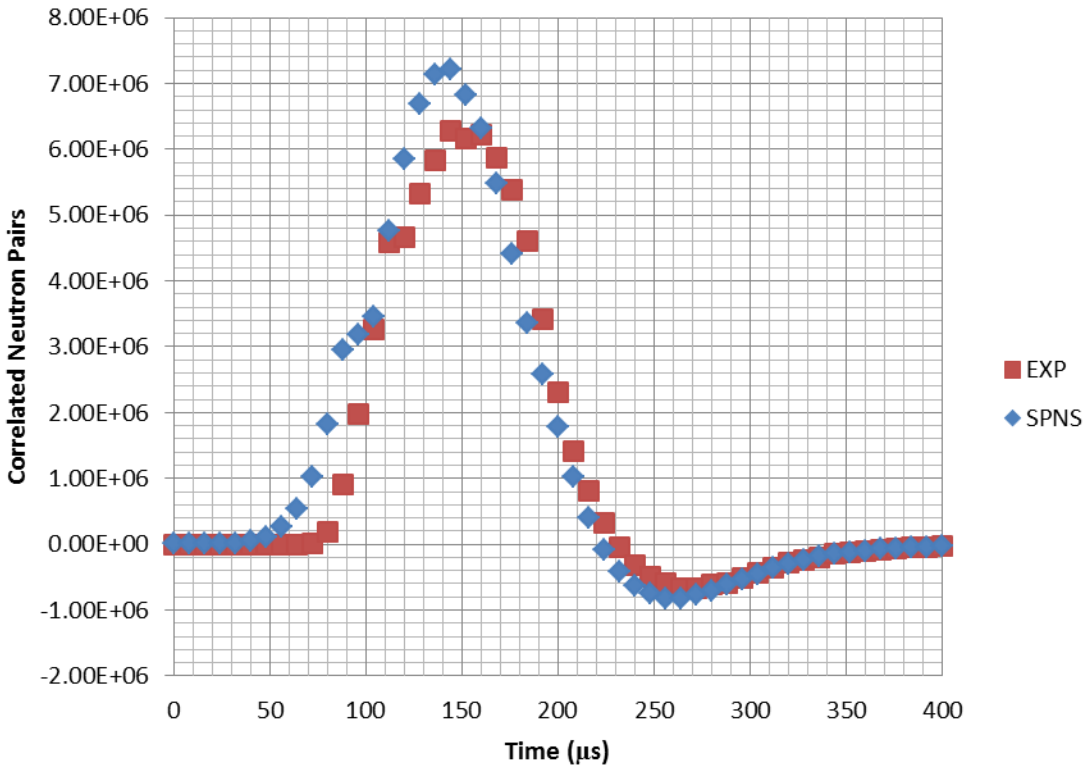

Figure 33. Doubles pulsed histogram of LSF fit SPNS onto experimental data.

The only other parameter that was not studied during the uniqueness study that represents the measurement object is the induced fission multiplicity distribution. The perturbation study with SPNS demonstrated that the induced fission multiplicity distribution can significantly impact the doubles pulsed histogram appearance, and so if the distribution is not representative of the sample, it may lead to the errors that are being seen in Figure 33. The current induced fission multiplicity distribution was set according to typical induced fission values for metal HEU, but it is possible that it could still be incorrect compared to the sample in front of the LINAC, since that sample would not be purely HEU. Another potential source of error could be the channel dead-time values. The channel dead-time in the system is known, but it is possible that there is a large enough difference between the model for dead-time used in SPNS and the actual behavior of the dead-time in the pulsed region that it leads to a large discrepancy between the SPNS fit and the experimental data in the doubles pulsed histogram.

Between the two potential sources of error, the discrepancy is more likely to be caused by an induced fission multiplicity distribution that is not representative of the measurement object. Due to how the induced fission multiplicity is implemented in SPNS, it is not possible to easily perturb the values to easily establish a more representative fit because the multiplicity is comprised of up to 20 potential entries. Therefore other methods of gaining a more representative fit of the doubles pulsed histograms are needed before the fit of these graphs will yield information on the nuclear data.

Changing the induced fission multiplicity distribution would likely lead to a different unique parameter set to fit the singles pulsed histogram since the induced fission multiplicity is not independent of the fission terms in the SPNS program. This means that a true fitting algorithm for SPNS would need to iterate several times between the doubles and singles pulsed histograms to find all of the parameters within acceptable error. With the issue in doubles pulsed histogram fitting noted, SPNS fitting of other experimental data sets will focus solely on the singles pulsed histogram, and it will be assumed that the HEU induced fission multiplicity distribution is representative of the induced fission multiplicity of the other samples.

CHAPTER VI

DATA ANALYSIS

Data analysis has been split into data sets in order to assess the capability of the analysis tools to meet the research objectives. The first set of data is intended to test the ability of the analysis to distinguish between samples with multiplying nuclear material such as plutonium (Pu) and highly enriched uranium (HEU) from non-multiplying samples. The second set of data is intended to gauge the ability of the analysis methodology to determine the mass of multiplying samples, and the third set of data has been constructed to explore what the analysis method can ascertain about the state of moderation in the sample. The fourth set of measurement data includes test objects from the US-UK measurement campaign. The exact construction of these objects and procedures related to the data collection is discussed extensively in [8], and the objects are included to get an idea of the analysis methods performance on more realistic verification objects.

VI.A. Fissile Content Analysis

The experimental data used to analyze the ability of PSR and SPNS to distinguish between samples with multiplying material versus non-multiplying material came from the US-UK measurement campaign[7]. Most of the objects in this part of the analysis were standards for comparison during that campaign, so their construction is not discussed in any of the literature in detail. In between measurements on an object, a box with 2 inches of polyethylene on each side could be added to the experimental setup to

provide a known amount of moderator. Additionally, the plutonium and highly enriched uranium standards were constructed such that each object had about the same multiplication. Table VII shows the name of each sample in this data set and the corresponding physical description.

Table VII. Physical descriptions of the measurement objects in the fissile content analysis data set

Name	Physical Description
DU	Depleted Uranium
W	Tungsten
W+p	Tungsten with 2” polyethylene box
HEU	Highly Enriched Uranium
HEU+p	Highly Enriched Uranium with 2” polyethylene box
Pu	Plutonium
Pu+p	Plutonium with 2” polyethylene box

The result of the shift register analysis with windows using the PSR is included in Table VIII. An important note on this table is that the D/S values do not represent the D column divided by the S column. In all of these tables, the D/S operation is performed with non-background subtracted D and S counts before the D/S from the background measurement is subtracted. From this table, the tungsten sample is clearly distinguishable from the rest of the samples, as it has a low singles and doubles rate both with and without moderation added. The depleted uranium (DU) standard was not measured with added moderation in this data set, but a different set of data with DU with

and without moderation is included in the mass data set. The difference in the singles and doubles rates with and without moderation added to both the HEU and the Pu sample clearly distinguish these samples from the rest of the measurement objects in the series, and indicate that there is multiplying material present in both samples. After adding polyethylene, both of these samples are also the only measurement objects with a D/S value greater than 1, which is another indicator of the presence of multiplying material.

Table VIII. Shift register analysis with the PSR and windows

	S	σ_S	D	σ_D	D/S	$\sigma_{D/S}$
DU	2890	35	217	10	-0.089	0.008
W	4	5	11	3	0.034	0.011
W+p	-20	4	-3	3	0.003	0.013
HEU	1985	20	271	6	-0.032	0.008
HEU+p	11669	93	49146	706	3.940	0.068
Pu	3805	15	361	14	-0.073	0.008
Pu+p	9205	104	18907	581	1.823	0.065

The pulsed histogram analysis with the PSR also distinguishes the Pu and HEU samples from the rest of the measurement objects based on the observed change in both the max peak value and the average time of the peak in the doubles pulsed histogram in Table IX and the D/S pulsed histogram in Table X. The pulsed histograms for all the data sets are included in Appendix A.

Table IX. Doubles pulsed histogram analysis for multiplying material samples

	W	W+p	DU	HEU	HEU+p	Pu	Pu+p
Max	80	25	1303	1540	361558	1443	133142
σ_{Max}	27	18	63	115	6906	62	4057
Time (μs)	296	272	304	312	332	312	332
σ_{time} (μs)	8	8	8	8	4	8	4

Based on the values in Table IX, the DU, HEU, and Pu samples are distinctly different from the Tungsten (W) sample before polyethylene is added based on their max doubles rates. The average time in these samples is also identical at this point. Once polyethylene is added, the Pu and HEU samples further distinguish themselves from the Tungsten since Tungsten shows a decrease in the average peak time of the samples doubles peak with the addition of polyethylene whereas both HEU and Pu show a significant increase. The average peak time value is expected to increase whenever moderation is added to an under moderated multiplying sample since it would return slow neutrons to the sample later in time thereby allowing for later fission reactions. The increase in fissions would also increase the doubles rate. Without multiplying material present, the addition of polyethylene will simply result in a lower count rate due to neutron absorption in the sample.

Table X. D/S pulsed histogram analysis for multiplying material samples

	W	W+P	DU	HEU	HEU+P	PU	PU+P
Max	0.13	0.02	1.20	1.43	25.86	1.40	15.72
σ_{Max}	0.16	0.02	0.19	0.18	0.64	0.16	0.59
Time (μs)	320	272	328	328	340	328	340
σ_{time} (μs)	8	8	8	8	4	8	4

The D/S results in Table X also support the results from the doubles pulsed histogram analysis. In this table, the HEU and Pu max multiplication values are actually higher than the max DU value, but they have overlapping error bars. The tungsten sample, on the other hand, has multiplication equivalent to zero both with and without the polyethylene moderator. Again both the Pu and HEU samples show a dramatic increase in multiplication with polyethylene added, and show an increase in the peak times as expected.

Table XI contains the measurement object fitting parameters from SPNS. In this table, the fission probabilities, which signify more fission reactions in a sample, are much higher whenever the polyethylene moderator is added to the HEU and Pu samples. These samples also demonstrate a notable lengthening of their fast die-away times as polyethylene is added. The fission probability cannot be used to distinguish between the HEU, PU, and DU samples before the moderator is added. The SPNS fitting also shows that the addition of a polyethylene moderator depresses the steady-state rate of samples outside of the pulsed region since more of those neutrons are absorbed by the poly. This same effect is likely the cause of the depression in the number of neutrons per pulse as

polyethylene is added. The addition of the polyethylene does not appear to have a consistent effect on the slower thermalization die-away time in this particular data set, although this is not really expected since the slower thermalization die-away is more often indicative of room conditions. The SPNS fits of all data are included in Appendix B.

Table XI. SPNS fitting parameters for the multiplying sample data set

	N/pulse	Fast Thermal Fission Prob.	Slow Thermal Fission Prob.	Fast Die-Away	Slow Die-Away	Steady-State Rate
DU	30000	0.0008	0	120	0	350000
W	16667	0.00057	0.00001	120	240	1800
W+p	16667	0.00054	0.00003	120	240	1500
HEU	30000	0.0001	0.00075	120	420	20000
HEU+p	22000	0.044	0.004	180	380	15000
Pu	30000	0.0008	0.0003	120	240	45000
Pu+p	25000	0.2	0.0055	200	300	30000

VI.B. Mass Analysis

The experimental data used for the mass assessment was taken as part of the June 2012 measurement campaign. There are eight data sets total in this set, four of which used depleted uranium and four of which used highly enriched uranium. The four DU data sets alternated between one and two plates of uranium in the beam line with and without polyethylene moderation, while the HEU repeated this pattern with two plates of HEU. Each HEU plate was 272g of uranium with a 93.27% U-235 enrichment, while each DU plate was 300g of uranium with a 0.2% U-235 enrichment. When the

polyethylene moderation was in place it consisted of 3 inches of polyethylene around the nuclear material on all sides in the plain of the LINAC and 1 inch of polyethylene moderation between the material and the sample stand on the bottom as well as on top of the nuclear material. The moderating structure was built out of 1 inch polyethylene blocks. The HEU or DU plates were centered on the LINAC using the LINACs alignment lasers for each run with the largest surface of the material facing the LINAC and the thinnest part of the material running in parallel to the line of the LINAC. Whenever both plates were in the beam line they were either stacked one behind the other when moderating material was left out of the beam line or with 1 inch of polyethylene between them whenever moderating material was present. This additional 1 inch of polyethylene was also present whenever only one plate was used, with an air gap filling the location where the other plate would reside. The plate closest to the LINAC was always the plate left in place. Table XII lists all of the sample descriptions next to their data set designations.

Table XII. Sample descriptions of the mass analysis data set

Name	Physical Description
1DU	1 Depleted Uranium plate (300g)
2DU	1 Depleted Uranium plates (600g)
1HEU	1 Highly Enriched Uranium plate (272g)
2HEU	2 Highly Enriched Uranium plates (544g)
1DU+p3	1 Depleted Uranium plate with 3" side polyethylene & 1" between, top, & bottom
2DU+p3	2 Depleted Uranium plates with 3" side polyethylene & 1" between, top, & bottom
1HEU+p3	1 Highly Enriched Uranium plate with 3" side poly & 1" between, top, & bottom
2HEU+p3	2 Highly Enriched Uranium plates with 3" side poly & 1" between, top, & bottom

The results from the shift register analysis using the PSR are included in Table XIII. As expected, the addition of another plate of material always increased the singles and the doubles rates. The HEU and DU materials have very similar singles and doubles behavior as plates are added without polyethylene, but exhibit very different singles and doubles behavior once moderation is added. With polyethylene in the sample a comparable amount of HEU produce more singles and doubles counts than DU. Adding a second plate of moderated HEU to the HEU measurement object also grows both the singles and doubles counts at a faster rate than adding a similar amount of DU to the DU measurement object. The D/S values are all lower than background with the exception of the larger polyethylene moderated DU measurement object and both moderated HEU measurement objects. This does not indicate that D/S is not useful for mass determination but may instead show that it is difficult to use D/S as a mass indicator for

objects with small amounts of nuclear material since those objects do not generate large neutron signals relative to background.

Table XIII. Shift register analysis results from the mass analysis data set

	S	σ_S	D	σ_D	D/S	$\sigma_{D/S}$
1DU	450	13	41	7	-0.23	0.02
2DU	879	35	113	20	-0.25	0.03
1HEU	382	14	55	9	-0.18	0.02
2HEU	745	18	129	11	-0.21	0.02
1DU+p3	63	7	24	8	-0.01	0.03
2DU+p3	195	8	127	8	0.08	0.02
1HEU+p3	472	13	389	17	0.22	0.03
2HEU+p3	1329	24	1935	53	0.82	0.04

Both singles and doubles values increase whenever mass is added, however more measurements would need to be taken to build a calibration curve for the material for these rates to be tied back into an actual mass value for the nuclear material in the measurement object. The present data set is not sufficient for building a true calibration curve because most verification objects in a treaty verification regime would have substantially more nuclear material, something on the order of several kilograms rather than the 0.5 kg that was measured here, and the measurement objects in this set were not representative samples of any weapons object.

The PSR doubles pulsed histogram analysis results are presented in Table XIV, and the D/S pulsed histogram analysis results are presented in Table XV. The max values for both Table XIV and XV approximately double when plates are added to the

DU and HEU samples without moderation. Plates added to the moderated samples result in a much larger change in the max value, although again the HEU experiences the greater change from the addition of a plate. The time values also perform as we would like them to, since they stay relatively the same as small amounts of material are doubled, but change whenever moderation is added. It may also be possible to build a calibration curve out of the max values from the pulsed histogram analysis, although the calibration curve analysis would likely benefit from the better statistics in the shift register analysis.

Table XIV. Doubles pulsed histogram results from the mass study

	1DU	2DU	1HEU	2HEU	1DU+p3	2DU+p3	1HEU+p3	2HEU+p3
Max	355	906	436	975	199	884	2593	12318
σ_{Max}	71	151	77	80	70	80	128	297
Time (µs)	296	300	304	304	312	308	316	320
σ_{time} (µs)	8	4	8	8	8	4	4	8

Table XV. D/S pulsed histogram results from the mass study

	1DU	2DU	1HEU	2HEU	1DU+p3	2DU+p3	1HEU+p3	2HEU+p3
Max	0.23	0.61	0.38	0.68	0.14	0.60	1.55	4.37
σ_{Max}	0.26	0.36	0.26	0.31	0.24	0.27	0.27	0.33
Time (µs)	320	320	328	328	336	328	336	336
σ_{time} (µs)	8	8	8	8	8	8	8	8

Table XVI contains the SPNS measurement object fitting parameters for the measurement objects in this data set. A nice feature of the SPNS fitting is that the die-away times do not really change as a function of mass. This agrees with the time results seen from the pulsed histogram analysis. The die-away SPNS fitting parameters are believed to be indicative of the moderation around the material as well as neutrons returning from the detector or the room, and so these values should not substantially change due to an increase in the mass of material. Added mass does bring changes in each of the other measurement object parameters. These are where the SPNS model would expect mass changes to manifest, and it is possible that these parameters could each be fitting against a mass calibration curve to determine the nuclear material of an object. Again though it would likely make the most sense to do calibration curve fitting with the shift register analysis and use SPNS parameters for other sample characteristic determinations.

Table XVI. SPNS fitting parameters from the mass study

	N/pulse	Fast Thermal Fission Prob.	Slow Thermal Fission Prob.	Fast Die-Away	Slow Die-Away	Steady-State Rate
1DU	30000	0.0004	0.0001	120	240	5800
2DU	37000	0.0004	0.0001	120	240	9500
1HEU	34000	0.001	0.0002	80	430	4000
2HEU	35000	0.0015	0.0003	80	430	7500
1DU+p3	24000	0.00045	0.000006	180	360	1000
2DU+p3	32000	0.00051	0.00003	180	360	1500
1HEU+p3	25000	0.0037	0.0009	100	200	1300
2HEU+p3	32500	0.0083	0.0015	105	195	3300

There are also some notable trends with this data set and the first data set in the SPNS parameters. SPNS neutrons per pulse and steady-state rate fitting parameters both decrease whenever moderation is added, but the fission probabilities tend to increase. In addition, the die-away times for the samples from the fast thermalization all increase whenever polyethylene is added. The slow die-away time value decreases in both the DU and the HEU samples as moderation is added. This effect may be due to the addition of a moderator since the moderator will absorb neutrons that might have previously returned to the sample after being thermalized by something in the room.

VI.C. Moderation Analysis

The data used as part of the moderation study was also taken during the June 2012 measurement campaign. This data used both of the 272g HEU plates with 93.27% U-235 enrichment for each measurement. The base case has both plates placed back to back with no moderation present. Again for all of these runs the plates are centered in the LINAC beam line using the lasers provided for alignment. Additionally the widest face of the plate always is facing the LINAC. After the no moderation case, 1 inch of polyethylene moderation was added on all sides of the HEU, including in between the two plates. All cases past this first moderation case add 1 inch of polyethylene on all sides of the material in the plate of the detector but not in between the sample stand and the material or on top of the material. Table XVII contains the sample descriptions paired with their experimental data designators.

Table XVII. Sample descriptions for the moderation data set

Name	Physical Description
2HEU	2 Highly Enriched Uranium plates (544g)
2HEU+p1	2 Highly Enriched Uranium plates with 1" side poly/ 1" between, top, & bottom
2HEU+p2	2 Highly Enriched Uranium plates with 2" side poly/ 1" between, top, & bottom
2HEU+p3	2 Highly Enriched Uranium plates with 3" side poly/ 1" between, top, & bottom

The results of the shift register analysis on the moderation data set are included in Table XVIII. The analysis shows the highest doubles and singles count rates for the sample surrounded by 2 inches of polyethylene moderation. This indicates that 2 inches of polyethylene moderation is the closest to an optimal moderation configuration. This also indicates that 3 inches of polyethylene moderation over moderates the HEU measurement object, which explains the decreased singles and doubles count rates. The first moderation case with 1 inch of polyethylene then under moderates the HEU measurement object. The D/S values also show an increase in multiplication throughout the series as polyethylene is added until the 3rd inch is placed.

Table XVIII. Shift register analysis of the moderation data

	S	σ_S	D	σ_D	D/S	$\sigma_{D/S}$
2HEU	745	18	129	11	-0.21	0.02
2HEU+p1	884	12	454	15	0.04	0.02
2HEU+p2	3380	71	5467	213	1.06	0.07
2HEU+p3	1329	24	1935	53	0.82	0.04

The pulsed histogram double analysis results are included in Table XIX, and the D/S results are included in Table XX. The max bin value results from both of these tables support the conclusions from the shift register analysis about the moderation state of each of the objects. The time average for the peaks largely behave as expected in both the doubles pulsed histogram and the D/S pulsed histogram by increasing as polyethylene is added until reaching optimal moderation and then decreasing due to moderator absorption rate overcoming the increased rate of fission in longer time periods in the 3 inch case.

Table XIX. Doubles pulsed histogram analysis results for the moderation data

	2HEU	2HEU+p1	2HEU+p2	2HEU+p3
Max	975	3098	42241	12318
σ_{Max}	80	165	1699	297
Time (μs)	308	300	332	320
σ_{time} (μs)	4	4	4	8

The decrease in the peak time between the no moderation sample and the 1 inch of moderation sample in both of the histograms was not expected but could be due to the polyethylene absorption of thermal neutron return from the room having a more significant effect in later time domains than the thermalization of neutrons from the measurement object. The overall rate of fission still would increase, as seen due to the increase in the max value, but the decrease in later time domains would explain the slight reduction in the time average since it would lower the correlated neutron signal in this time domain. This is quite possible because there were some gaps between the

polyethylene moderation that could have allowed neutrons to escape easily with this moderation configuration, thus reducing the population of neutrons that was moderated by the polyethylene more than would be expected. This chance was significantly lessened as more polyethylene blocks were added to the outside of the measurement object in the subsequent data runs since those blocks tended to cover the cracks from the previous blocks. No matter the cause, the values are within 2 sigma of each other, so it is a fairly minor effect.

Table XX. D/S pulsed histogram analysis results for the moderation data

	2HEU	2HEU+p1	2HEU+p2	2HEU+p3
Max	0.68	1.69	7.96	4.37
σ_{Max}	0.31	0.30	0.49	0.33
Time (μs)	328	324	340	336
σ_{time} (μs)	8	4	4	8

The SPNS fitting parameters for the measurement objects are included in Table XXI. As seen in the other data sets, the steady-state rate tends to fall as polyethylene is added to a sample, although this effect is not seen as 1 inch of polyethylene is added to the bare sample. In this case the additional polyethylene added does not absorb enough neutrons past 1000 μs to overcome the increase in the delayed neutrons from more thermal fissions. This effect could have been amplified by neutron leakage out of gaps in the moderator. The steady-state rate then begins to decrease as another inch of polyethylene is added because the polyethylene absorption is enough to reduce the neutrons in the quasi-steady-state region.

Another unpredicted effect is that the neutrons per pulse in the 3 inch case are greater than the neutrons per pulse from the 2 inch case. This is most likely due to the fact that these samples were not least squares fit and were instead fit by eye. The neutrons per pulse parameter is one of the more difficult to fit with SPNS since SPNS does not accurately model the noise effects from the gamma flash, and effects from the gamma flash can still be dominant in the time domain most impacted by the neutrons per pulse parameter. Another interesting effect is that the fast die-away time increases dramatically from the 1 inch case to the 2 inch case, but then decreases in the 3 inch case. This may be a result of the polyethylene absorbing the longer lived neutrons out of the system so that they are not detected, and it is supported by the appearance of a similar effect in the time average data. The fast fission probability behaves as expected for these fits by gradually increasing through the optimally moderated case before decreasing as the sample becomes over-moderated.

Table XXI: SPNS fitting parameters for the moderation data set

	N/pulse	Fast Thermal Fission Prob.	Slow Thermal Fission Prob.	Fast Die-Away	Slow Die-Away	Steady-State Rate
2HEU	35000	0.0015	0.0003	80	430	7500
2HEU+p1	40000	0.0025	0.0001	80	430	9000
2HEU+p2	30000	0.013	0.0002	220	420	3500
2HEU+p3	32500	0.0083	0.0015	105	195	3300

VI.D. Inspection Object Analysis

This data set contains the test objects that were the focus of the US-UK measurement campaign. Their construction is covered in detail in [8]. As part of the verification exercise, each object was referred to as an "inspection object" followed by a number to identify the object. All of the inspection objects were hidden inside equally sized aluminum boxes so that scientists involved in the exercise could not see the material making up the object. Scientists were allowed to add a 2 inch polyethylene box as moderation to the exterior of each inspection object as part of the exercise. For this section of the thesis, only the PSR analysis method, including the shift register analysis and the pulsed histogram analysis, was applied since it is the more mature of the two tools. A table of general descriptions of all the inspection objects is included in Table XXII.

Table XXII. Inspection object physical descriptions

Name	Physical Description
IO5	4.8kg of HEU shielded by 6.2kg DU and 11 kg LiH
IO5+P	IO5 with a 2 inch polyethylene box added
IO6	5.5kg of DU with 5.7kg W and 11kg LiH shielding
IO6+P	IO6 with a 2 inch polyethylene box added
IO7	5.2kg of HEU shielded by 17.3 kg DU
IO7+P	IO7 with a 2 inch polyethylene box added
IO8	1.6kg of PuAl shielded by 5.6kg DU and 0.04kg W
IO8+P	IO8 with a 2 inch polyethylene box added
IO9	1.6kg of PuAl with 4.8 kg of HEU shielding
IO9+P	IO9 with a 2 inch polyethylene box added
IO10	1.6kg of PuAl with 5.3kg of poly, 0.04 kg of W, and 5.7kg of DU shielding
IO10+P	IO10 with a 2 inch polyethylene box added

Table XXIII contains the shift register analysis of the IOs. From these rates relative to the standards that were previously measured, the objects that clearly contain nuclear material are IO7, IO8, IO9, and IO10. IO5 appears to have similar behavior to depleted uranium from the standards and other measurements, and so it is unlikely that this would flag as a multiplying object with the shift register analysis technique. IO6 would also correctly register as not having multiplying material in it. IO7, IO8, and IO9 would all flag as having multiplying material in them based on the significant increase in the doubles count rate in the objects as the polyethylene was added. IO10 would also

likely flag as having multiplying material because of the large doubles count rate before any moderation was added.

There is no way of determining the mass of multiplying material in the various IOs without having other similar samples of varying mass. It is possible to say something about the moderation of the IOs as the polyethylene box is added to each one. IO5, IO6, and IO10 appear to be over-moderated before the addition of the polyethylene box while the other samples appear to be under-moderated before the addition of the polyethylene box.

Table XXIII. Shift register analysis of the inspection objects

	S	σ_S	D	σ_D	D/S	$\sigma_{D/S}$
IO5	2536	27	169	7	-0.095	0.008
IO5+p	915	10	43	3	-0.095	0.008
IO6	323	6	25	3	-0.051	0.008
IO6+p	84	4	4	3	-0.028	0.009
IO7	4853	67	403	22	-0.085	0.009
IO7+p	7573	96	15049	300	1.747	0.045
IO8	3117	18	221	16	-0.093	0.009
IO8+p	3863	38	3321	81	0.639	0.022
IO9	3517	31	385	9	-0.059	0.008
IO9+p	9422	95	28659	540	2.783	0.063
IO10	3513	23	8359	111	2.038	0.033
IO10+p	2014	21	2809	54	1.068	0.027

The PSR histogram analysis for doubles pulsed histograms and for D/S pulsed histograms is included in Table XXIV. The pulsed histogram analysis would also identify IO5 and IO6 as non-multiplying measurement objects and IO7, IO8, IO9, and

IO10 as multiplying measurement objects based on the max values of both the doubles and D/S pulsed histograms. Both the max value and the peak time average values support the shift register analysis conclusions that IO5, IO6, and IO10 are over moderated while the other samples are under moderated. The pulsed histogram analysis also could not indicate anything about the samples mass since representative measurements of these objects to create a calibration curve were not taken.

Table XXIV. Pulsed histogram analysis of the inspection objects

	Doubles				**D/S**			
	Max	σ_{Max}	*Time (μs)*	σ_{time} *(μs)*	*Max*	σ_{Max}	*Time (μs)*	σ_{time} *(μs)*
IO5	773	69	304	8	0.82	0.16	328	8
IO5+P	294	32	304	8	0.38	0.18	320	8
IO6	159	25	296	8	0.28	0.20	320	8
IO6+P	62	25	296	8	0.14	0.20	328	8
IO7	2170	87	308	4	1.78	0.17	328	8
IO7+P	107482	2932	332	4	14.13	0.51	340	4
IO8	1022	55	308	4	1.06	0.14	328	8
IO8+P	23042	692	332	4	6.30	0.27	340	4
IO9	1840	92	316	4	1.64	0.20	332	4
IO9+P	199663	4666	332	4	19.57	0.56	340	4
IO10	43062	599	320	8	10.24	0.32	332	4
IO10+P	16210	464	324	4	5.77	0.22	332	4

CHAPTER VII

SUMMARY AND CONCLUSIONS

The treaty verification field is of renewed importance as continued nuclear weapons disarmament is prioritized nationally in partnership with other nuclear weapons states. The renewed interest in the field has led to research and development on technologies that could support future US verification missions including a technology that employs pulsed high-energy photons to interrogate measurement objects, and examines the neutron signal from this interrogation to determine characteristics of the object. An assessment of this technique was conducted to determine if analysis tools could be written to utilize information from the pulsed region of the neutron data to identify objects containing nuclear material, to find the mass of the nuclear material, and to determine the moderation state of the material.

This assessment has shown that the analysis tools developed for this interrogation technique are capable of determining whether or not a measurement object contains multiplying material in all cases except for when that object is very heavily shielded. The analysis tools are also able to determine the moderation state of the object, and could be used to determine the mass if the results from the analysis were calibrated to known standards representative of the expected measurement objects.

Both the shift register technique and the pulsed histogram analysis techniques provide adequate information to make determinations for all the categories of interest. The shift register technique generally has better statistics because it utilizes all

measurement data outside of the window to compute values representative of the sample whereas the analysis of the pulsed histograms utilizes values either in just one bin or the pulsed region alone. It is highly unlikely that the pulsed histogram plots would be made available to any weapons inspectors in a treaty verification regime due to classification issues, so the analysis methods employed with this data serves as an information barrier for the technique, and makes it easy for alarm levels to be set in any future equipment deployed for an actual treaty verification regime.

The SPNS tool shows promise as a way to distinguish between different types of nuclear material since the fitting is sensitive to the multiplicity distribution of the object. However, the distinction between types of nuclear material could usually be made through a passive measurement of a verification object since plutonium should passively emit a detectable amount of radiation, and so a tool of SPNS complexity may not be needed to be coupled with this technique. SPNS fitting could potentially distinguish between the declared measurement object and a spoof in a verification regime, although more development would be needed to explore this capability.

In order for SPNS to be useful for field implementation in an actual verification campaign, sensitivity studies need to be performed to show the resolution of the fitting parameters, an automated fitter needs to be written for the program, and both the automated fitter and SPNS need to be configured to utilize parallel processing. The sensitivity studies are needed before a fitter can be designed because it will be important to understand the magnitude of change each SPNS parameter creates whenever it is modified. This can help an automated fitter more efficiently fit SPNS parameters to a

measurement object, and would also help determine the error associated with the SPNS parameters.

Designing an automated fitter is rather simple for the case of fitting singles pulsed histograms since least squares fitting has already been used to find a unique parameter set for one measurement object, and the SPNS fitting parameters are mostly independent of each other. However, the ability to fit doubles pulsed histograms by modifying the induced fission multiplicity distribution and remains the biggest obstacle to implementing a fitting algorithm. If the process of fitting the multiplicity histogram could be automated, then the final result of that fit could serve as a method to identify the material within the measurement object since most objects have different rates of neutron production from fission. Additionally, the combination of an automated fitter with SPNS could become computationally expensive since SPNS is presently written to only run on one processor and the program would need to be run multiple times for one fitting, so the programs would also need to be modified to operate in parallel.

Other ways to improve this measurement technique could be made through further development of the electronics, specifically by characterizing the He-3 detectors behavior during a gamma flash. Most of the doubles neutrons produced by measurement objects in the pulsed photon environment occur around and immediately after the pulse, and so finding methods of improving the He-3 tubes recovery from the flash, or enabling the recovery of neutron signals during the gamma flash would allow for more accurate doubles results. At present the PND detectors do give reliable results for shift register analysis when a window is employed. Additionally many of the electronic noise issues

present in the data during the US-UK measurement campaign did not manifest themselves in the June 2012 measurement campaign data.

REFERENCES

1. J. Doyle, "Verification Challenges for Future Nuclear Weapons Reductions," *Proceedings of the 51st Annual Meeting for the Institute of Nuclear Materials Management*, Baltimore, MD, July 11-15, 2010.

2. D.R. Norman, J.L. Jones, K.J. Haskell, P.E. Vanier, and L. Forman, "Active Nuclear Material Detection and Imaging," *2005 IEEE Nuclear Science Symposium Conference Record*, pp. 1004-1008. 2005.

3. M.W. Johnson, J.E. Doyle, and C.L. Murphy, "Recovering START Institutional Knowledge," *Proceedings of the 52nd Annual Meeting for the Institute of Nuclear Materials Management*, Palm Desert, CA, July 17-21, 2011.

4. D.K. Hauck, D.W. MacArthur, M.C. Browne, and R.F. Parker, "The Role of Portal Monitors in Arms Control and Development Needs," *Proceedings of the 53rd Annual Meeting of the Institute of Nuclear Materials Management*, Orlando, FL, 15-19 July, 2012.

5. J.T. Mihalczo, J.A. Mullens, J.K. Mattingly, and T.E. Valentine, "Physical Description of Nuclear Materials Identification System (NMIS) Signatures," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors, and Associated Equipment,* Vol. 450, pp. 531-555, 2000.

6. E.D. Sword and S.M. McConchie, "Edge Detection and Shape Recognition in Neutron Transmission Images," *Proceedings of the 53rd Annual Meeting of the Institute of Nuclear Materials Management*, Orlando, FL, 15-19 July, 2012.

7. D. Norman, J. Jones, K. Haskell, R. Watson, L. Montierth, J. Zabriskie, W. Geist, J Thron, C. Freeman, M. Swinhoe, S. McConchie, and E. Sword, "Pulsed, Photonuclear-Induced, Neutron Measurements of Nuclear Materials with Composite Shielding," *Proceedings of the 52nd Annual Meeting for the Institute of Nuclear Materials Management*, Palm Desert, CA, July 17-21, 2011.

8. R. Neibert, J. Zabriskie, C. Knight, and J.L. Jones, "Passive and Active Radiation Measurements Capability at the INL Zero Power Physics Reactor (ZPPR) Facility," *INL/EXT-11-20876*, Idaho National Laboratory, Dec 2010.

9. G.F. Knoll, *Radiation Detection and Measurement*, Fourth Edition, John Wiley & Sons, Inc., River Street, Hoboken, NJ, 2010.

10. M. King, T. Gozani, J. Stevenson, and T. Shaw, "Simulation of a Photofission Based Cargo Interrogation System," *AIP Conference Proceedings,* 1336, pp. 700-704, 2011.

11. T. Gozani, "Fission Signatures for Nuclear Material Detection," *IEEE Transactions on Nuclear Science*, Vol. 56, No. 3, June 2009.

12. J.L. Jones, D.R. Norman, K.J. Haskell, M.T. Swinhoe, S.J. Tobin, W.H. Geist, R.B. Rothrock, and C.R. Freeman, "Coincidence /Multiplicity Photofission Measurements," *INL/EXT-09-16971,* Idaho National Laboratory, Sep. 2009.

13. J.L. Jones, B.D. Bennett, K.J. Haskell, J.T. Johnson, D.R. Norman, J.W. Sterbentz, R.W. Watson, S.M. Watson, W.Y. Yoon, J.M. Zabriskie, C.E. Moss, K.L. Folkman, A.W. Hunt, C.C. O'Neil, and R.J. Spaulding, "Pulsed Photonuclear Assessment (PPA) Technique: CY-05 Project Summary Report," *INL/EXT-05-01020,* Idaho National Laboratory, Dec. 2005.

14. M.T. Swinhoe, J.B. Marlow, and H.O. Menlove, "Neutron List Mode Data for Advanced Safeguards," *Proceedings of the Global 2007 Meeting of the American Nuclear Society,* Boise, Idaho, July 2007.

15. N. Ensslin, W.C. Harker, M.S. Krick, D.G. Langner, M.M. Pickrell, and J.E. Stewart, "Application Guide to Neutron Multiplicity Counting," *LA-13422-M*, Los Alamos National Laboratory, Nov 1998.

16. D. Reilly, N. Ensslin, H. Smith, and S. Kreiner, *Passive Nondestructive Assay of Nuclear Materials*, United States Nuclear Regulatory Commission, Washington, DC, 1991.

17. P.M.J. Chard and S. Croft, "Preliminary Investigation of Active Neutron Coincidence Counting in Differential Die-Away Assay," *Proceedings of the 7th International Conference on Radioactive Waste Management and Environmental Remediation*, Sept. 1999.

18. S.L. Stewart, M.T. Swinhoe, J.L. Thron, W.H. Geist, and W.S. Charlton, "Monte Carlo Modeling of Correlations in Pulsed Neutron Data," *Proceedings of the 53rd Annual Meeting of the Institute of Nuclear Materials Management*, Orlando, FL, 15-19 July, 2012.

19. W. H. Press, *Numerical Recipes in C: The Art of Scientific Computing*, Second Edition, Cambridge University Press, Cambridge, NY, 1992.

20. MCNP X-5 Monte Carlo Team, "MCNP—A General Purpose Monte Carlo NParticle Transport Code, Version 5," *LA UR 03 1987*, Los Alamos National Laboratory, April 2003.

APPENDIX A

PSR pulsed histogram graphs for each data set.

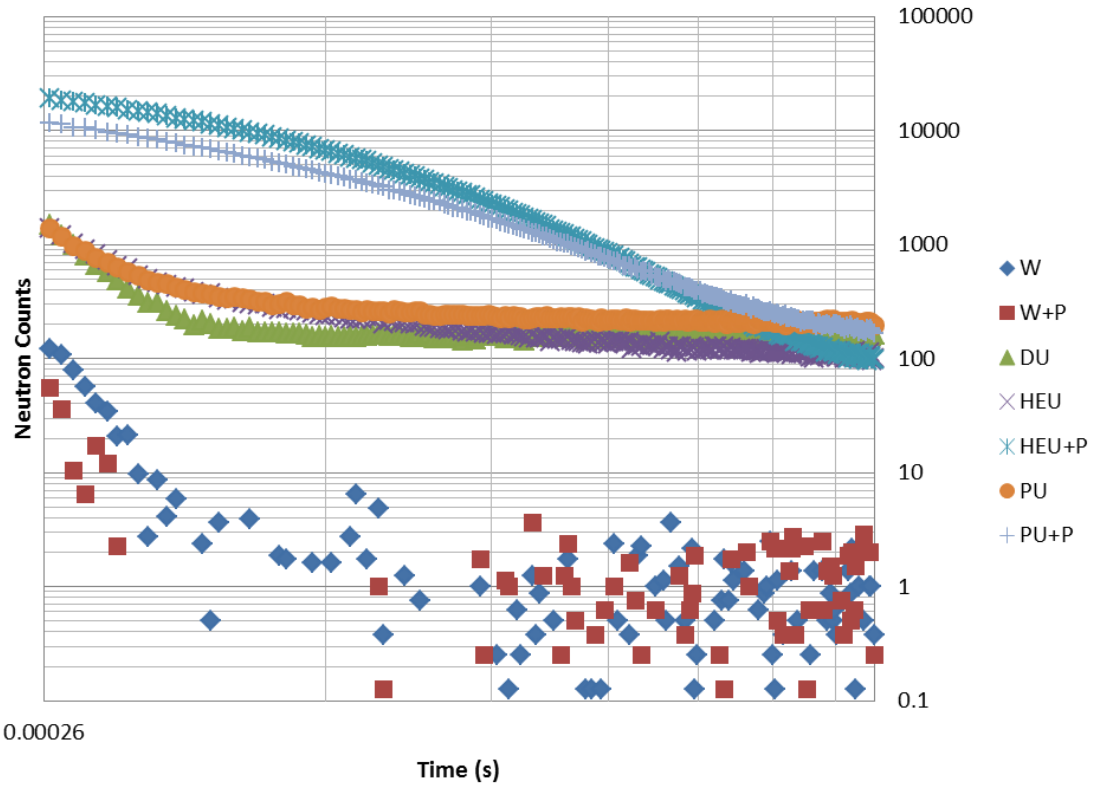

Figure A.1. Background subtracted singles pulsed histograms for the fissile content analysis

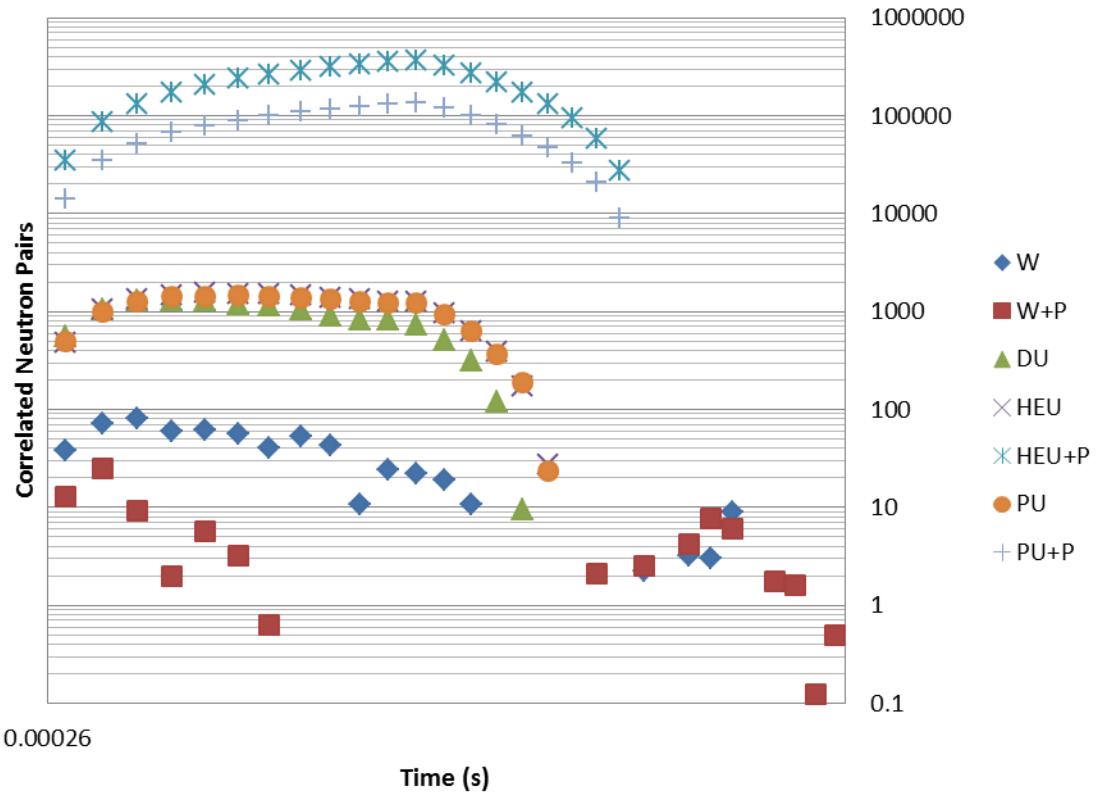

Figure A.2. Background subtracted doubles pulsed histograms for the fissile content analysis

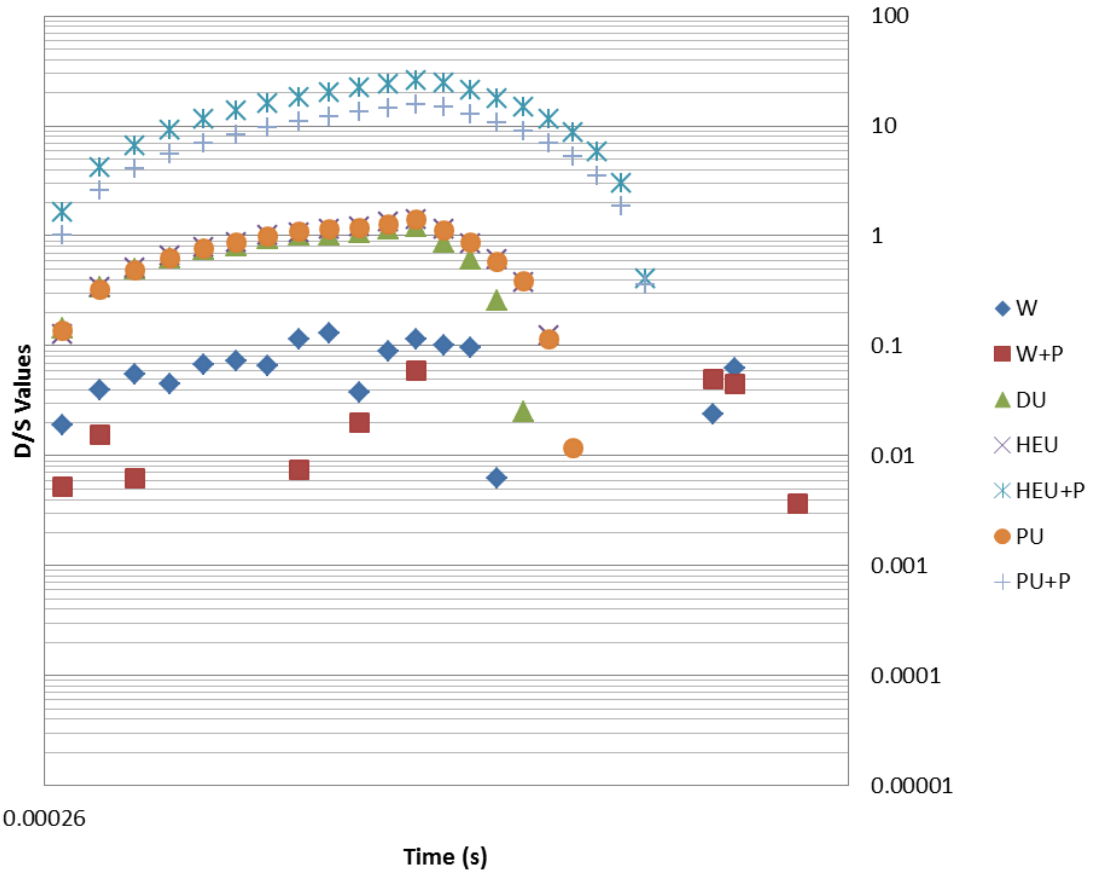

Figure A.3. Background subtracted D/S pulsed histograms for the fissile content analysis

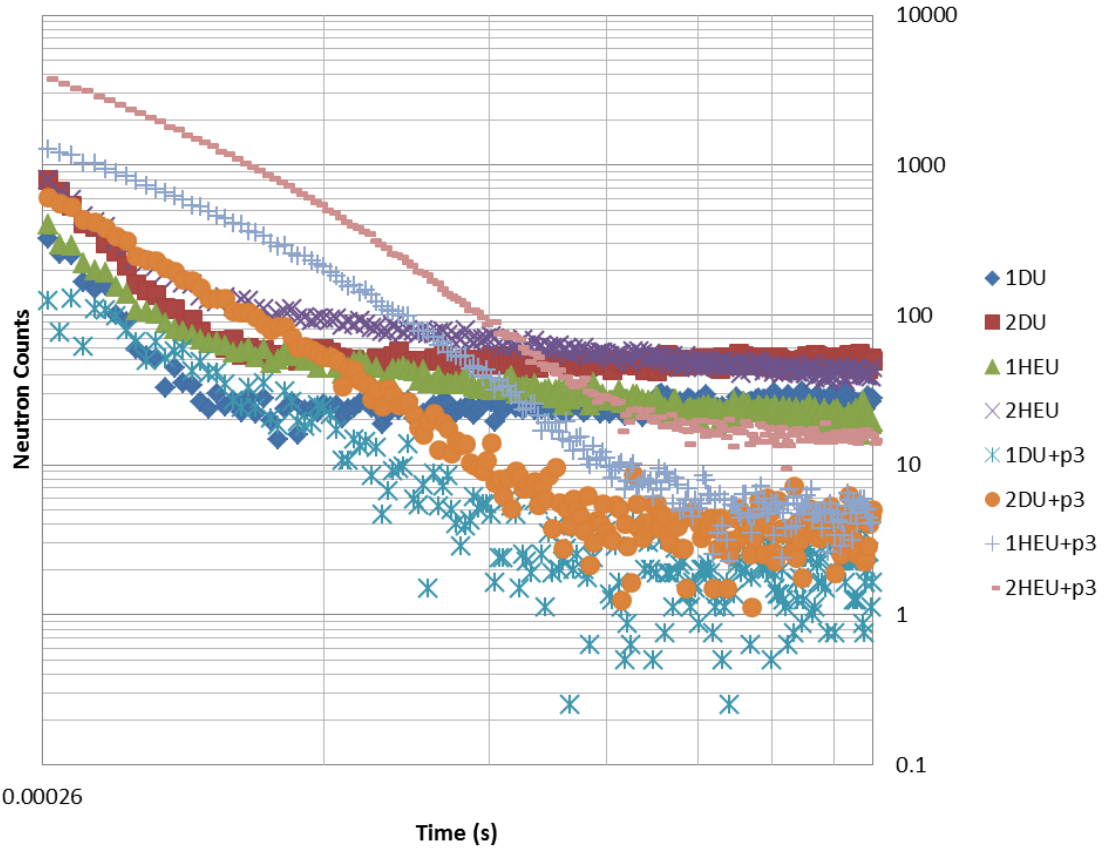

Figure A.4. Background subtracted singles pulsed histograms for the mass analysis

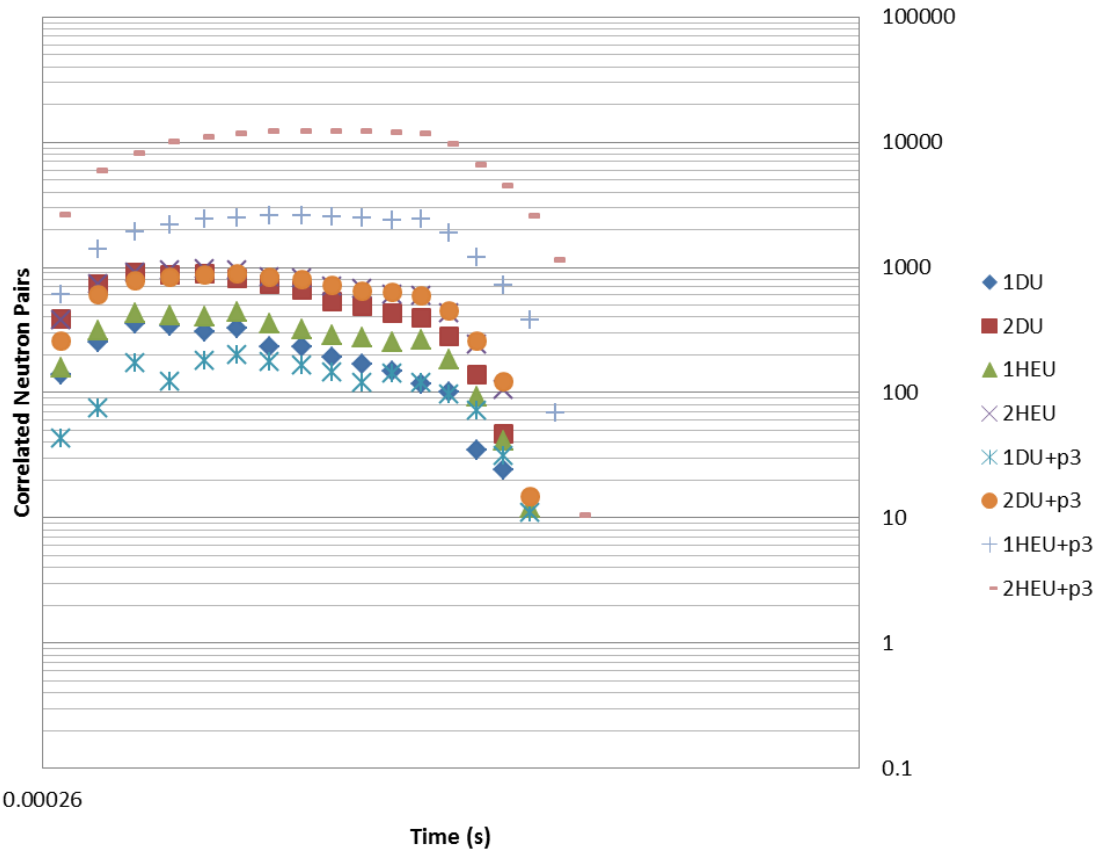

Figure A.5. Background subtracted doubles pulsed histograms for the fissile content analysis

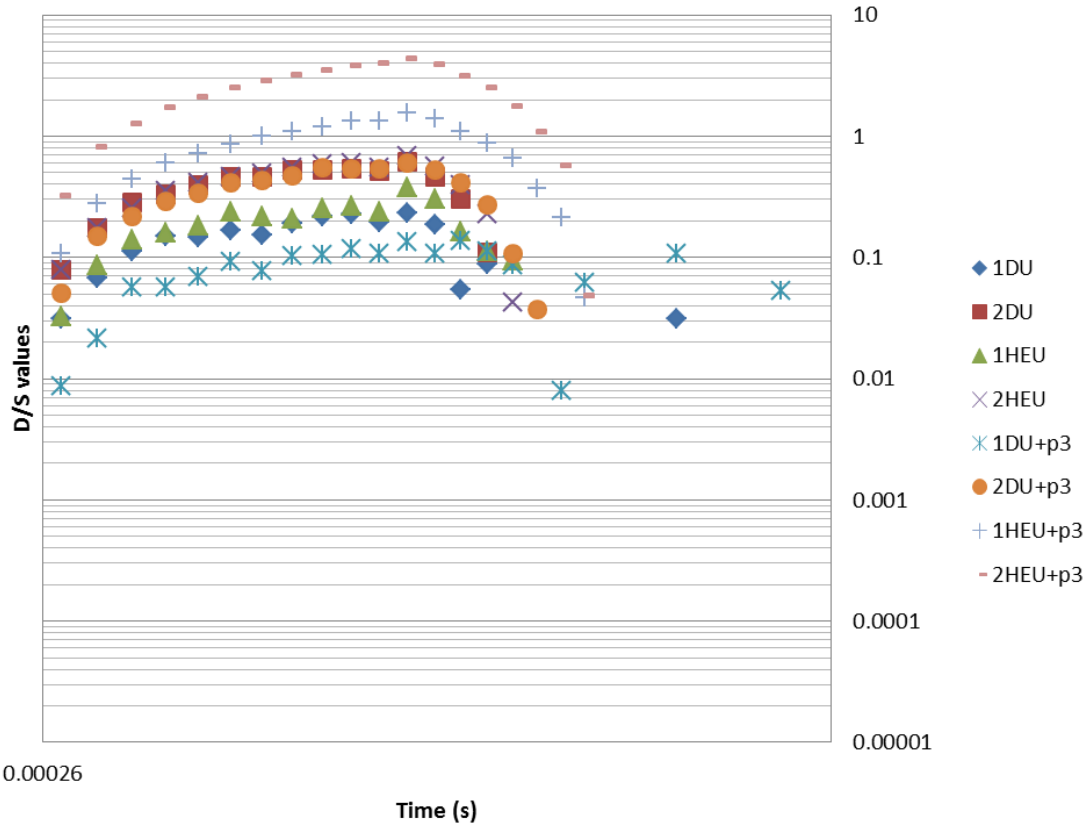

Figure A.6. Background subtracted D/S pulsed histograms for the mass analysis

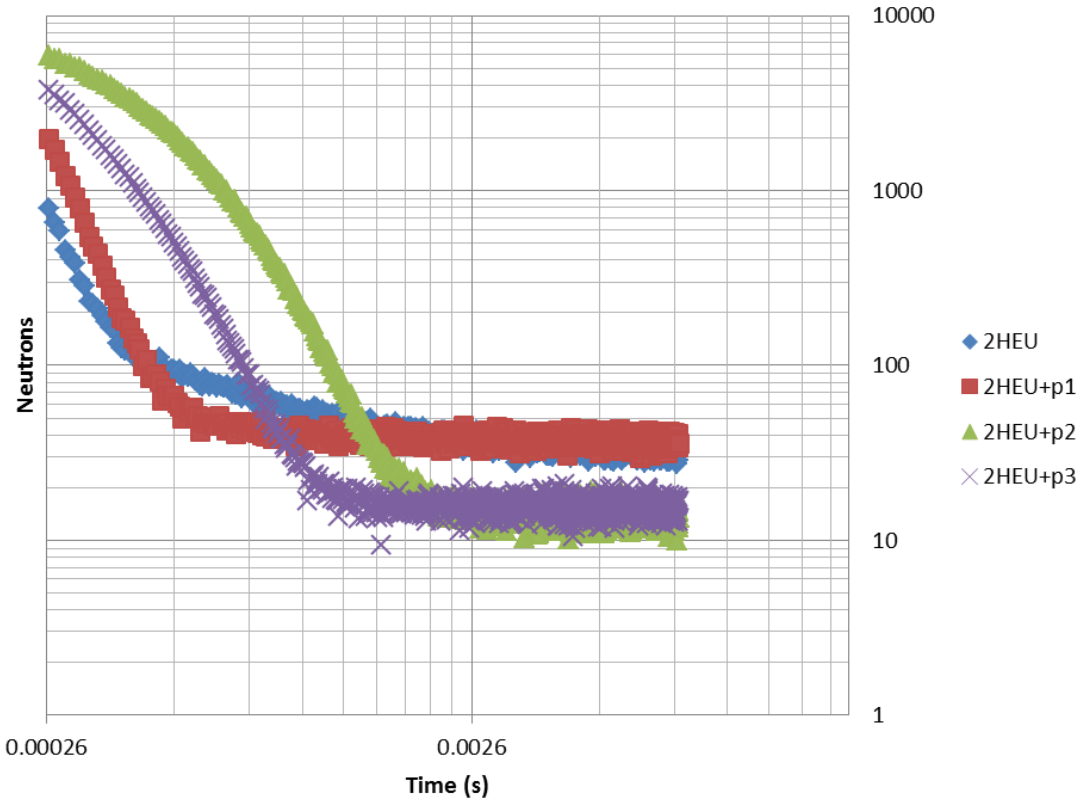

Figure A.7. Background subtracted singles pulsed histograms for the moderation analysis

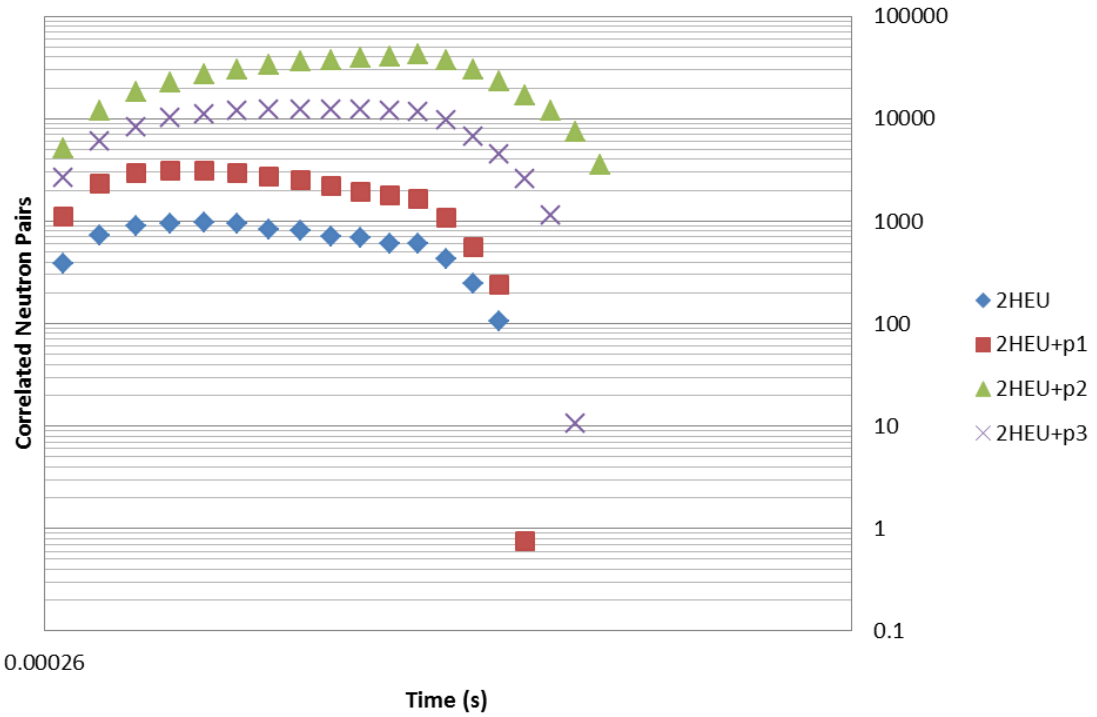

Figure A.8. Background subtracted doubles pulsed histograms for the moderation analysis

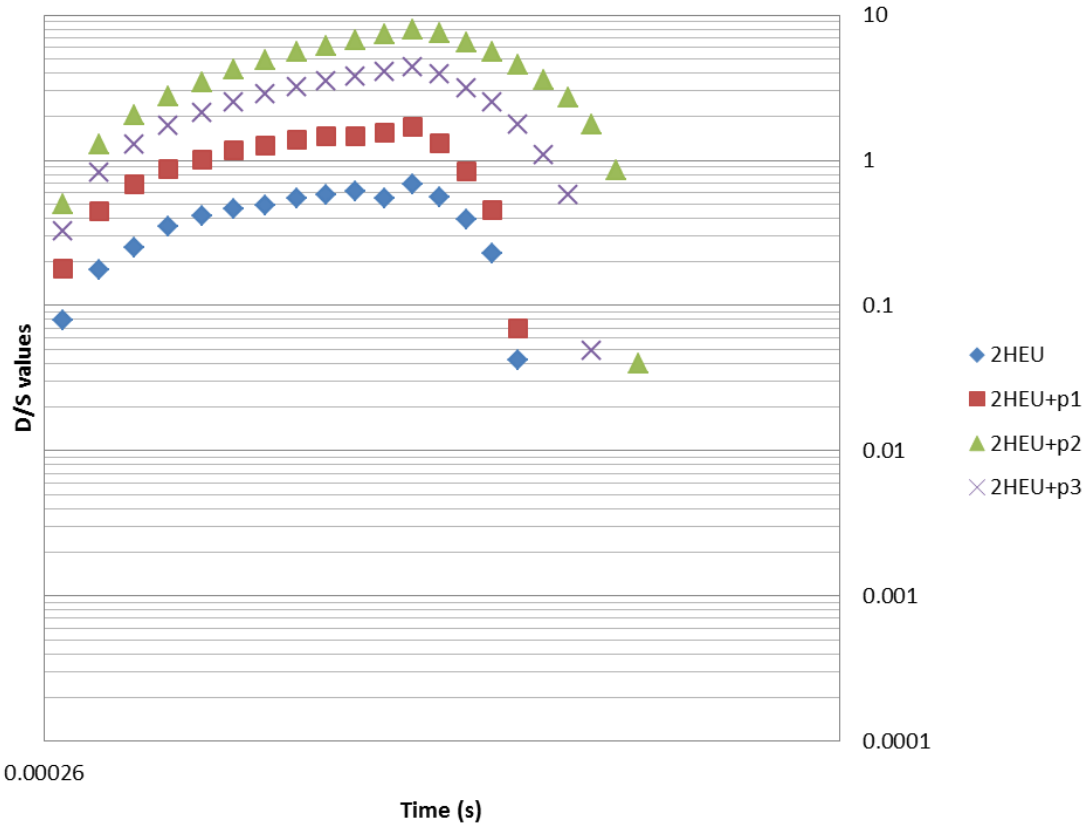

Figure A.9. Background subtracted D/S pulsed histograms for the moderation analysis

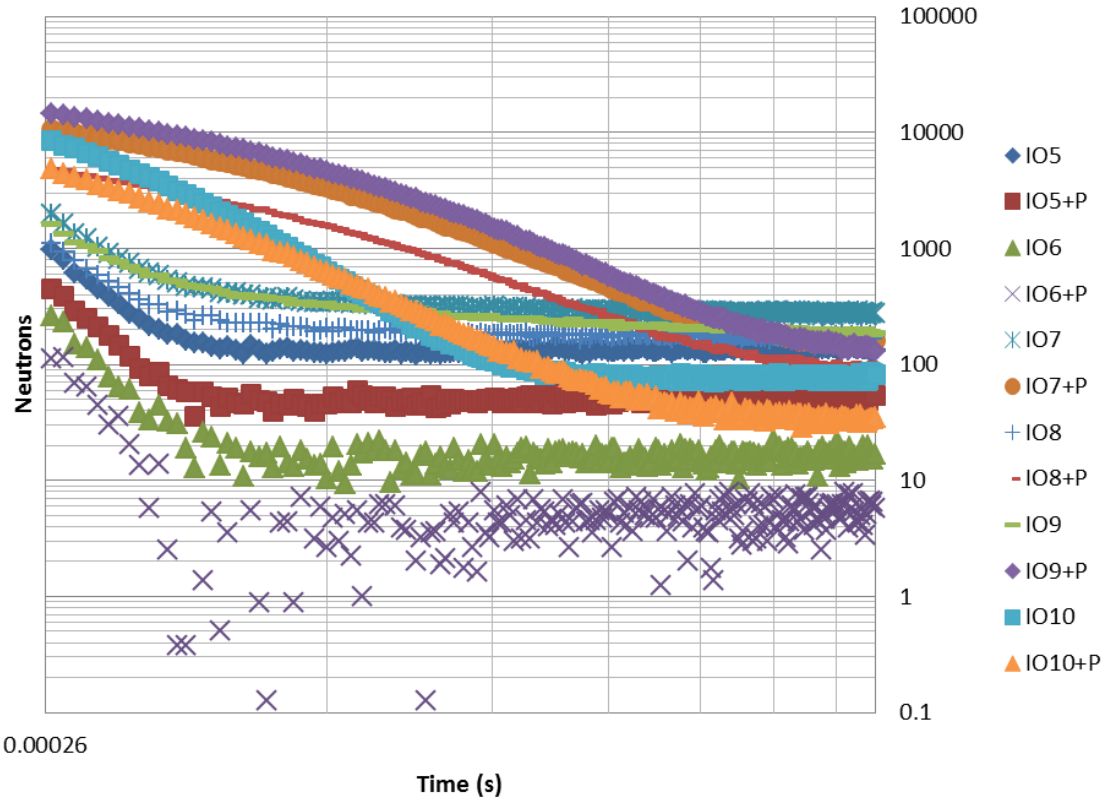

Figure A.10. Background subtracted singles pulsed histograms for the IO analysis

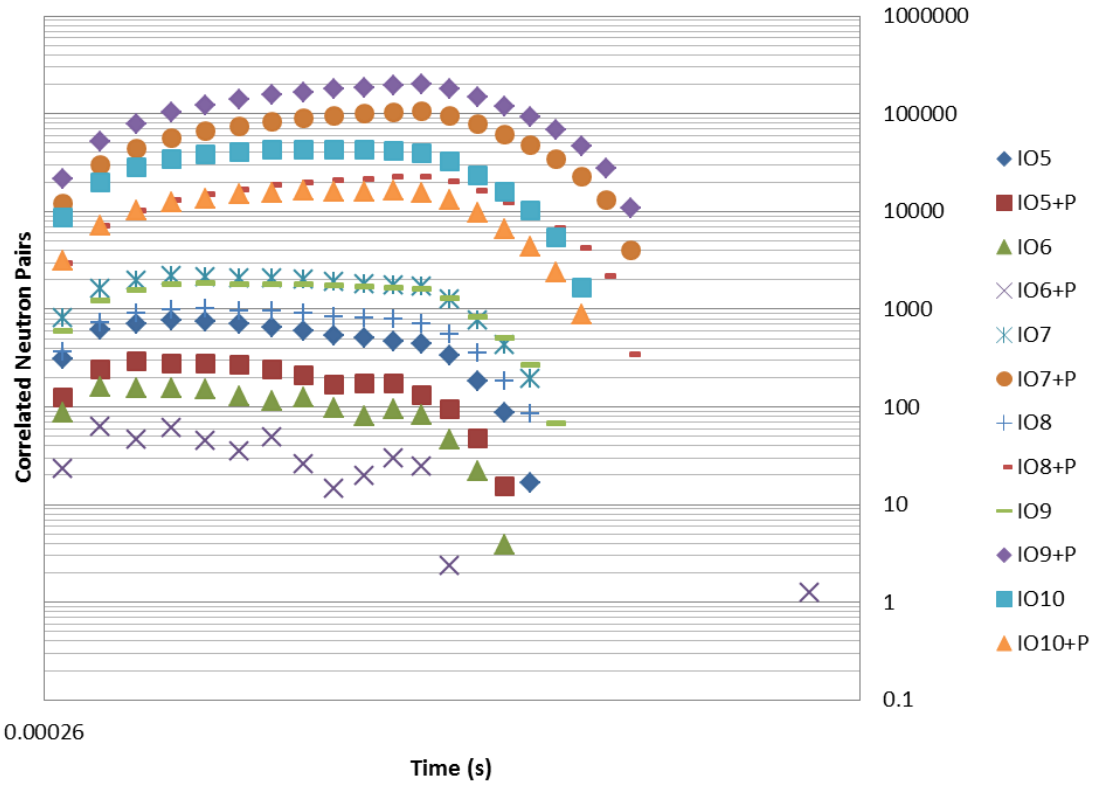

Figure A.11. Background subtracted doubles pulsed histograms for the IO analysis

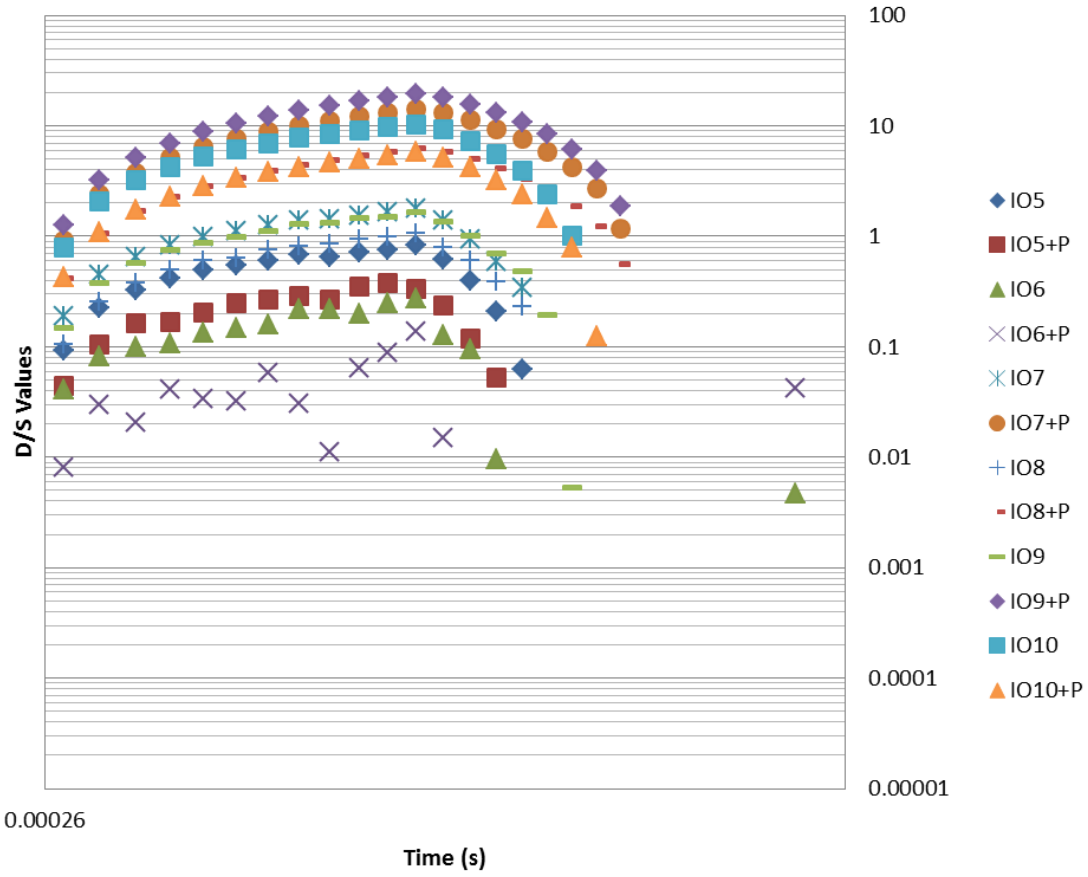

Figure A.12. Background subtracted D/S pulsed histograms for the IO analysis

APPENDIX B

SPNS Fitting of Singles Pulsed Histograms

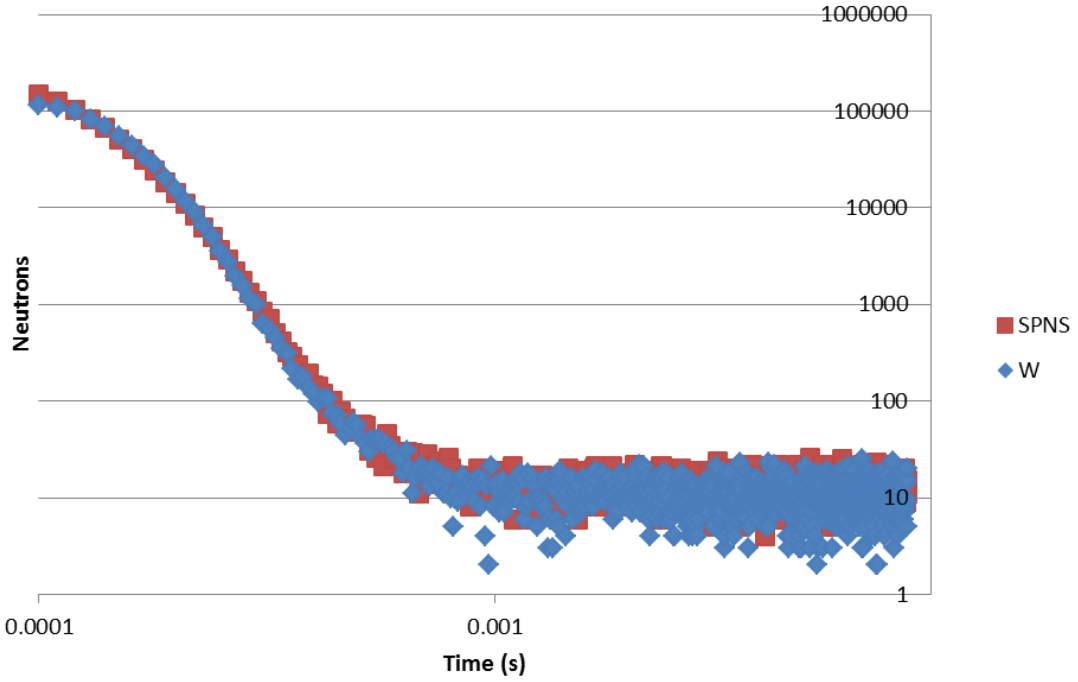

Figure B.1. SPNS Fitting of W

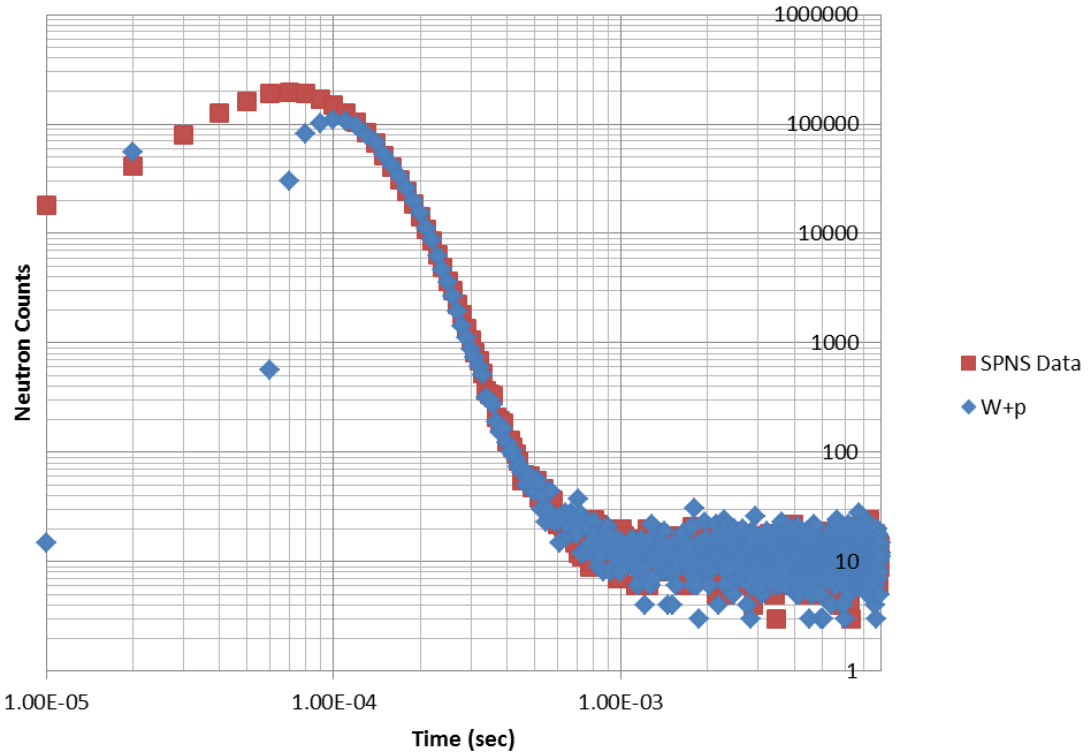

Figure B.2. SPNS Fitting of W+p

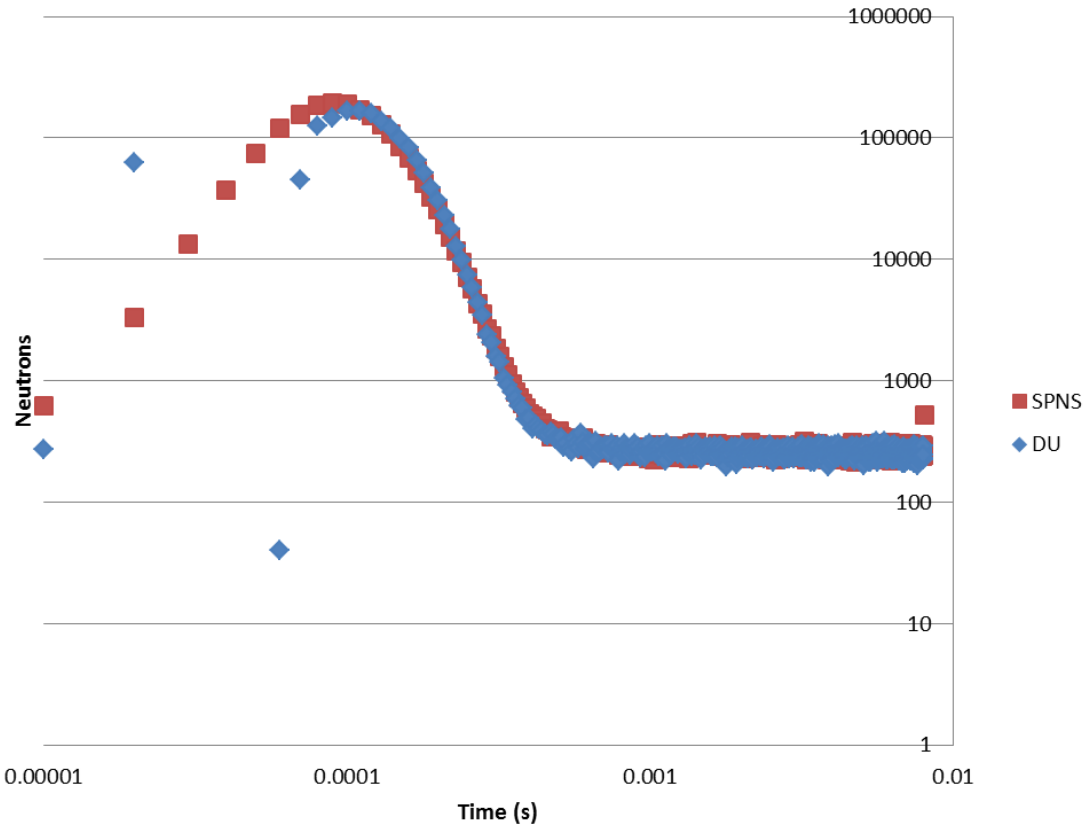

Figure B.3. SPNS Fitting of DU

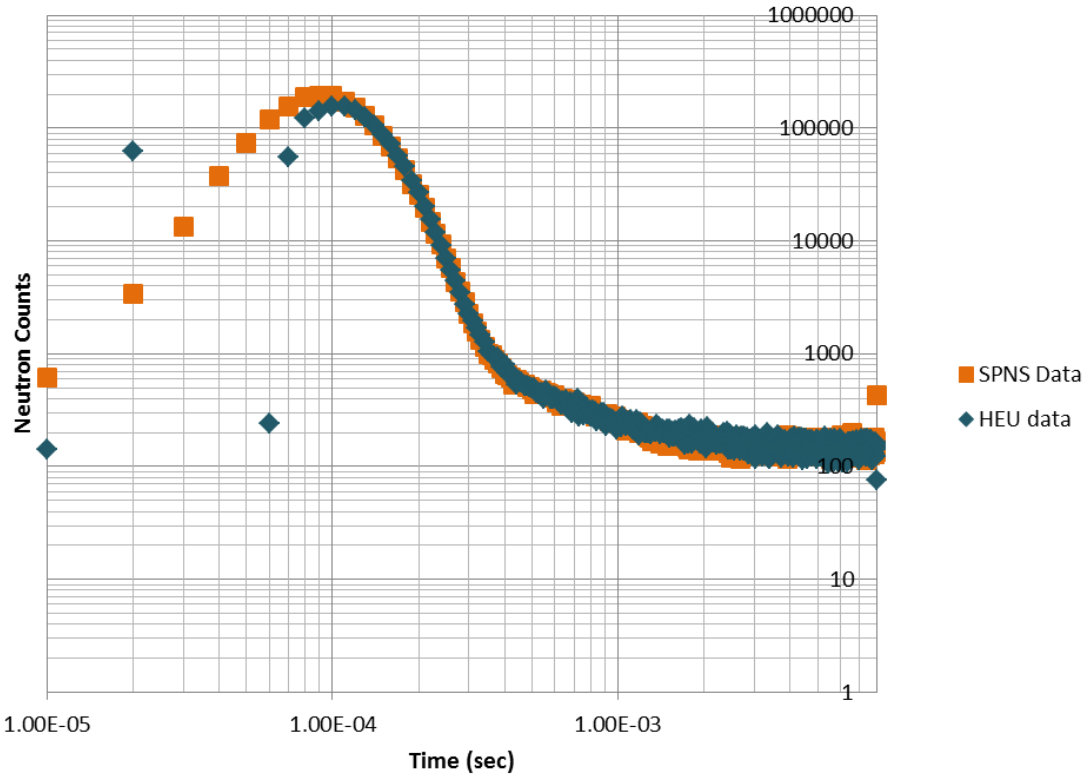

Figure B.4. SPNS Fitting of HEU

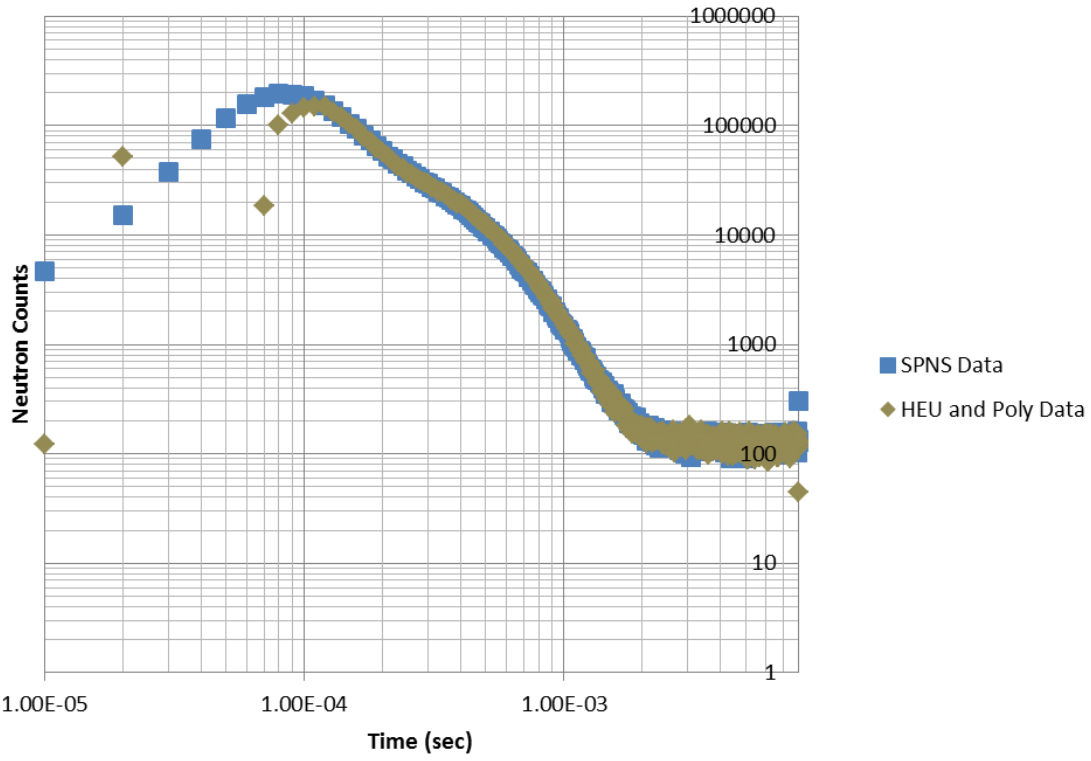

Figure B.5. SPNS Fitting of HEU+p

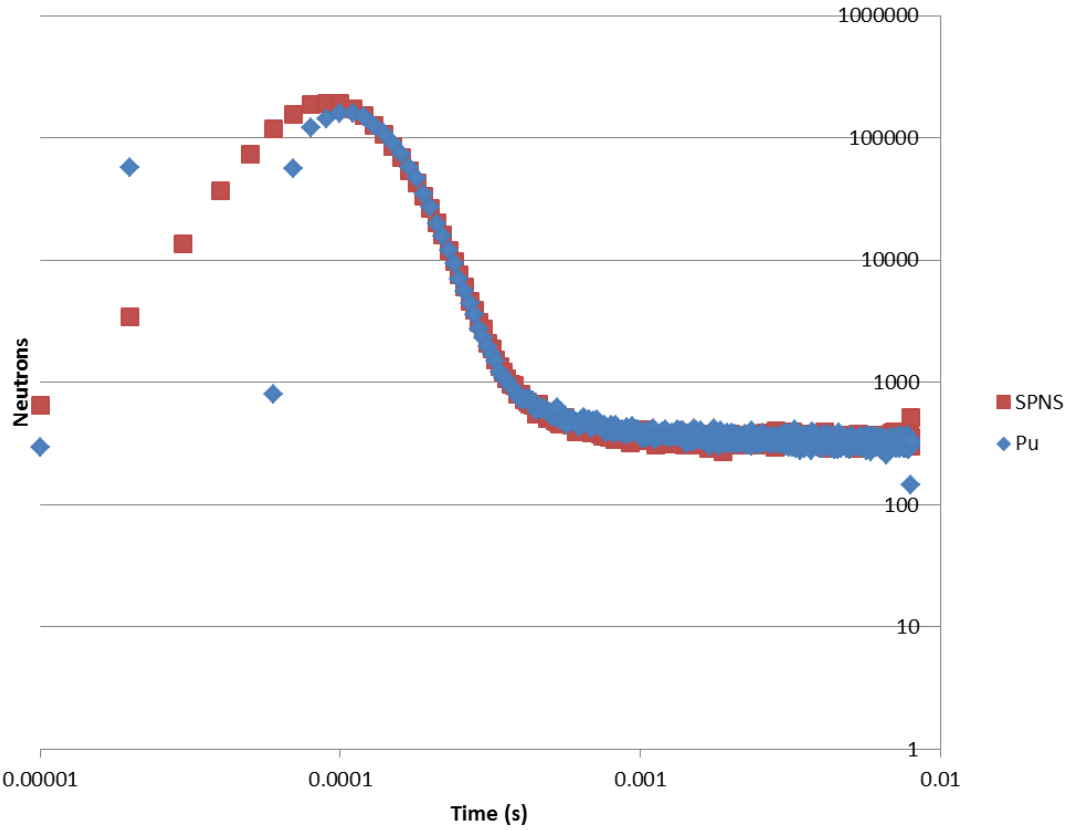

Figure B.6. SPNS Fitting of Pu

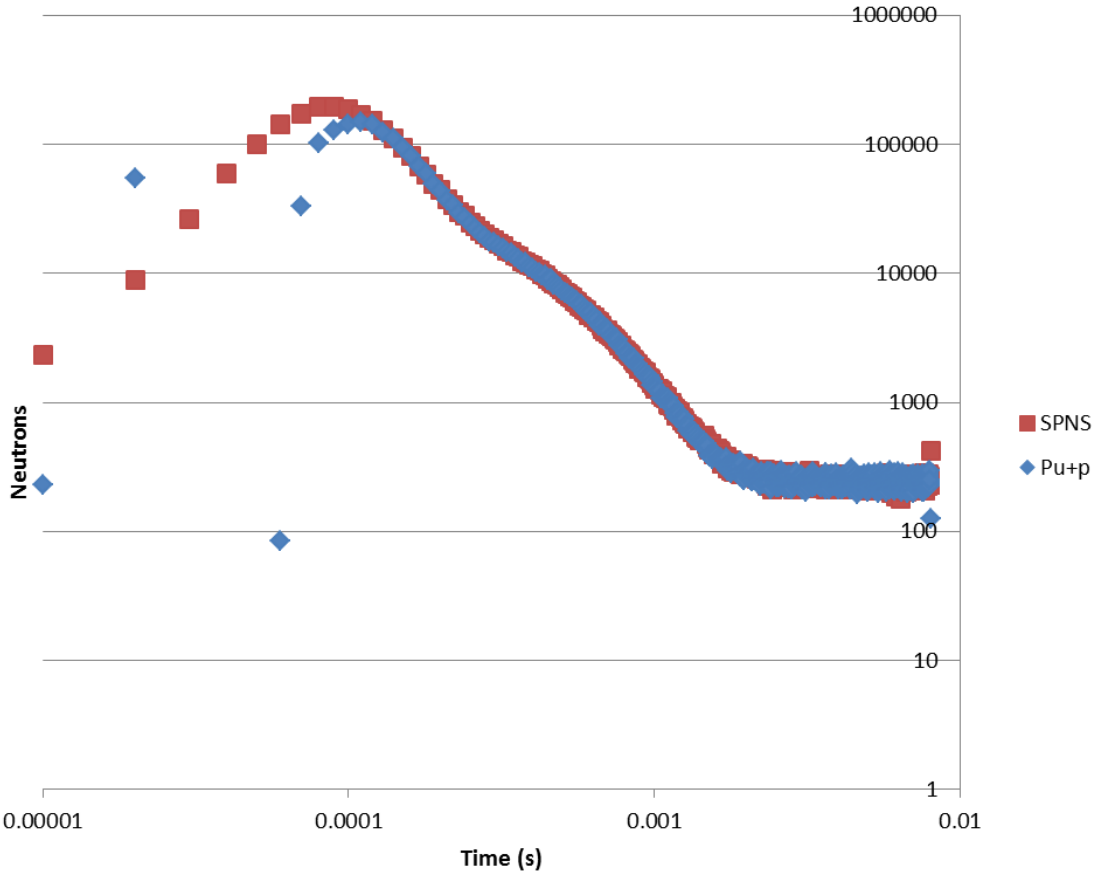

Figure B.7. SPNS Fitting of Pu+p

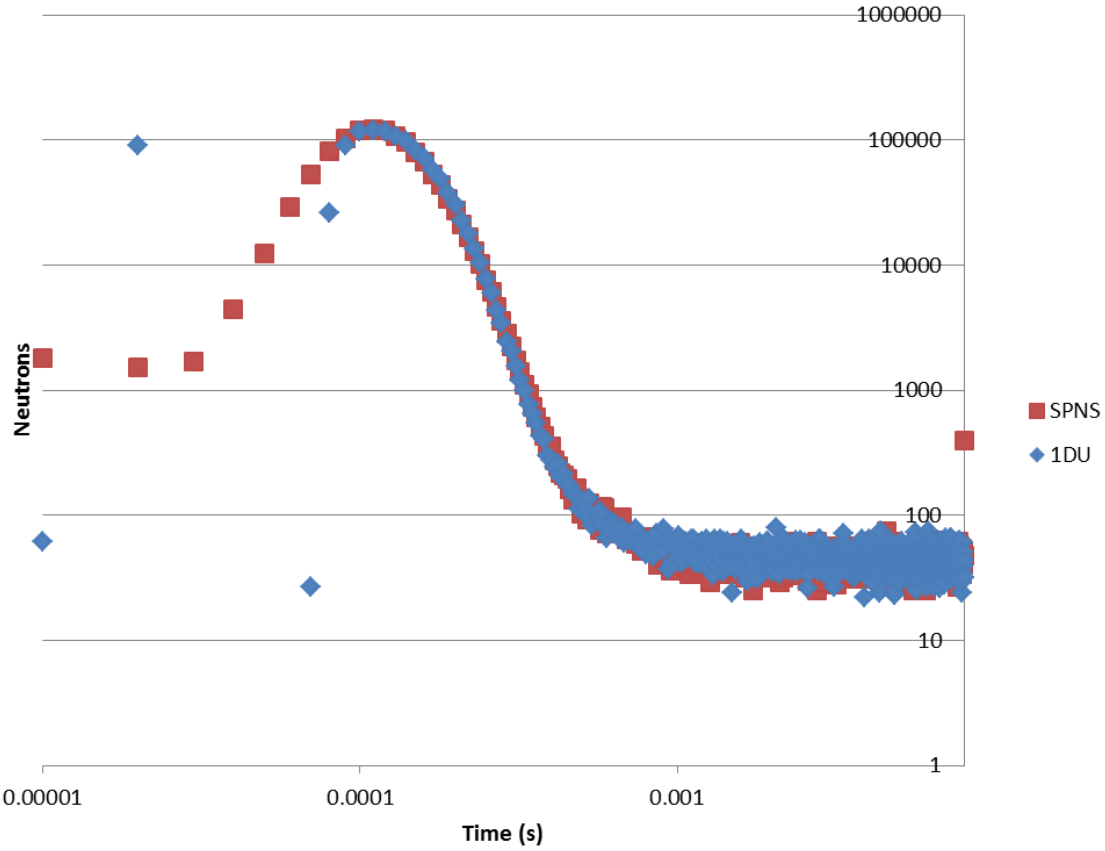

Figure B.8. SPNS Fitting of 1DU

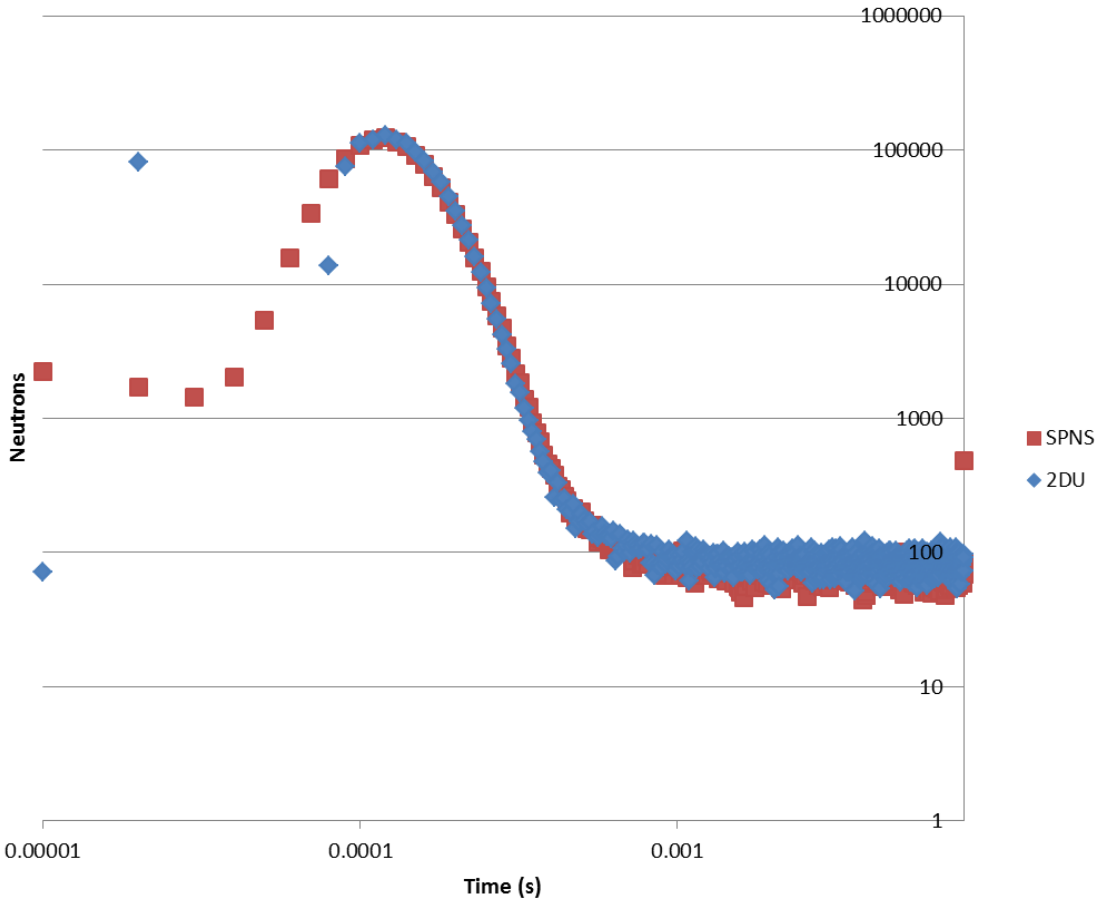

Figure B.9. SPNS Fitting of 2DU

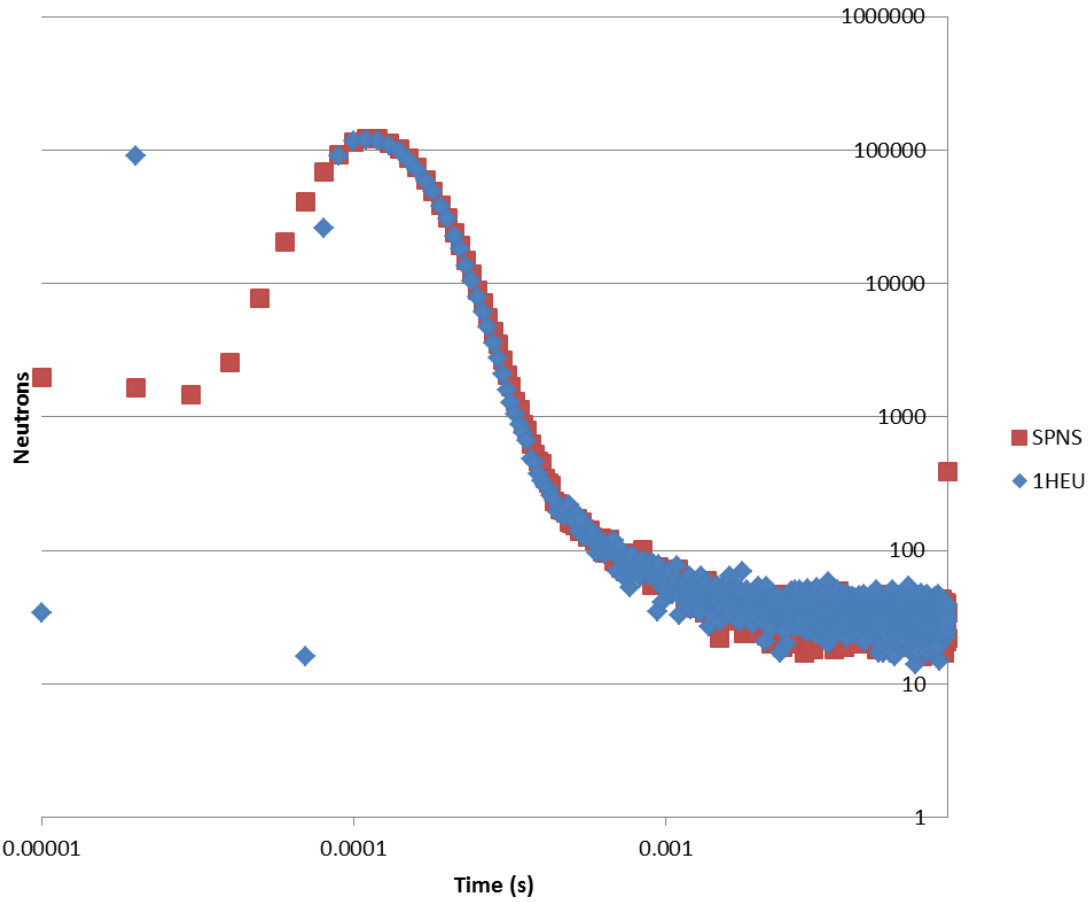

Figure B.10. SPNS Fitting of 1HEU

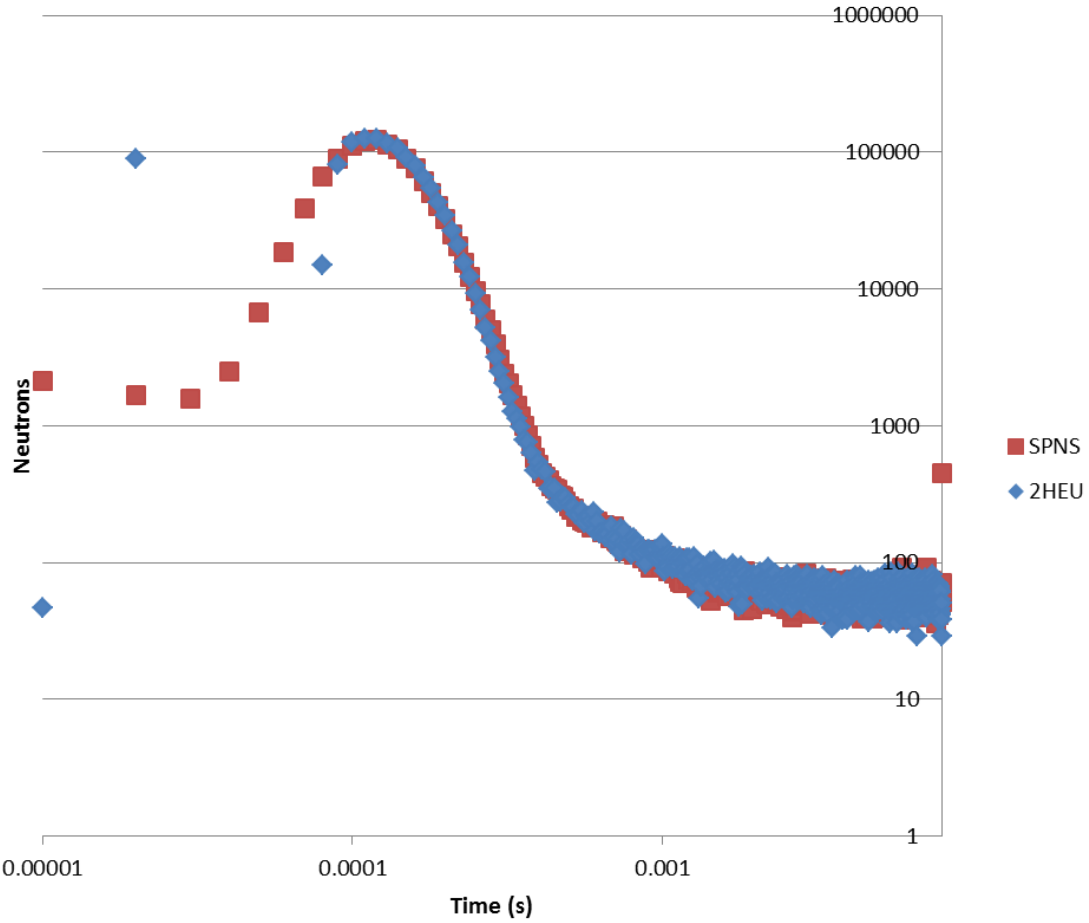

Figure B.11. SPNS Fitting of 2HEU

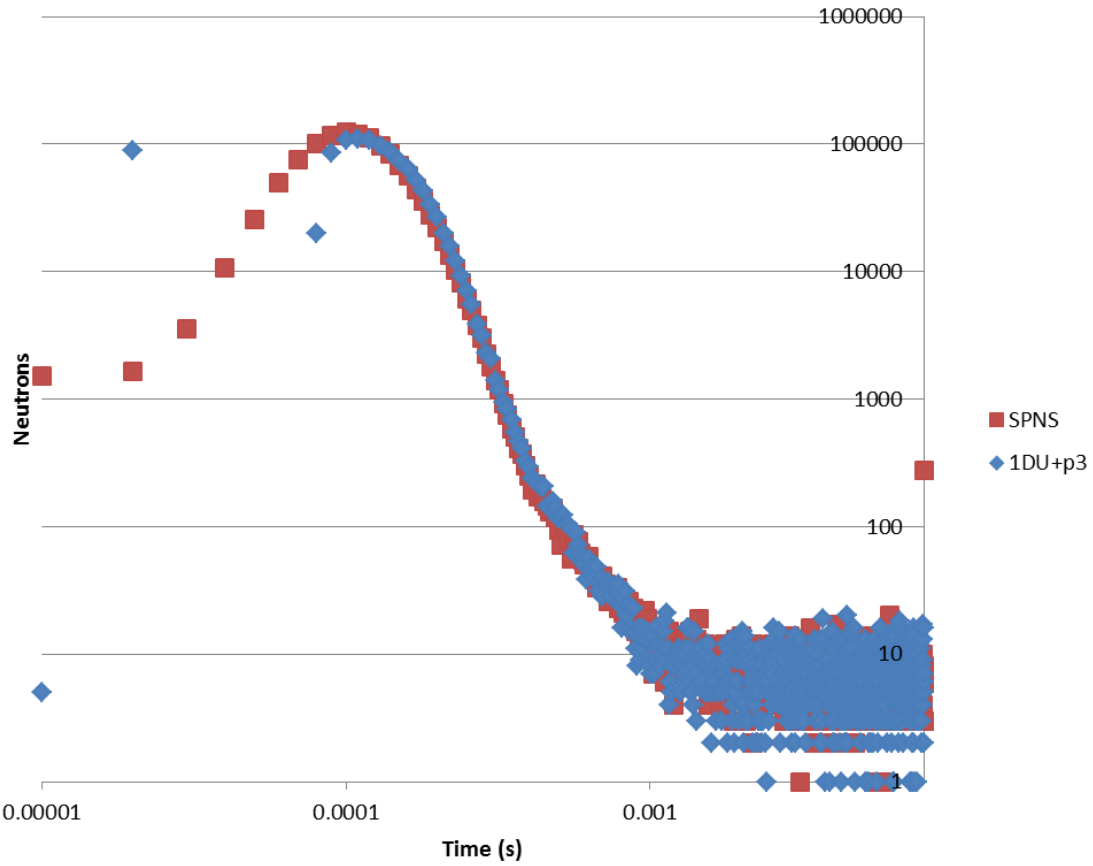

Figure B.12. SPNS Fitting of 1DU+p3

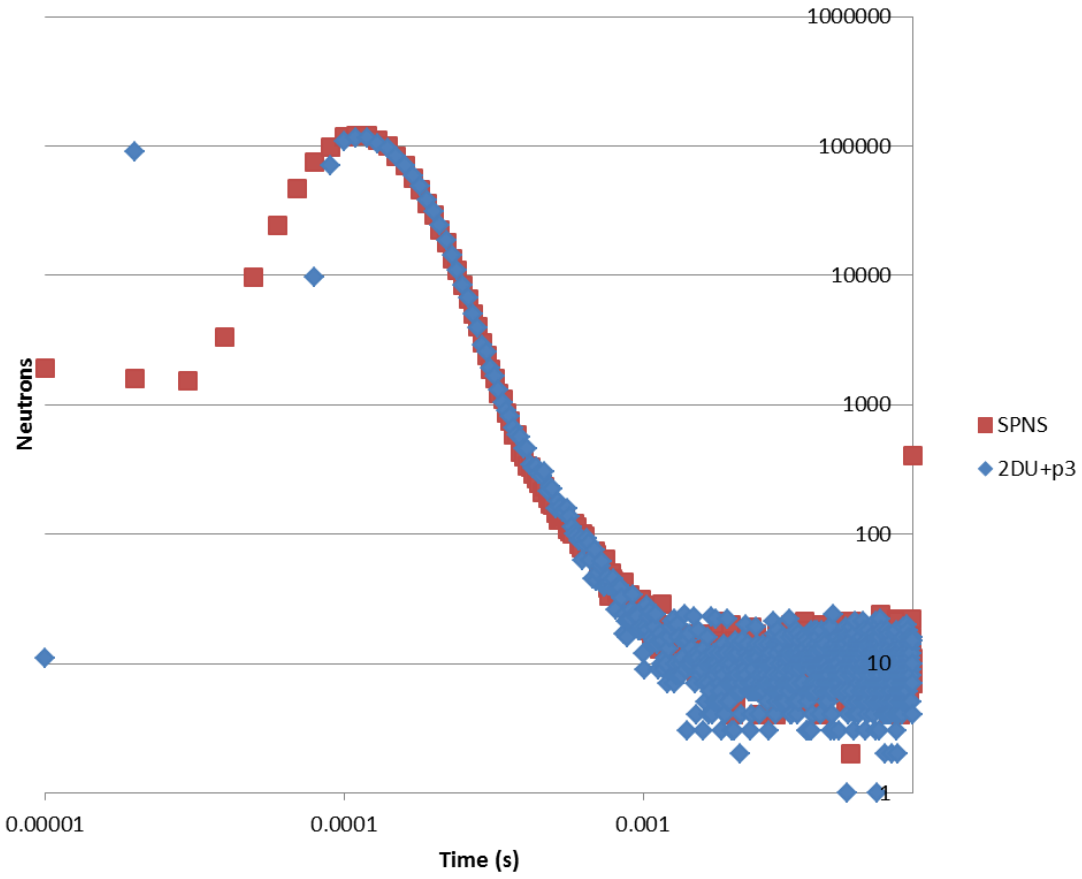

Figure B.13. SPNS Fitting of 2DU+p3

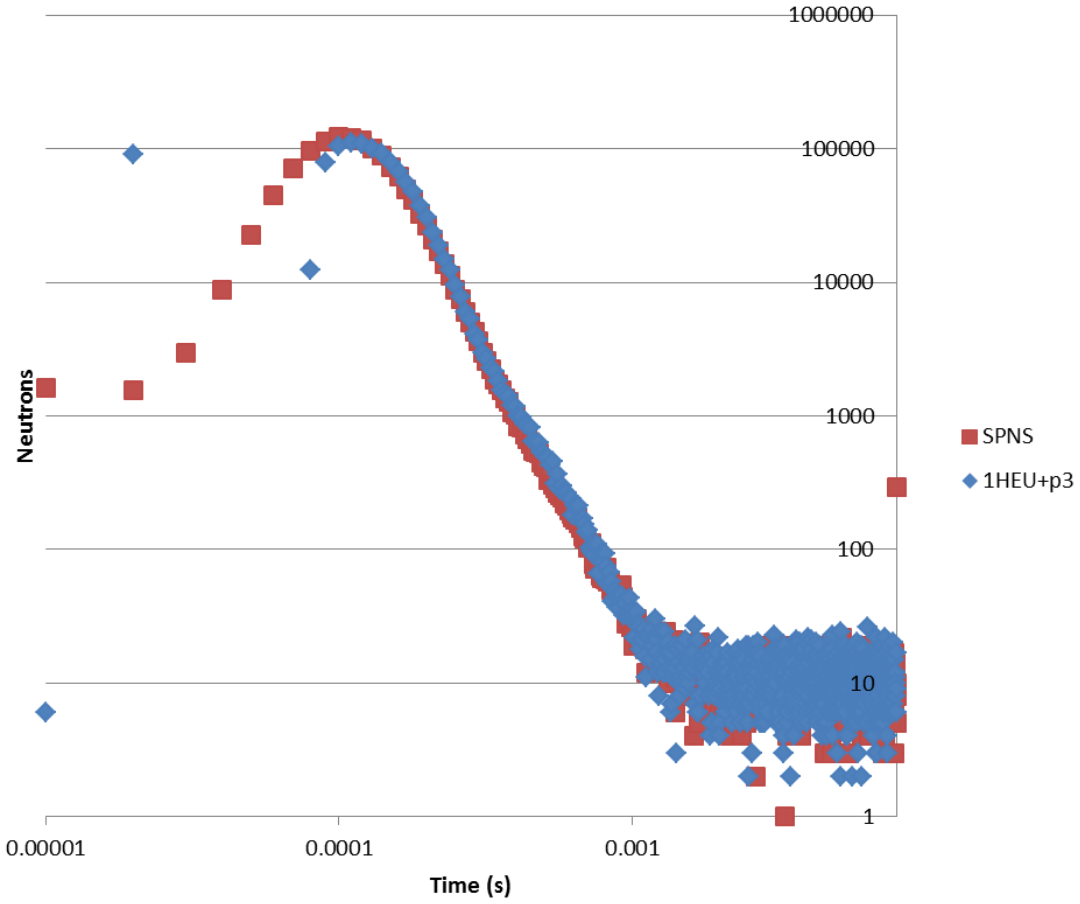

Figure B.14. SPNS Fitting of 1HEU+p3

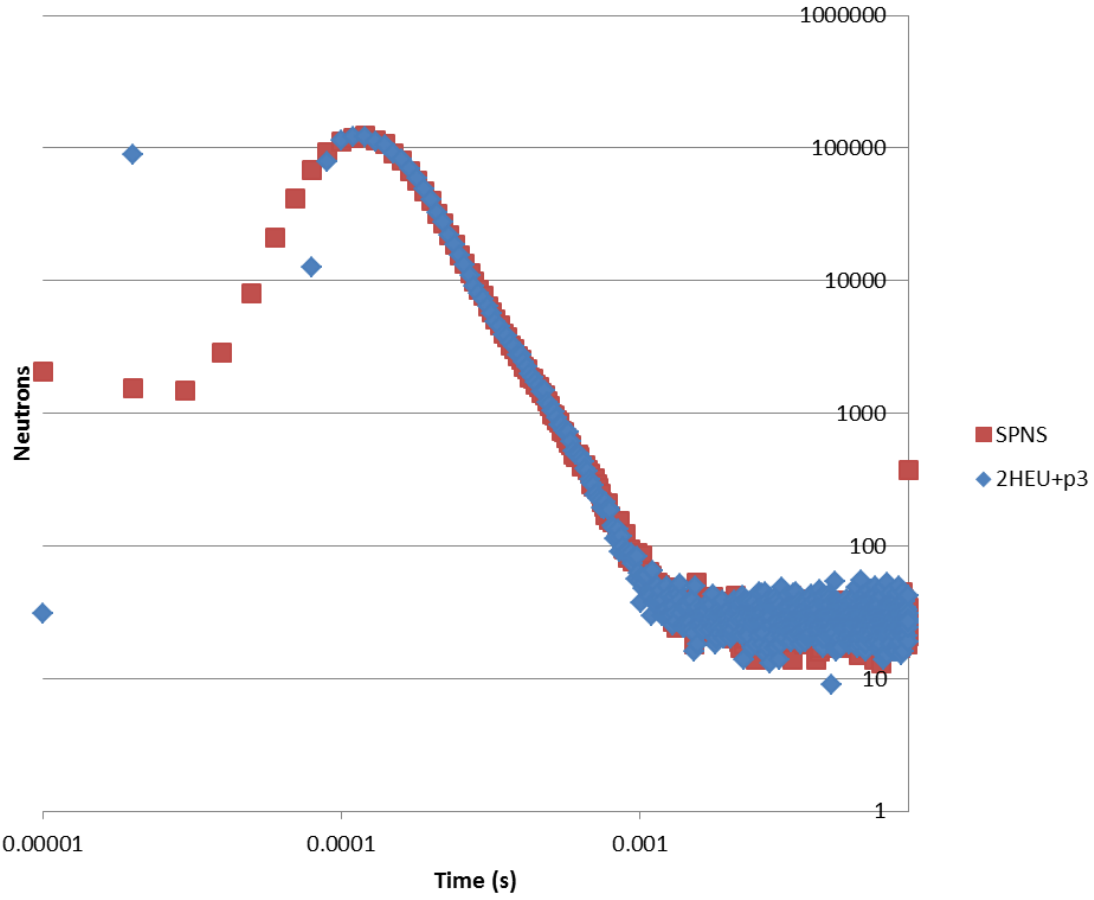

Figure B.15. SPNS Fitting of 2HEU+p3

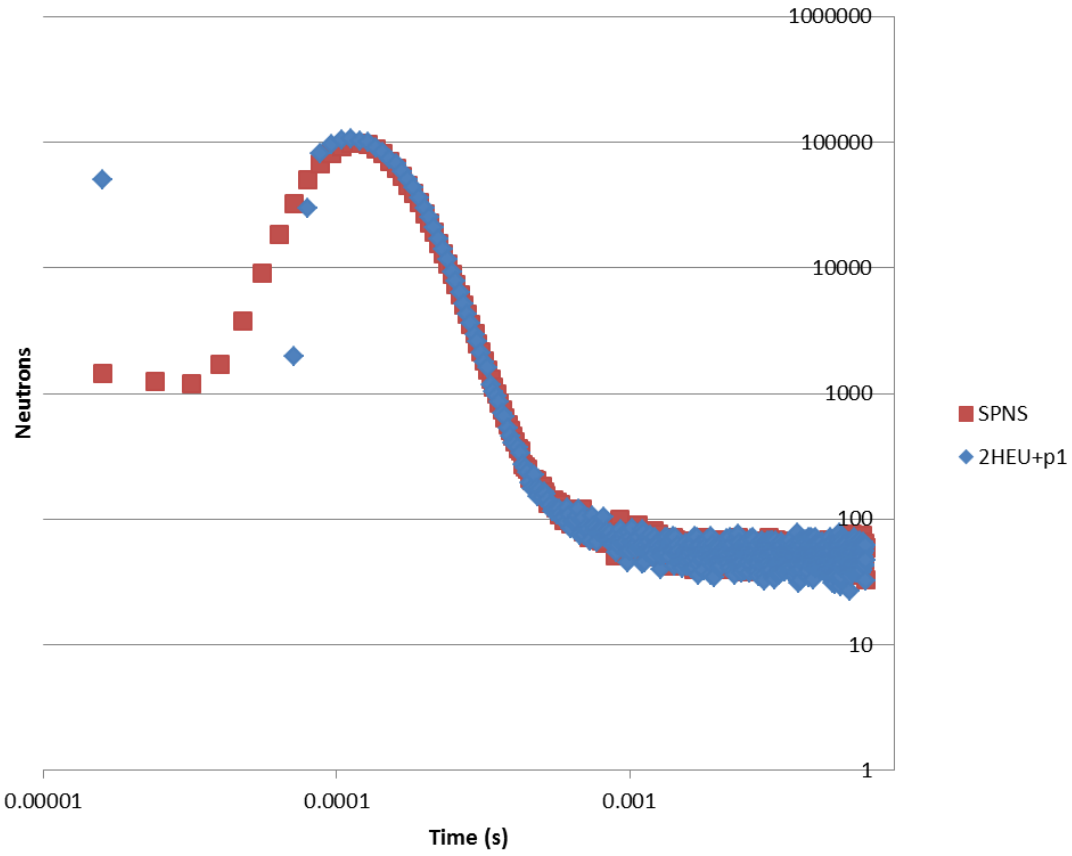

Figure B.16. SPNS Fitting of 2HEU+p1

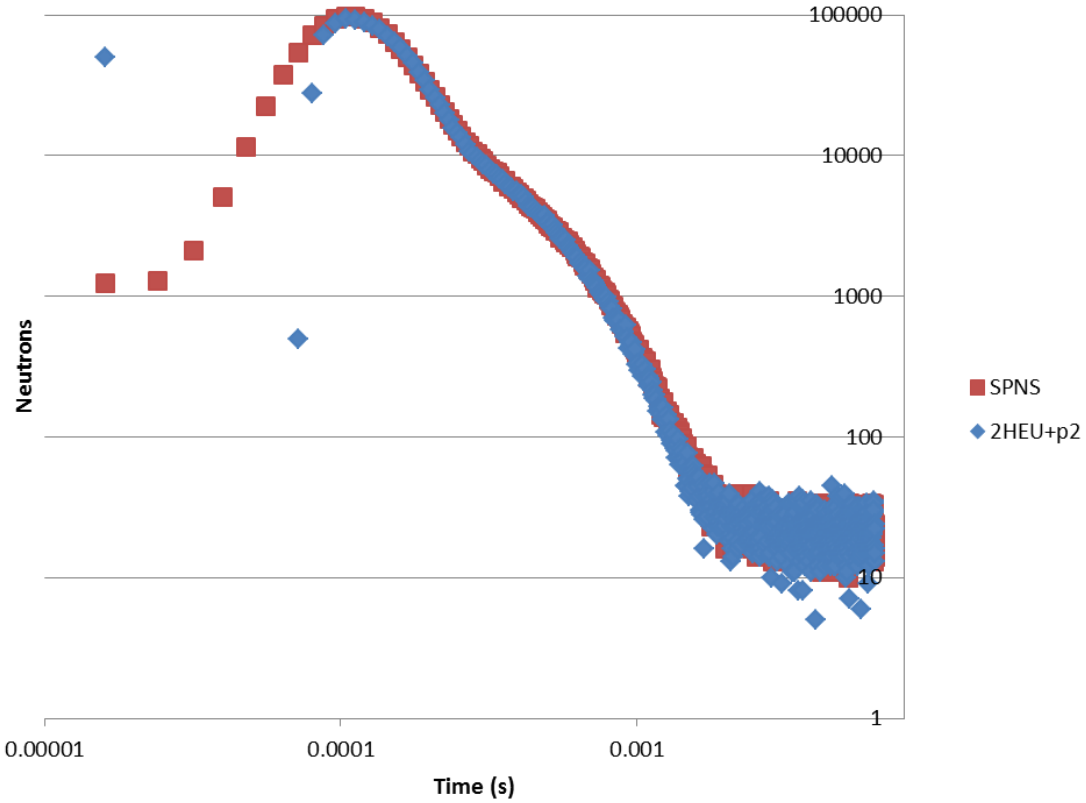

Figure B.17. SPNS Fitting of 2HEU+p2

APPENDIX C

PSR Program C++ Source Code (compiled with Microsoft Visual C++ 2010 Express)

C.1. PSR Main Program: PulsedSR.cpp (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
// Scott Stewart, 2012
// Pulsed Shift Register Program for
// INL Pulsed Neutron Data Analysis

#include "stdafx.h"
#include "SR_Func.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <queue>
#include <string>
#include <math.h>
#include <stdio.h>
#include <sstream>

using namespace std;

int main()
{
    SR_Func srf;
       string line,data,filename; //used to parse input parameter file
       NCDInfo firstn,itern; //used for file parsing
       size_t comment,datastart;

       long double gatew; //gate width SR param
       long double triggoff; // offset of trigger pulses from the time the pulse
occured
       long double window; // how long after the pulse to ignore data
       long double predelay; //how long to not count after a neutron pulse is
detected
       long double ldelay; //long delay (typically set to the pulse repeat rate)
       long double tsub=0; //time to subtract from count time (due to how we are
processing the file)
       long double dbls=0; //doubles value for the file
       long double vto=0; //veto value for the ncd file
       unsigned _int64 pulses=0; //singles value for the file
       unsigned _int64 RA[1000],A[1000]; //stores R+A and A information for the
pulses
       const int tbin=1000; //number of bins for doubles hist analysis
       long double tbinsize=8E-6; //size of tbin
       unsigned _int64 tRA[tbin],tA[tbin],tS[1000];
```

```
	queue<NCDInfo> pd; //stores pulses that are in the pre-delay
	queue<NCDInfo> rplusa; //stores pulses that are currently in the R+A gate
	queue<NCDInfo> longdelay; //stores pulses that are in the long delay region
	queue<NCDInfo> acci; //stores pulses that are in the Accidentals region
	queue<string> files; //stores file names to process

	ifstream input("PSR_Inp.txt",ios::in); //input parameters file
	ifstream tester; //used to test the filenames to make sure they work

	if(!input.is_open()) //if someone foolishly deleted the input file, this
scolds them
	{
		cout<<"ERROR: Could not find PSR_Inp.txt. Make sure it is in the
same directory as this program's .exe"<<endl;
		exit(0); //no reason to run program if there are no input
parameters. so the program quits
	}

	unsigned long int iter=0;
	long double iparam[6];

	for(int i=0;i<6;i++)
	{
		iparam[i] = 0;
	}

	while(!input.eof())
	{
		getline(input,line); //input grabbed a line at a time
		comment = line.find("//"); //finds whether or not there is a // on a
line. if there is this line is ignored (so don't put // on a line with input
parameters

		if(comment==string::npos&&line.length()!=0) //if there is no // on a
line, and if the line is not empty then we look for an input parameter
		{
			datastart = line.find("=");

			if(datastart==string::npos) //if there isn't an = sign on an
input parameter line, the input parameters are screwed up and the program quits
			{
				cout<<"ERROR: No equal sign (=) on expected input
parameter line."<<endl;
				exit(1);
			}

			int start=0,end=0;
			while(isspace((int)line[start+datastart+1])) //finds white
spaces at front of numbers
			{
				start++;
			}
```

```
                    while(isspace((int)line[line.length()-end-1])) //finds white
spaces at end of numbers
                    {
                           end++;
                    }

                    if(iter<6) //shift register parameters written as doubles
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);
                           istringstream datastream(data,istringstream::in);
                           datastream >> iparam[iter];
                    }
                    else
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);
                           filename = data;
                           tester.open(filename.c_str(),ios::in);

                           if(tester.is_open())
                           {
                                  files.push(filename);
                           }
                           else
                           {
                                  cout<<"ATTN: "<<filename<<" was not
found."<<endl;
                           }
                           tester.close();
                    }
                    iter++;
             }
       }
       input.close();

       //specifying input parameters
       predelay=iparam[0]; //predelay
       gatew=iparam[1]; //gate width
       ldelay=iparam[2]; //long delay
       triggoff=iparam[3]; //trigger offset
       window=iparam[4]; //window
       vto=iparam[5]; //veto
       cout<<endl<<"Input parameters processed."<<endl;

       ofstream summ("summary_out.psr.csv",ios::out);
       summ<<"filename,singles,doubles"<<endl<<endl;
       while(!files.empty())
       {
             //initilizations
             tsub=0;
             dbls=0;
             pulses=0;
             for(unsigned int i=0;i<1000;i++)
             {
```

```
                RA[i]=0;
                A[i]=0;
                tRA[i]=0;
                tA[i]=0;
                tS[i]=0;
        }
        srf.msize = 0;
        srf.tlast = 0;
        srf.veto=0;
        srf.exptrig=8000E-6;
        srf.lowtrig=0;
        srf.hightrig=0;
        srf.correcttrig=0;
        srf.trigtsub=0;

        srf.veto=vto;
        srf.win=window;
        srf.toff=triggoff;

        //File handling
        filename = files.front();
        files.pop();
        srf.BinaryReadIn(filename);
        srf.ProcessBinary();
        srf.WindowSubtraction();

        pulses = (int) srf.winsub.size();

        if(!srf.winsub.empty())
        {
                itern = srf.winsub.front();
                srf.winsub.pop();
        }
        else
        {
                cout<<"ATTN: No pulses in file."<<endl;
                exit(1);
        }

        //this handles the initial filling of the queues. By filling them
this way, the first pulse is being ignored for R+A - A purposes
        while(itern.time<gatew&&!srf.winsub.empty())
        {
                acci.push(itern);
                srf.winsub.pop();

                if(!srf.winsub.empty())
                {
                        itern = srf.winsub.front();
                }
                else
                {
                        break;
                }
        }
```

```
while(itern.time<(gatew+ldelay)&&!srf.winsub.empty())
{
	longdelay.push(itern);
	srf.winsub.pop();

	if(!srf.winsub.empty())
	{
		itern = srf.winsub.front();
	}
	else
	{
		break;
	}
}
while(itern.time<(gatew+ldelay+gatew)&&!srf.winsub.empty())
{
	rplusa.push(itern);
	srf.winsub.pop();

	if(!srf.winsub.empty())
	{
		itern = srf.winsub.front();
	}
	else
	{
		break;
	}
}
while(itern.time<(gatew+ldelay+gatew+predelay)&&!srf.winsub.empty())
{
	pd.push(itern);
	srf.winsub.pop();

	if(!srf.winsub.empty())
	{
		tsub = itern.time;
		itern = srf.winsub.front();
	}
	else
	{
		break;
	}
}

//Coincidence Analysis Section
int rasize=0,asize=0;
itern.ch=0;
itern.time=0;
int timed=0;
NCDInfo trig1;
bool timeanalysis=true, tldone=false;

if(!srf.winsub.empty())
{
	firstn = srf.winsub.front();
```

```
		}
		else
		{
			cout<<"ATTN: Doubles not computed due to input
parameters."<<endl;
		}

		if(!srf.outtrig.empty())
		{
			trig1 = srf.outtrig.front();
			srf.outtrig.pop();
		}
		else
		{
			cout<<"ATTN: Outtrig was empty before coincident
analysis."<<endl;
			timeanalysis=false;
		}

		while(!srf.winsub.empty())
		{
			//selecting neutron for coincidence analysis
			firstn= srf.winsub.front();

			//filling rplusa queue
			if(!pd.empty())
			{
				itern = pd.front();
			}
			while(itern.time<(firstn.time-predelay)&&!pd.empty())
			{
				rplusa.push(itern);
				pd.pop();

				if(!pd.empty())
				{
					itern = pd.front();
				}
				else
				{
					break;
				}
			}

			//filling long delay queue
			if(!rplusa.empty())
			{
				itern = rplusa.front();
			}
			while(itern.time<(firstn.time-predelay-
gatew)&&!rplusa.empty())
			{
				longdelay.push(itern);
				rplusa.pop();
```

```
                        if(!rplusa.empty())
                        {
                                itern = rplusa.front();
                        }
                        else
                        {
                                break;
                        }
                }

                //filling accidentals queue
                if(!longdelay.empty())
                {
                        itern = longdelay.front();
                }
                while(itern.time<(firstn.time-predelay-gatew-
ldelay)&&!longdelay.empty())
                {
                        acci.push(itern);
                        longdelay.pop();

                        if(!longdelay.empty())
                        {
                                itern = longdelay.front();
                        }
                        else
                        {
                                break;
                        }
                }

                //empty accidentals queue
                if(!acci.empty())
                {
                        itern = acci.front();
                }
                while(itern.time<(firstn.time-predelay-gatew-ldelay-
gatew)&&!acci.empty())
                {
                        acci.pop();

                        if(!acci.empty())
                        {
                                itern = acci.front();
                        }
                        else
                        {
                                break;
                        }
                }

                //coincidence computation
                rasize = (int) rplusa.size();
                asize = (int) acci.size();
```

```
                        //time binning coincidence analysis
                        if(timeanalysis)
                        {
                                while(!tldone)
                                {
                                        if(firstn.time<(trig1.time+srf.exptrig))
                                        {
                                                timed= (int) floor((firstn.time-
trig1.time)/tbinsize);

                                                if(timed<tbin)
                                                {
                                                        tRA[timed]+=rasize;
                                                        tA[timed]+=asize;
                                                        tS[timed]++;
                                                }

                                                tldone=true;
                                        }
                                        else if(!srf.outtrig.empty())
                                        {
                                                trig1 = srf.outtrig.front();
                                                srf.outtrig.pop();
                                        }
                                        else
                                        {
                                                tldone=true;
                                        }
                                }
                                tldone=false;
                        }

                        if(rasize<1000&&asize<1000)
                        {
                                RA[rasize]++;
                                A[asize]++;
                                dbls+=rasize;
                                dbls-=asize;
                        }
                        else
                        {
                                cout<<"ERROR: Increase size of R+A and A
histogram"<<endl;
                        }

                        //popping first neutron element used for this analysis
                        srf.winsub.pop();

                        //filling predelay queue
                        pd.push(firstn);
                }

                //computing results and outputing them
```

```
		long double ctime=(firstn.time)*(1-125*window)-tsub-srf.trigtsub;
//subtracts the amount of time ignored to properly account for accidentals from
the count time, also considers only the live time by removing the window
		long double stime=firstn.time*(1-125*window)-srf.trigtsub; //removes
the window time to account for livetime
		long double singleout=(double) pulses/stime;
		long double doubleout=dbls/ctime;
		string fn=filename+".psr.csv";

		remove(fn.c_str());
		ofstream output(fn.c_str(),ios::out); //opens output file based on
name of input file. will append the current data in the file

		output<<"Coincidence analysis for "<<filename<<endl;
		output<<"Singes:,"<<setprecision(15)<<singleout<<endl;
		output<<"Doubles:,"<<setprecision(15)<<doubleout<<endl;
		output<<"Time Loss to Bad Triggers:,"<<srf.trigtsub<<endl;
		output<<endl;
		output<<",R+A,A"<<endl;
		for(int i=0;i<1000;i++)
		{
			output<<i<<","<<RA[i]<<","<<A[i]<<endl;
		}
		output.close();

		if(timeanalysis)
		{
			string fn2=filename+".tcoinc.csv";
			remove(fn2.c_str());
			ofstream output2(fn2.c_str(),ios::out);

			output2<<"binstart,S,R+A,A"<<endl;
			for(int i=0;i<tbin;i++)
			{

	output2<<i*tbinsize<<","<<tS[i]<<","<<tRA[i]<<","<<tA[i]<<endl;
			}
			output2.close();
		}

		cout<<endl;
		cout<<filename<<" processed."<<endl;
		cout<<"Singes: "<<setprecision(15)<<singleout<<endl;
		cout<<"Doubles: "<<setprecision(15)<<doubleout<<endl;
		cout<<"Time Loss to Bad Triggers: "<<srf.trigtsub<<endl;


	summ<<filename<<","<<setprecision(15)<<singleout<<","<<doubleout<<endl;

		//destructor section
		while(!pd.empty())
		{
			pd.pop();
		}
		while(!rplusa.empty())
```

```
                {
                        rplusa.pop();
                }
                while(!longdelay.empty())
                {
                        longdelay.pop();
                }
                while(!acci.empty())
                {
                        acci.pop();
                }

                //destructor from srf class
                delete[] srf.memblock;

                while(!srf.ptrain.empty())
                {
                        srf.ptrain.pop();
                }
                while(!srf.triggers.empty())
                {
                        srf.triggers.pop();
                }
                while(!srf.outtrig.empty())
                {
                        srf.outtrig.pop();
                }
                while(!srf.winsub.empty())
                {
                        srf.winsub.pop();
                }
        }
        summ.close();

    return 0;
}
```

C.2. PSR Class Header File: SR_Func.h (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
#pragma once

#include <iostream>
#include <iomanip>
#include <ctime>
#include <fstream>
#include <string>
#include <math.h>
#include <list>
#include <queue>
using namespace std;

struct NCDInfo
{
	int ch; //channel number
	long double time; //time in seconds
};

class SR_Func
{
private:
	int ReadChannel(void); //determines the ch number and returns it as an int
	long double ReadTime(void); //determines the time from the ncd binary
values

public:
	char * memblock; //stores binary .ncd file in memory
	int msize; //tracks present location in memblock
	int intsize; //total size of memblock
	queue<NCDInfo> triggers; //a queue just storing the trigger information
	queue<NCDInfo> outtrig; //queue to keep triggers for time analysis
	queue<NCDInfo> ptrain; //a queue containing the pulse train information
sans triggers
	queue<NCDInfo> winsub; //a queue containing the pulse train information
with the window subtracted out
	long double toff; // offset of trigger pulses from the time the pulse
occured
	long double win; // how long after the pulse to ignore data
	long double tlast; //last time tracker for time function
	long double veto; //veto value for the ncd file
	bool channel[32]; //channel tracker
	int hightrig; //trigger is greater than expected
	int correcttrig; //trigger is in the range expected
	int lowtrig; //trigger is lower than expected
	long double exptrig; //expected time spacing between triggers
	long double trigtsub; //time subtracted from errors with trigger pulses

	SR_Func(void);
	~SR_Func(void);
	void BinaryReadIn(string); //reads in binary files
```

```
	void ProcessBinary(void); //processes the binary input stream into channel
numbers and time information. also seperates triggers
	void WindowSubtraction(void); //subtracts the pulses that lie within the
window
};
```

C.3. PSR Class Function File: SR_Func.cpp (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
#include "StdAfx.h"
#include "SR_Func.h"
#include <iostream>
#include <bitset>
#include <iomanip>
#include <fstream>
#include <queue>
#include <string>
#include <cmath>
using namespace std;

SR_Func::SR_Func(void) // constructor
{
}

SR_Func::~SR_Func(void) // destructor
{
}

void SR_Func::BinaryReadIn(string filename)
{
	ifstream file(filename.c_str(),ios::in|ios::binary|ios::ate); //input file
	if (file.is_open())
	{
		ifstream::pos_type bytesize;
		//reads in the file data
		bytesize=file.tellg(); //gets the number of bytes in the file
		intsize= (int) bytesize;
		memblock=new char[intsize]; //sets memblock to the size of the file
		file.seekg(0,ios::beg);
		file.read(memblock,bytesize); //sends memblock the memory location
of the bytes from the file
		file.close(); //closes the file after reading in the binary data as
char
	}
	else
	{
		cout<<"ATTN: Unable to open the ncd file."<<endl;
		exit(1);
	}
}

void SR_Func::ProcessBinary(void)
{
	NCDInfo ni;
	int cinfo=0; //channel indicator
	int charconv=0;
	long double neuttime=0; //for a particular event
	long double cht[32];
	int chcount=0;
	long double lasttrig=0;
```

```
	for(int i=0;i<32;i++)
	{
		cht[i]=0;
	}

	while(msize<intsize)
	{
		cinfo = ReadChannel();
		if(cinfo==99999) //two channels hit
		{
			for(int i=0;i<32;i++)
			{
				if(channel[i]==true)
				{
					chcount++;
				}
			}

			if(chcount<3)
			{
				neuttime = ReadTime();
				ni.time = neuttime;
				for(int i=0;i<32;i++)
				{
					if(channel[i]==true)
					{
						ni.ch = i;
						ptrain.push(ni);
					}
				}
			}
			else
			{
				msize+=4;
			}
			chcount=0;
		}
		else if(cinfo==88888) //message indicator
		{
			charconv = (int) memblock[msize];
			msize+=(charconv+1); //the additional plus one here accounts
for the piece of memblock that was consumed by the charconv variable
		}
		else
		{
			ni.ch=cinfo;
			neuttime = ReadTime();
			ni.time = neuttime;

			if(ni.ch==31)
			{
				ni.time-=toff; //subtracts the trigger offset time

				if((ni.time-lasttrig)<(exptrig-exptrig*.1))
```

```
                        {
                                lowtrig++;
                        }
                        else if((ni.time-lasttrig)>(exptrig))
                        {
                                hightrig++;
                        }
                        else
                        {
                                correcttrig++;
                        }

                        lasttrig=ni.time;
                        triggers.push(ni);
                        outtrig.push(ni);
                }
                else if(ni.ch==30)
                {
                        //for pulsed experiments, this contains beam current
info. Info ignored for now
                }
                else
                {
                        if(cht[ni.ch]!=0)
                        {
                                if((cht[ni.ch]+veto)<ni.time)
                                {
                                        cht[ni.ch]=ni.time;
                                        ptrain.push(ni);
                                }
                        }
                        else
                        {
                                cht[ni.ch]=ni.time;
                                ptrain.push(ni);
                        }
                }
        }
        cinfo=0;
        charconv=0;
        neuttime=0;
        }
}

int SR_Func::ReadChannel(void)
{
        const size_t bits = 8;
        std::bitset< bits > bit_val;
        unsigned char value = memblock[msize]; //selects a byte from the file
        bool twohit=false,hit=false; //indicators for hit channels and failed
conditions
        int cnum=0;
        int ccount=0;
```

```
	for(int i=0;i<32;i++) //sets all channels to false so that true indicates a
hit from that channel
	{
		channel[i]=false;
	}

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]

		if(bit_val[i]==1) //indicates that a particular channel was hit
		{
			channel[i+24]=true;
		}
	}
	msize++;
	value=memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]

		if(bit_val[i]==1) //indicates that a particular channel was hit
		{
			channel[i+16]=true;
		}
	}
	msize++;
	value=memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]

		if(bit_val[i]==1) //indicates that a particular channel was hit
		{
			channel[i+8]=true;
		}
	}
	msize++;
	value=memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]

		if(bit_val[i]==1) //indicates that a particular channel was hit
		{
			channel[i]=true;
		}
	}
```

```
	msize++;

	for(int i=0;i<32;i++)
	{
		if(channel[i]==true) //stores channel number
		{
			ccount++;
			cnum=i;
		}
	}

	if(ccount==1)
	{
		return cnum;
	}
	else if(ccount==0) //indicates that no channels were hit at all (message)
	{
		return 88888;
	}
	else //this indicates that more than one channel was hit
	{
		return 99999;
	}
}

long double SR_Func::ReadTime(void) //process time information from binary data
stream
{
	//***This function does not account for rollover***
	const size_t bits = 8;
	std::bitset< bits > bit_val;
	unsigned char value = memblock[msize];
	int bin[4*8];
	long double mult=1;
	long double retval=0;

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]
		bin[i+24]=bit_val[i];
	}
	msize++;
	value = memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]
		bin[i+16]=bit_val[i];
	}
	msize++;
	value = memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
```

```
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]
		bin[i+8]=bit_val[i];
	}
	msize++;
	value = memblock[msize];

	for(unsigned int i = 0 ; i < bits ; ++i)
	{
		bit_val[i] = (value >> i) & 1; // least significant bit is in
bit_val[0]
		bin[i]=bit_val[i];
	}
	msize++;

	for(int i=0; i<32; i++)
	{
		retval+=bin[i]*mult;
		mult*=2;
	}

	if(retval*100E-9<tlast)
	{
		cout<<"ERROR: Rollover has occured."<<endl;
	}

	tlast = retval*100E-9;
	return retval*100E-9; //ncd values are stored in 100E-9 bins, this converts
that info back to seconds
}

void SR_Func::WindowSubtraction(void)
{
	NCDInfo trig, pulse;
	bool notrig=true;
	bool overrun=false;
	long double oldtrig=0;
	long double lastptime=0;

	while(!ptrain.empty())
	{
		if(!triggers.empty())
		{
			notrig=false;
			trig = triggers.front();
			pulse = ptrain.front();

			if(pulse.time>(trig.time+win))
			{
				oldtrig=trig.time;
				if(overrun)
				{
					overrun=false;
					trigtsub+=(trig.time-lastptime);
```

```
                        }
                        triggers.pop();
                    }
                    else if(pulse.time<trig.time&&!overrun)
                    {
                        if(pulse.time<(oldtrig+exptrig))
                        {
                            winsub.push(pulse);
                        }
                        else
                        {
                            overrun=true;
                            lastptime=pulse.time;
                        }
                        ptrain.pop();
                    }
                    else
                    {
                        ptrain.pop();
                    }
                }
                else
                {
                    if(notrig)
                    {
                        pulse = ptrain.front();
                        winsub.push(pulse);
                        ptrain.pop();
                    }
                    else
                    {
                        pulse = ptrain.front();
                        if(pulse.time>(trig.time+win))
                        {
                            if(pulse.time<(oldtrig+exptrig))
                            {
                                winsub.push(pulse);
                            }
                        }
                        ptrain.pop();
                    }
                }
            }
        }
```

APPENDIX D

SPNS Program C++ Source Code (compiled with Microsoft Visual C++ 2010 Express)

D.1. SPNS Main Program File: NeutronSim.cpp (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
// Scott Stewart
// NeutronSim.cpp

#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <ctime>
#include <string>
#include <fstream>
#include <stdio.h>
#include <sstream>
#include <math.h>
#include <queue>
#include <bitset>
#include "NeutFunc.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
	//initilizations
	NeutFunc nf;
	NeutInfo ni;
	long double timep,timesf,timea; //neutron times; p = pulse, sf = SF, a = AN
	long double dataq[110]; //stores input long doubles for later assignment to
variables
	long double rnum = 0; //random number roll value
	string line,data; //used in parsing file
	size_t comment,datastart; //comment used to say where // is, datastart used
to say where = is on a line
	int iter=0,iter2=0; //placeholders while processing input file. iter is #
of lines read in. iter2 is num stored in dataq so far
	double sftest=0,fftest=0,iftest=0; //error code tracking variables

	ifstream input("SPNS_Inp2.txt",ios::in); //input parameters file

	if(!input.is_open()) //if someone foolishly deleted the input file, this
scolds them
	{
		cout<<"ERROR: Could not find SNS_Inp2.txt. Make sure it is in the
same directory as this program's .exe"<<endl;
```

```
			exit(0); //no reason to run program if there are no input
parameters. so the program quits
		}

		//grabs variables from file
		while(!input.eof())
		{
			getline(input,line); //input grabbed a line at a time
			comment = line.find("//"); //finds whether or not there is a // on a
line. if there is this line is ignored (so don't put // on a line with input
parameters

			if(comment==string::npos&&line.length()!=0) //if there is no // on a
line, and if the line is not empty then we look for an input parameter
			{
				datastart = line.find("=");

				if(datastart==string::npos) //if there isn't an = sign on an
input parameter line, the input parameters are screwed up and the program quits
				{
					cout<<"ERROR: No equal sign (=) on expected input
parameter line."<<endl;
					exit(1);
				}

				int start=0,end=0;
				while(isspace((int)line[start+datastart+1])) //finds white
spaces at front of numbers
				{
					start++;
				}

				while(isspace((int)line[line.length()-end-1])) //finds white
spaces at end of numbers
				{
					end++;
				}

				if(iter==0) //special case for the first line of data
				{
					data = line.substr(datastart+1+start,line.length()-
end); //start and end used here to eliminate any potential whitespace
					nf.filename=data; //keeps the filename to be used
later
					remove(nf.filename.c_str()); //removes any other file
with this name
				}
				else if(iter==1) //for the 2nd line of data
				{
					data = line.substr(datastart+1+start,line.length()-
end);
					istringstream datastream(data,istringstream::in);
					datastream >> nf.Repeat; //this is not stored in dataq
like the other numbers because it is an int.
				}
```

```
                    else if(iter==2) //for the third line of data
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);

                           if(data=="True"||data=="true"||data=="TRUE") //if
there is any indication of true on the line, sets ncd to true, which will lead to
an .ncd file output format
                                  nf.ncd=true;
                           else //if there is no indication of true on the line,
then the program assumes false
                                  nf.ncd=false;
                    }
                    else if(iter==3) //for the 4th line of data
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);

                           if(data=="True"||data=="true"||data=="TRUE") //any
indication of true will tell the program to sort the output
                                  nf.sort=true;
                           else //otherwise it won't
                                  nf.sort=false;
                    }
                    else if(iter==4) //for the 5th line of data
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);

                           if(data=="True"||data=="true"||data=="TRUE") //any
indication of true will tell the program to sort the output
                                  nf.doublefile=true;
                           else //otherwise it won't
                                  nf.doublefile=false;
                    }
                    else if(iter==6) //for the 7th line of data
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);
                           istringstream datastream(data,istringstream::in);
                           datastream >> nf.StartNeut; //this is not stored in
dataq like the other numbers because it is an int.
                    }
                    else //all other lines are written to dataq
                    {
                           data = line.substr(datastart+1+start,line.length()-
end);
                           istringstream datastream(data,istringstream::in);
                           datastream >> dataq[iter2];
                           iter2++; //times written to dataq (should be 32 by the
end)
                    }
                    iter++; //number of lines containing data
             }
      }
```

```
	if(iter2!=110||iter!=116) //if there was anything other than the values
expected in iter2 and in iter, then someone messed with the input parameters in
the file, and the program didn't take in the parameters properly
	{
		cout<<"ERROR: Did not read in expected number of parameters."<<endl;
		exit(1); //if this happened, then the output would be bad anyway, so
we exit now to save the users time
	}

	input.close();

	//Filling variables with info from file that were stored in dataq
	nf.CountTime = dataq[0]; //count time of detector
	nf.pulsetime = dataq[1]; //time of one pulse
	nf.PulseFreq = dataq[2]; //frequency of the pulse in Hz
	nf.StartExcl = dataq[3]; //start time of first exclusion zone
	nf.EndExcl = dataq[4]; //end time of first exclusion zone

	nf.SFrate = dataq[5]; //for steady-state; in Hz
	nf.ANrate = dataq[6]; // for steady-state; in Hz
	nf.ProbDetect = dataq[7];
	nf.ProbThermShort = dataq[8];
	nf.ProbThermLong = dataq[9];
	nf.ProbFiss = dataq[10]; //probability of fission given that the neutron is
thermalized
	nf.ProbFastFiss = dataq[11]; //probability of fast fission

	//Neutrons produced through induced fission
	for(int i=0;i<21;i++)
	{
		nf.IFissN[i]=dataq[i+12];
		iftest+=dataq[i+12];
	}

	//Neutrons produced through spontaneous fission
	for(int i=0;i<21;i++)
	{
		nf.SFissN[i] = dataq[i+33];
		sftest+=dataq[i+33];
	}

	//Neutrons produced through fast fission
	for(int i=0;i<9;i++)
	{
		nf.FFissN[i] = dataq[i+54];
		fftest+=dataq[i+54];
	}

	//Short Thermal / Long Thermal / Detector DieAway
	nf.detectordieaway = dataq[75];
	nf.shortdieaway = dataq[76];
	nf.longdieaway = dataq[77];

	//Pulse Trigger offset
```

```
	nf.trigoff = dataq[78];

	//Deadtime in Channels
	nf.deadch = dataq[79];

	//Probability of detection in one channel
	for(int i=0;i<30;i++)
	{
		nf.PNch[i]=dataq[i+80];
	}

	//Messages alert user to potential sources of error if ATTN. If ERROR they
state a known source of error and exit program
	if(iftest!=1)
		cout<<"ATTN: Induced fission probabilities do not add up to
1."<<endl;
	if(sftest!=1)
		cout<<"ATTN: Spontaneous fission probabilities do not add up to
1."<<endl;
	if(fftest!=1)
		cout<<"ATTN: Fast fission probabilities do not add up to 1."<<endl;
	if((nf.ProbDetect+nf.ProbThermShort+nf.ProbThermLong)>1)
		cout<<"ATTN: Probability of Detection, Thermalization, and Escape
are greater than 1."<<endl;
	if(nf.ProbFiss>1)
		cout<<"ATTN: Probability of Fission is greater than 1."<<endl;
	if(nf.PulseFreq==0&&nf.StartNeut!=0) //if someone has a zero pulse
frequency, but a non-zero size for a neutron pulse
	{
		cout<<"ERROR: PulseFreq is set to 0. Times for neutrons in pulses
will not be computed correctly."<<endl;
		exit(1);
	}
	if(nf.CountTime>429&&nf.ncd) //if count time is greater than 429 for .ncd
files, the four byte detector time will rollover
		cout<<"ATTN: Count Time is larger than what can be stored in 4
bytes. Binary time values will overflow."<<endl;
	if((nf.PNch[30]+nf.PNch[29]+nf.PNch[28]+nf.PNch[27]+nf.PNch[26]+nf.PNch[25]
+nf.PNch[24]+nf.PNch[23]+nf.PNch[22]+nf.PNch[21]+nf.PNch[20]+nf.PNch[19]+nf.PNch[1
8]+nf.PNch[17]+nf.PNch[16]+nf.PNch[15]+nf.PNch[14]+nf.PNch[13]+nf.PNch[12]+nf.PNch
[11]+nf.PNch[10]+nf.PNch[9]+nf.PNch[8]+nf.PNch[7]+nf.PNch[6]+nf.PNch[5]+nf.PNch[4]
+nf.PNch[3]+nf.PNch[2]+nf.PNch[1]+nf.PNch[0])!=1.0)
		cout<<"ATTN: Channel Detection Probabilities do not add up to
1."<<endl;
	if(nf.ncd&&!nf.sort)
		cout<<"ATTN: Deadtime will not work unless the sort option is turned
on."<<endl;
	/*if(nf.sort&&(nf.ANrate+nf.SFrate)>0&&nf.StartNeut>0)
	{
		cout<<"ATTN: Sorting will not work with SF, AN, and Pulsed neutrons.
It has been turned off."<<endl;
		nf.sort=false;
	}*/

	for(int m=0;m<nf.Repeat;m++)
```

```
	{
		//opening file
		string temp="",temp2="";

		if(nf.Repeat!=1)
		{
			stringstream yar;
			yar << (m+1);
			temp = yar.str();
			temp2 = temp;
			temp=nf.filename+"."+temp;
			temp2 = nf.filename+"_detfiss."+temp2+".csv";
		}
		else
		{
			temp=nf.filename;
			temp2 = nf.filename+"_detfiss"+".csv";
		}

		if(nf.doublefile)
		{
			nf.outfiss.open(temp2.c_str());
			nf.outfiss<<"Fission time, 1st Generation Fission #, Fission
Event #"<<endl;
		}

		if(nf.ncd)
		{
			temp=temp+".ncd";
			nf.output.open(temp.c_str(),ios_base::binary);
		}
		else
		{
			temp=temp+".pulse";
			nf.output.open(temp.c_str(),ios::out);
		}

		timep=0;
		timea=0;
		timesf=0;
		//Populates bank with initial neutrons

	while((timesf<nf.CountTime&&nf.SFrate>0)||(timea<nf.CountTime&&nf.ANrate>0)
||(timep<nf.CountTime&&(nf.StartNeut+nf.PulseFreq)>0))
		{
			if(!((nf.StartNeut+nf.PulseFreq)>0))
			{
				timep=nf.CountTime+1.0;
			}
			if(!(nf.ANrate>0))
			{
				timea=nf.CountTime+1.0;
			}
			if(!(nf.SFrate>0))
			{
```

```
                        timesf=nf.CountTime+1.0;
                }

                //pulse case, if the user put in non-zero start neutrons and
if timep is less than the count time

        if((nf.StartNeut+nf.PulseFreq)>0&&timep<nf.CountTime&&timep<=timesf&&timep<
=timea)
                {
                        for(int i=0;i<nf.StartNeut;i++)
                        {
                                long double randpt=nf.unifRand(0,nf.pulsetime);
//uses uniform random distribution to ensure an even distribution of neutrons over
the pulse time

                                if((timep+randpt)<nf.CountTime)
                                {
                                        ni.time = timep+randpt;
                                        ni.firstgen = 0;
                                        ni.fissnum = 0;
                                        nf.bank.push(ni); //bank is populated
with StartNeut neutrons.
                                        nf.bankhits++; //increments up for each
neutron stored in bank
                                }
                        }
                        if(nf.ncd&&(timep+nf.trigoff)<nf.CountTime) //prints
trigger channel for pulses in ncd mode
                        {
                                if(nf.sort) //sorted for later output
                                {
                                        ni.time = timep+nf.trigoff;
                                        ni.firstgen = -1; //triggers will be the
only events associated with a negative fission number of any kind
                                        ni.fissnum = -1;
                                        nf.SortOutput(ni); //ATTN: a delay has
been added to each trigger pulse
                                }
                                else //unsorted, so sent to output ASAP
                                {
                                        ni.time = timep+nf.trigoff;
                                        ni.firstgen = -1;
                                        ni.fissnum = -1;
                                        nf.Output_Handler(ni); //ATTN: a delay
has been added to each trigger pulse
                                }
                        }
                        timep+=1.0/nf.PulseFreq; //increments the timep by the
pulse frequency
                }
                //Alpha,N neut handling
                else
if(timea<nf.CountTime&&nf.ANrate>0&&timea<=timesf&&timea<=timep) //if time is less
than the count time, if the ANrate isn't zero
                {
```

```
                        timea += nf.StablePoi(nf.ANrate); //increments time by
steady state equation

                        if(timea<nf.CountTime)
                        {
                                ni.time = timea;
                                ni.firstgen = 0;
                                ni.fissnum = 0;
                                nf.bank.push(ni); //adds the one neutron to the
bank
                                nf.bankhits++;
                        }
                }

                //SF neut handling
                else
if(timesf<nf.CountTime&&nf.SFrate>0&&timesf<=timep&&timesf<=timea) //while less
than the count time with a nonzero SF rate
                {
                        rnum = nf.ran2(nf.idum); //rolls a random number to
determine how many neutrons are generated from the SF event
                        timesf += nf.StablePoi(nf.SFrate); //incrememnts time

                        if(timesf<nf.CountTime)
                        {
                                ni.time = timesf;
                                nf.firstgen++;
                                ni.firstgen = nf.firstgen;
                                nf.fissnum++;
                                ni.fissnum = nf.fissnum;

                                if(nf.SFissN[1]!=0 && rnum<nf.SFissN[1])
//spontaneous fission, 1 neutron produced
                                {
                                        nf.bank.push(ni); //neutron pushed onto
the end of the queue
                                        nf.bankhits++; //incremented to indicate
the neutron was generated
                                }
                                else if(nf.SFissN[2]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2])) //spontaneous fission, 2 neutrons produced
                                {
                                        for(int i=0;i<2;i++)
                                        {
                                                nf.bank.push(ni);
                                                nf.bankhits++;
                                        }
                                }
                                else if(nf.SFissN[3]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3])) //spontaneous fission, 3 neutrons
produced
                                {
                                        for(int i=0;i<3;i++)
                                        {
                                                nf.bank.push(ni);
```

```
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[4]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4])) //spontaneous fission,
4 neutrons produced
                        {
                                for(int i=0;i<4;i++)
                                {
                                        nf.bank.push(ni);
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[5]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]))
//spontaneous fission, 5 neutrons produced
                        {
                                for(int i=0;i<5;i++)
                                {
                                        nf.bank.push(ni);
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[6]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
])) //spontaneous fission, 6 neutrons produced
                        {
                                for(int i=0;i<6;i++)
                                {
                                        nf.bank.push(ni);
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[7]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7])) //spontaneous fission, 7 neutrons produced
                        {
                                for(int i=0;i<7;i++)
                                {
                                        nf.bank.push(ni);
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[8]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8])) //spontaneous fission, 8 neutrons produced
                        {
                                for(int i=0;i<8;i++)
                                {
                                        nf.bank.push(ni);
                                        nf.bankhits++;
                                }
                        }
                        else if(nf.SFissN[9]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
```

```
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9])) //spontaneous fission, 9 neutrons
produced
                                        {
                                                for(int i=0;i<9;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[10]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10])) //spontaneous fission, 10
neutrons produced
                                        {
                                                for(int i=0;i<10;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[11]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]))
//spontaneous fission, 11 neutrons produced
                                        {
                                                for(int i=0;i<11;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[12]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
)) //spontaneous fission, 12 neutrons produced
                                        {
                                                for(int i=0;i<12;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[13]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13])) //spontaneous fission, 13 neutrons produced
                                        {
                                                for(int i=0;i<13;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[14]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
```

```
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14])) //spontaneous fission, 14 neutrons produced
                                        {
                                                for(int i=0;i<14;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[15]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15])) //spontaneous fission, 15 neutrons
produced
                                        {
                                                for(int i=0;i<15;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[16]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15]+nf.SFissN[16])) //spontaneous fission,
16 neutrons produced
                                        {
                                                for(int i=0;i<16;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[17]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15]+nf.SFissN[16]+nf.SFissN[17]))
//spontaneous fission, 17 neutrons produced
                                        {
                                                for(int i=0;i<17;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
                                                }
                                        }
                                        else if(nf.SFissN[18]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15]+nf.SFissN[16]+nf.SFissN[17]+nf.SFissN[1
8])) //spontaneous fission, 18 neutrons produced
                                        {
                                                for(int i=0;i<18;i++)
                                                {
                                                        nf.bank.push(ni);
                                                        nf.bankhits++;
```

```
                                        }
                                }
                                else if(nf.SFissN[19]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15]+nf.SFissN[16]+nf.SFissN[17]+nf.SFissN[1
8]+nf.SFissN[19])) //spontaneous fission, 19 neutrons produced
                                {
                                        for(int i=0;i<19;i++)
                                        {
                                                nf.bank.push(ni);
                                                nf.bankhits++;
                                        }
                                }
                                else if(nf.SFissN[20]!=0 &&
rnum<(nf.SFissN[1]+nf.SFissN[2]+nf.SFissN[3]+nf.SFissN[4]+nf.SFissN[5]+nf.SFissN[6
]+nf.SFissN[7]+nf.SFissN[8]+nf.SFissN[9]+nf.SFissN[10]+nf.SFissN[11]+nf.SFissN[12]
+nf.SFissN[13]+nf.SFissN[14]+nf.SFissN[15]+nf.SFissN[16]+nf.SFissN[17]+nf.SFissN[1
8]+nf.SFissN[19]+nf.SFissN[20])) //spontaneous fission, 20 neutrons produced
                                {
                                        for(int i=0;i<20;i++)
                                        {
                                                nf.bank.push(ni);
                                                nf.bankhits++;
                                        }
                                }
                                else
                                {
                                        //Zero case, does nothing
                                }
                        }
                }
                else
                {
                        cout<<"ATTN: Unexpected case."<<endl;
                }

                nf.ProcessNeutronQueue(); // processes neutron queue after
pulse, AN, and SF generate one set of neutrons
        }

        if(nf.sort)
        {
                nf.PrintSorted();
        }

        nf.output.close();
        if(nf.doublefile)
                nf.outfiss.close();

        //lets the user know the number of neutrons that were ever in the
bank (thus total neutrons generated) and the number of neutrons successfully
detected. for one file
        cout<<endl;
        cout<<temp<<" created successfully."<<endl;
```

```
		cout<<"Number of neutrons generated: "<<nf.bankhits<<endl;
		cout<<"Number of neutrons detected: "<<nf.detectorhits<<endl; //does
not subtract neutrons that were excluded. so in that case not an indicator of
events in the file.
		if(nf.ncd&nf.sort)
			cout<<"Number of neutrons lost due to deadtime:
"<<nf.deadtimeneuts<<endl;

		//need to "reinitilize" the variables initilized at the beginning of
the class here since a new class isn't created for the next file
		nf.bankhits=0;
		nf.detectorhits=0;
		nf.deadtimeneuts=0;

		//continued reseeding of better random num generator. reseeding
since this is a new file
		time_t val=0;
		while(val==0)
			val=time(NULL);

		nf.idum = (long*) calloc(1,sizeof(long));
		*nf.idum = (long) val*-1;

		for(int i=0;i<30;i++)
		{
			nf.prevtime[i]=0;
			nf.ncdhit[i]=false;
		}
		nf.firstgen = 0;
		nf.fissnum = 0;
	}

	return 0;
}
```

D.2. SPNS Class Header File: NeutFunc.h (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
#pragma once

#include <iostream>
#include <iomanip>
#include <ctime>
#include <fstream>
#include <string>
#include <math.h>
#include <list>
#include <queue>
using namespace std;

//#define values used in ran2. do not alter
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

struct NeutInfo //neut sort is private because nothing other than functions inside
the class access this struct. it is only used to store data in the sorted list
{
	long double time; //neutron time
	long firstgen; //this is the number of the first generation fission that
created the chain
	long fissnum; //this is the integer number representing which fission
number this is sequentially in the program. it will not be sequential in time
};

class NeutFunc
{
	public:
		//This is the order that SPNS_Inputs.txt should be in to work
		string filename;//input file
		int Repeat; //number of times to repeat file generation
		bool ncd; //should the output be in ncd format
		bool sort; //should the output be sorted
		bool doublefile; //output file with fission times
		long double CountTime; //count time of detector
		int StartNeut; //size of pulse
		long double pulsetime; //time of one pulse
		long double PulseFreq; //frequency of the pulse in Hz
```

```
            long double StartExcl; //start time of first exclusion zone
            long double EndExcl; //end time of first exclusion zone
            long double SFrate; //for steady-state; in Hz
            long double ANrate; // for steady-state; in Hz
            long double ProbDetect; //probability that neutrons are detected by
the detector
            long double ProbThermShort; //prob of a fast thermalization
            long double ProbThermLong; //prob of a slow thermalization
            long double ProbFiss; //probability of fission given that the
neutron is thermalized
            long double ProbFastFiss; //probability fast fission will occur

            //Neutrons produced through induced fission
            long double IFissN[21];
            //Neutrons produced through spontaneous fission
            long double SFissN[21];
            //Neutrons produced via Fast Fission
            long double FFissN[21];

            //Short Thermal / Long Thermal / Detector DieAway
            long double longdieaway;
            long double shortdieaway;
            long double detectordieaway;

            //Offset for Pulse triggers
            long double trigoff;

            //channel deadtime
            long double deadch;
            //probability of seeing a neutron in channel
            long double PNch[30];

            //End of SPNS_Inputs file

            ofstream output,outfiss; //output files
            queue<NeutInfo> bank; //neutron bank; a queue is a first in, first
out data structure. this basically means that the neutrons wait in line until it
is their turn to be processed (like the badge office except the people are
neutrons)
            list<NeutInfo> neutlist; //sorted neutron time list; you can think
of this like a really large array, except I can easily write data to anywhere on
that array. technically this is a doubly linked list
            long bankhits; //total number of neutrons that have been in the bank
            long detectorhits; //total number of neutrons detected
            long deadtimeneuts; //number of neutrons not detected due to
deadtime
            long double prevtime[30]; //previous time per channel. tracked for
ncd output
            bool ncdhit[30]; //keeps track of when a channel has detected a
neutron
            long firstgen, fissnum;

            long * idum; //used for random numbers

            //functions
```

```
            NeutFunc(void); //cosntructor
            ~NeutFunc(void); //destructor
            void seed(); //seeds the pseudo random num generator (should only be
done once)
            float ran2(long*); //improved random number generator
            long double unifRand(long double, long double); //pseudo random num
generator for range [a,b)
            long double unifRand(); //pseudo random num generator for range
[0,1)
            long double StablePoi(long double); //generates time steps for the
steady state a,n and SF
            long double ShortExp(); //short exponential die-away. used for fast
thermalization
            long double LongExp(); //long exponential die away. used for slow
thermalization
            long double DetExp(); //used to simulate detector die away time
            void ProcessNeutronQueue(); //called when you want the neutron queue
to be emptied (neutrons are processed until the queue is empty)
            void Output_Handler(NeutInfo); //called to write output to a file
            bool Deadtime_Check(long double); //Handes deadtime per channel
            void PrintChar(int[8]); //called to write binary for the .ncd files
            void SortOutput(NeutInfo); //this is where detected neutrons are
placed in the sorted list based on their time
            void PrintSorted(); //this dumps what is in the list to an output
file
};
```

D.3. SPNS Class Function File: NeutFunc.cpp (line rollover occurs in MS Word so enclosed code cannot be compiled without correction)

```
#include "StdAfx.h"
#include "NeutFunc.h"
#include <iostream>
#include <iomanip>
#include <ctime>
#include <fstream>
#include <string>
#include <math.h>
#include <stdio.h>
#include <queue>
#include <list>
#include <bitset>
using namespace std;

NeutFunc::NeutFunc(void)
{
	//Class initilizations
	bankhits=0;
	detectorhits=0;
	deadtimeneuts=0;
	seed();

	//uses random num generator to initilize idum for ran2
	time_t val=0;
	while(val==0)
		val=time(NULL);

	idum = (long*) calloc(1,sizeof(long));
	*idum = (long) val*-1;

	for(int i=0;i<30;i++)
	{
		prevtime[i]=0;
		ncdhit[i]=false;
	}

	//initilizing long values to keep track of doubles
	firstgen = 0;
	fissnum = 0;
}


NeutFunc::~NeutFunc(void)
{
	delete idum;
}

// Generates a random number between 0 and 1 (uniformly distributed)
long double NeutFunc::unifRand()
{
	long double ans=1;
```

```
	while(ans==1) //this loop makes sure that rand() doesn't return 1. this is
done because ln(1-1) would throw an error in some of our calculations
	{
		ans = rand() / long double(RAND_MAX);
	}
    return ans;
}

//Generates a random number in the interval [a,b)
long double NeutFunc::unifRand(long double a, long double b)
{
    return (b-a)*ran2(idum) + a; //returns a rand for the range specified. note
that the range is [a,b) because of how unifRand() has been modified
}

// Reset the random number generator with the system clock.
void NeutFunc::seed()
{
    srand(time(NULL)); //seeds the pseudo rand num generator. should only be done
once per program, otherwise values will not be random
}

/*
From numerical recipies:

Long period (> 2 × 10^18) random number generator of L'Ecuyer with Bays-Durham
shuffle
and added safeguards. Returns a uniform random deviate between 0.0 and 1.0
(exclusive of
the endpoint values). Call with idum a negative integer to initialize; thereafter,
do not alter
idum between successive deviates in a sequence. RNMX should approximate the
largest floating
value that is less than 1.
*/
float NeutFunc::ran2(long *idum)
{
	int j;
	long k;
	static long idum2=123456789;
	static long iy=0;
	static long iv[NTAB];
	float temp;
	if (*idum <= 0) //Initialize.
	{
		if (-(*idum) < 1) *idum=1; //Be sure to prevent idum = 0.
		else *idum = -(*idum);
		idum2=(*idum);

		for (j=NTAB+7;j>=0;j--) //Load the shuffle table (after 8 warm-ups).
		{
			k=(*idum)/IQ1;
			*idum=IA1*(*idum-k*IQ1)-k*IR1;
			if (*idum < 0) *idum += IM1;
			if (j < NTAB) iv[j] = *idum;
```

```
			}
			iy=iv[0];
		}

		k=(*idum)/IQ1; //Start here when not initializing.
		*idum=IA1*(*idum-k*IQ1)-k*IR1; //Compute idum=(IA1*idum) % IM1 without
		if (*idum < 0) *idum += IM1; //overflows by Schrage's method.
		k=idum2/IQ2;
		idum2=IA2*(idum2-k*IQ2)-k*IR2; //Compute idum2=(IA2*idum) % IM2 likewise.
		if (idum2 < 0) idum2 += IM2;
		j=iy/NDIV; //Will be in the range 0..NTAB-1.
		iy=iv[j]-idum2; //Here idum is shuffled, idum and idum2 are
		iv[j] = *idum; //combined to generate output.
		if (iy < 1) iy += IMM1;

		if ((temp=AM*iy) > RNMX) return RNMX; //Because users don't expect endpoint
values.
		else return temp;
}

long double NeutFunc::StablePoi(long double rate)
{
		return -(log(1-ran2(idum)))/rate;
}

long double NeutFunc::ShortExp()
{
		return shortdieaway*-log(1-ran2(idum));
}

long double NeutFunc::LongExp()
{
		return longdieaway*-log(1-ran2(idum));
}

long double NeutFunc::DetExp()
{
		return detectordieaway*-log(1-ran2(idum));
}

void NeutFunc::ProcessNeutronQueue()
{
		bool excluded = false;
		long double rnum; //random num roll
		NeutInfo ni;

		//while loop until the bank is empty
		while(!bank.empty())
		{
			rnum = ran2(idum);

			if(rnum<ProbFastFiss)
			{
				ni = bank.front(); //for fast fission daughters have same
time as parent
```

```
                    //sets first generation fission number if it hasn't been
already
                    if(ni.firstgen==0)
                    {
                           firstgen++;
                           ni.firstgen = firstgen;
                    }

                    //sets fission number which tracks the neutrons created in
this specific fission event
                    fissnum++;
                    ni.fissnum=fissnum;

                    if(ni.time<CountTime)
                    {
                           rnum = ran2(idum); //reroll again to ensure proper
probability distribution
                           if(FFissN[1]!=0 && rnum<FFissN[1]) //induced fission
adds 1 neutron
                           {
                                  bank.push(ni);
                                  bankhits++; //increments int value of bankhits
to indicate that another neutron was put in the bank
                           }
                           else if(FFissN[2]!=0 && rnum<(FFissN[1]+FFissN[2]))
//induced fission adds 2 neutrons
                           {
                                  for(int i=0;i<2;i++)
                                  {
                                         bank.push(ni);
                                         bankhits++;
                                  }
                           }
                           else if(FFissN[3]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3])) //induced fission adds 3 neutrons
                           {
                                  for(int i=0;i<3;i++)
                                  {
                                         bank.push(ni);
                                         bankhits++;
                                  }
                           }
                           else if(FFissN[4]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4])) //induced fission adds 4 neutrons
                           {
                                  for(int i=0;i<4;i++)
                                  {
                                         bank.push(ni);
                                         bankhits++;
                                  }
                           }
                           else if(FFissN[5]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5])) //induced fission adds 5
neutrons
```

```
                        {
                                for(int i=0;i<5;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[6]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6])) //induced
fission adds 6 neutrons
                        {
                                for(int i=0;i<6;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[7]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]))
//induced fission adds 7 neutrons
                        {
                                for(int i=0;i<7;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[8]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]))//induced fission adds 8 neutrons
                        {
                                for(int i=0;i<8;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[9]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]))//induced fission adds 9 neutrons
                        {
                                for(int i=0;i<9;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[10]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]))//induced fission adds 10 neutrons
                        {
                                for(int i=0;i<10;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
```

```
                        }
                    }
                    else if(FFissN[11]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]))//induced fission adds 11 neutrons
                    {
                        for(int i=0;i<11;i++)
                        {
                            bank.push(ni);
                            bankhits++;
                        }
                    }
                    else if(FFissN[12]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]))//induced fission adds 12 neutrons
                    {
                        for(int i=0;i<12;i++)
                        {
                            bank.push(ni);
                            bankhits++;
                        }
                    }
                    else if(FFissN[13]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]))//induced fission adds
13 neutrons
                    {
                        for(int i=0;i<13;i++)
                        {
                            bank.push(ni);
                            bankhits++;
                        }
                    }
                    else if(FFissN[14]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]))//induced
fission adds 14 neutrons
                    {
                        for(int i=0;i<14;i++)
                        {
                            bank.push(ni);
                            bankhits++;
                        }
                    }
                    else if(FFissN[15]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]))/
/induced fission adds 15 neutrons
                    {
                        for(int i=0;i<15;i++)
                        {
                            bank.push(ni);
                            bankhits++;
                        }
                    }
```

```
                        else if(FFissN[16]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]+FF
issN[16]))//induced fission adds 16 neutrons
                        {
                                for(int i=0;i<16;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[17]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]+FF
issN[16]+FFissN[17]))//induced fission adds 17 neutrons
                        {
                                for(int i=0;i<17;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[18]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]+FF
issN[16]+FFissN[17]+FFissN[18]))//induced fission adds 18 neutrons
                        {
                                for(int i=0;i<18;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[19]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]+FF
issN[16]+FFissN[17]+FFissN[18]+FFissN[19]))//induced fission adds 19 neutrons
                        {
                                for(int i=0;i<19;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
                        }
                        else if(FFissN[20]!=0 &&
rnum<(FFissN[1]+FFissN[2]+FFissN[3]+FFissN[4]+FFissN[5]+FFissN[6]+FFissN[7]+FFissN
[8]+FFissN[9]+FFissN[10]+FFissN[11]+FFissN[12]+FFissN[13]+FFissN[14]+FFissN[15]+FF
issN[16]+FFissN[17]+FFissN[18]+FFissN[19]+FFissN[20]))//induced fission adds 20
neutrons
                        {
                                for(int i=0;i<20;i++)
                                {
                                        bank.push(ni);
                                        bankhits++;
                                }
```

```
                        }
                        else
                        {
                                //zero case; does nothing
                        }
                }
                bank.pop();
        }
        else
        {
                rnum = ran2(idum);

                if(rnum<ProbDetect) // detect case
                {
                        ni = bank.front(); //retrieves time from neutron in
front of queue
                        ni.time+=DetExp(); //adds detector die away to time of
neutron
                        if(ni.time<CountTime)
                        {
                                detectorhits++; //indicates that a neutron has
been detected. this does not subtract the neutrons that are excluded.
                                if(StartNeut>0&&(StartExcl+EndExcl)!=0) //if
the exclusion zone option is selected, this loop handles the exclusions
                                {
                                        int iter = 0;
                                        excluded=false; //this indicates whether
or not the value is excluded

       while(ni.time>StartExcl+(1.0/PulseFreq)*iter) //steps through the potential
exclusion zones within the count time. stops when the start of an exclusion zone
passes the time of the neutron
                                        {

       if(ni.time<(EndExcl+(1.0/PulseFreq)*(iter))) //given that the time is
greater than the current start of an exclusion zone, if it is also less than the
end of the exclusion zone it should be excluded
                                                        excluded = true; //tells
me that this value should be excluded
                                                iter++;
                                        }
                                        if(!excluded) //if it wasn't excluded
                                        {
                                                if(sort) //when it needs to be
sorted
                                                {
                                                        SortOutput(ni); //sends to
the sorted list
                                                }
                                                else //otherwise
                                                {
                                                        Output_Handler(ni);
//sends to output handler to be written to a file
                                                }
                                        }
```

```
                                }
                                else //if there is no exclusion zone
                                {
                                        if(sort)
                                        {
                                                SortOutput(ni);
                                        }
                                        else
                                        {
                                                Output_Handler(ni);
                                        }
                                }
                        }
                        bank.pop(); //deletes the neutron in the front of the
queue
                }
                else if(rnum<(ProbDetect+ProbThermShort+ProbThermLong)) //
thermalize short / long case
                {
                        ni = bank.front(); //retrieves the neutron information
from the front of the queue
                        if(rnum<(ProbDetect+ProbThermShort)) // adds a small
time to neutron time
                        {
                                ni.time += ShortExp();
                        }
                        else // adds a larger time to neutron time
                        {
                                ni.time += LongExp();
                        }

                        //probability it will fission
                        rnum = ran2(idum); //must reroll random number here,
otherwise we would use the rnum that got us into this else if statement, thus
biasing our answer
                        if(rnum<ProbFiss&&ni.time<CountTime)
                        {
                                rnum = ran2(idum); //reroll again to ensure
proper probability distribution

                                //sets first generation fission number if it
hasn't been already
                                if(ni.firstgen==0)
                                {
                                        firstgen++;
                                        ni.firstgen = firstgen;
                                }

                                //sets fission number which tracks the neutrons
created in this specific fission event
                                fissnum++;
                                ni.fissnum=fissnum;

                                if(IFissN[1]!=0 && rnum<IFissN[1]) //induced
fission adds 1 neutron
```

```
                                {
                                        bank.push(ni);
                                        bankhits++; //increments int value of
bankhits to indicate that another neutron was put in the bank
                                }
                                else if(IFissN[2]!=0 &&
rnum<(IFissN[1]+IFissN[2])) //induced fission adds 2 neutrons
                                {
                                        for(int i=0;i<2;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[3]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3])) //induced fission adds 3 neutrons
                                {
                                        for(int i=0;i<3;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[4]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4])) //induced fission adds 4 neutrons
                                {
                                        for(int i=0;i<4;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[5]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5])) //induced fission adds 5
neutrons
                                {
                                        for(int i=0;i<5;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[6]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6])) //induced
fission adds 6 neutrons
                                {
                                        for(int i=0;i<6;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[7]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]))
//induced fission adds 7 neutrons
```

```
                                {
                                        for(int i=0;i<7;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[8]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]))//induced fission adds 8 neutrons
                                {
                                        for(int i=0;i<8;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[9]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]))//induced fission adds 9 neutrons
                                {
                                        for(int i=0;i<9;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[10]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]))//induced fission adds 10 neutrons
                                {
                                        for(int i=0;i<10;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[11]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]))//induced fission adds 11 neutrons
                                {
                                        for(int i=0;i<11;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[12]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]))//induced fission adds 12 neutrons
                                {
                                        for(int i=0;i<12;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
```

```
                                        }
                                }
                                else if(IFissN[13]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]))//induced fission adds
13 neutrons
                                {
                                        for(int i=0;i<13;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[14]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]))//induced
fission adds 14 neutrons
                                {
                                        for(int i=0;i<14;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[15]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[15]))/
/induced fission adds 15 neutrons
                                {
                                        for(int i=0;i<15;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[16]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[16]))/
/induced fission adds 16 neutrons
                                {
                                        for(int i=0;i<16;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[17]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[16]+IF
issN[17]))//induced fission adds 17 neutrons
                                {
                                        for(int i=0;i<17;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
```

```
                                        }
                                }
                                else if(IFissN[18]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[16]+IF
issN[17]+IFissN[18]))//induced fission adds 18 neutrons
                                {
                                        for(int i=0;i<18;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[19]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[16]+IF
issN[17]+IFissN[18]+IFissN[19]))//induced fission adds 19 neutrons
                                {
                                        for(int i=0;i<19;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else if(IFissN[20]!=0 &&
rnum<(IFissN[1]+IFissN[2]+IFissN[3]+IFissN[4]+IFissN[5]+IFissN[6]+IFissN[7]+IFissN
[8]+IFissN[9]+IFissN[10]+IFissN[11]+IFissN[12]+IFissN[13]+IFissN[14]+IFissN[16]+IF
issN[17]+IFissN[18]+IFissN[19]+IFissN[20]))//induced fission adds 20 neutrons
                                {
                                        for(int i=0;i<20;i++)
                                        {
                                                bank.push(ni);
                                                bankhits++;
                                        }
                                }
                                else
                                {
                                        //zero case; does nothing
                                }
                        }
                        // kills the neutron that thermalized and caused
fission / escaped
                        bank.pop(); //deletes the neutron in the front of the
queue
                }
                else //escape case
                {
                        bank.pop(); //deletes the neutron from the front of
the queue
                }
        }
    }
}

void NeutFunc::Output_Handler(NeutInfo ni) //handles output to file
```

```
{
	if(doublefile) //outputs double information file
	{
		outfiss << setprecision(15) /*set precision forces all numbers to be
output without an E. this prevents an error in VBTAP*/<< ni.time*1E8; /*time
converted to shakes*/
		outfiss<<","<<ni.firstgen<<","<<ni.fissnum<<endl;
	}

	if(ncd) //.ncd file case
	{
		int t = (int) (ni.time/100E-9); //this forced conversion limits the
accuracy of the .ncd output, as all information past the decimal place is
truncated (.ncd files are stored in 100ns bins)
		int conv [4*8]; //binary values are stored in reverse in these
arrays, so least significant bit is in i=0, and most is in i=31
		int subset[8]; //subsets are also stored in reverse
		int place = 24;
		bool dead=false; //has the ncd data been ignored due to deadtime
		//prints channel information

		if(ni.firstgen==-1) //trigger channels are channel 31
		{
			for(int i=0;i<8;i++)
			{
				subset[i]=0;
			}
			subset[7]=1; //sets channel i=31
			PrintChar(subset); //print char handles a byte at a time
			subset[7]=0;
			PrintChar(subset);
			PrintChar(subset);
			PrintChar(subset);
		}
		else //non trigger ncd data is handled here
		{
			if(sort) //deadtime will only work if the neutrons are sorted
before being output
			{
				dead = Deadtime_Check(ni.time);
			}

			if(dead)
			{
				deadtimeneuts++;
			}
			else
			{
				for(int i=0;i<8;i++)
				{
					subset[i]=0;
					if(i<6)
					{
						if(ncdhit[i+24])
						{
```

```
                                        subset[i]=1;
                                        ncdhit[i+24]=false;
                                }
                        }
                }
                PrintChar(subset); //a char is one byte of information

                for(int i=0;i<8;i++)
                {
                        if(ncdhit[i+16])
                        {
                                subset[i]=1;
                                ncdhit[i+16]=false;
                        }
                        else
                                subset[i]=0;
                }
                PrintChar(subset);

                for(int i=0;i<8;i++)
                {
                        if(ncdhit[i+8])
                        {
                                subset[i]=1;
                                ncdhit[i+8]=false;
                        }
                        else
                                subset[i]=0;
                }
                PrintChar(subset);

                for(int i=0;i<8;i++)
                {
                        if(ncdhit[i])
                        {
                                subset[i]=1;
                                ncdhit[i]=false;
                        }
                        else
                                subset[i]=0;
                }
                PrintChar(subset);
        }
}

//prints time information
if(!dead)
{
        for(int i=0;i<4*8;i++)
        {
                conv[i]=t%2; //since this method generates a binary
num that is inverted, rollover will occur when the time exceeds ~429 seconds
                t/=2; //an additional check for rollover could be done
here. if this number is greater than 0 by the end of the for loop, then you know
rollover has occured
```

```
                        }

                        for(int j=0;j<4;j++) //bytes are sent to Print char from most
significant to least significant
                        {
                                for(int i=0;i<8;i++)
                                {
                                        subset[i]=conv[i+place]; //remember that bits
are stored in reverse order
                                }
                                place-=8; //place is an iterator that lets me specify
the byte to send to print char via the subset array
                                PrintChar(subset);
                        }
                }
        }
        else //if it isn't a .ncd file, output is written to a file with filename
specified by the user
                output << setprecision(15) /*set precision forces all numbers to be
output without an E. this prevents an error in VBTAP*/<< ni.time*1E8 /*time
converted to shakes*/<< endl;
}

bool NeutFunc::Deadtime_Check(long double time)
{
        long double crnum = ran2(idum);

        if((PNch[29]!=0)&&crnum<(PNch[29]))//ch 29
        {
                if(prevtime[29]==0)
                {
                        prevtime[29]=time;
                        ncdhit[29]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[29]+deadch))
                        {
                                prevtime[29]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[29]=time;
                                ncdhit[29]=true;
                                return false;
                        }

                }
        }
        else if((PNch[28]!=0)&&crnum<(PNch[29]+PNch[28]))//ch 28
        {
                if(prevtime[28]==0)
                {
```

```
                prevtime[28]=time;
                ncdhit[28]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[28]+deadch))
                {
                        prevtime[28]=time;
                        return true;
                }
                else
                {
                        prevtime[28]=time;
                        ncdhit[28]=true;
                        return false;
                }

        }
}
else if((PNch[27]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]))//ch 27
{
        if(prevtime[27]==0)
        {
                prevtime[27]=time;
                ncdhit[27]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[27]+deadch))
                {
                        prevtime[27]=time;
                        return true;
                }
                else
                {
                        prevtime[27]=time;
                        ncdhit[27]=true;
                        return false;
                }

        }
}
else if((PNch[26]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]))//ch26
{
        if(prevtime[26]==0)
        {
                prevtime[26]=time;
                ncdhit[26]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[26]+deadch))
```

```
                        {
                                prevtime[26]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[26]=time;
                                ncdhit[26]=true;
                                return false;
                        }

                }
        }
        else
if((PNch[25]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]))//ch 25
        {
                if(prevtime[25]==0)
                {
                        prevtime[25]=time;
                        ncdhit[25]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[25]+deadch))
                        {
                                prevtime[25]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[25]=time;
                                ncdhit[25]=true;
                                return false;
                        }

                }
        }
        else
if((PNch[24]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]))//
ch 24
        {
                if(prevtime[24]==0)
                {
                        prevtime[24]=time;
                        ncdhit[24]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[24]+deadch))
                        {
                                prevtime[24]=time;
                                return true;
                        }
```

```
                        else
                        {
                                prevtime[24]=time;
                                ncdhit[24]=true;
                                return false;
                        }

                }
        }
        else
if((PNch[23]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]))//ch23
        {
                if(prevtime[23]==0)
                {
                        prevtime[23]=time;
                        ncdhit[23]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[23]+deadch))
                        {
                                prevtime[23]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[23]=time;
                                ncdhit[23]=true;
                                return false;
                        }

                }
        }
        else
if((PNch[22]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]))//ch22
        {
                if(prevtime[22]==0)
                {
                        prevtime[22]=time;
                        ncdhit[22]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[22]+deadch))
                        {
                                prevtime[22]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[22]=time;
```

```
                        ncdhit[22]=true;
                        return false;
                }

        }
    }
    else
if((PNch[21]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]))//ch21
    {
        if(prevtime[21]==0)
        {
            prevtime[21]=time;
            ncdhit[21]=true;
            return false;
        }
        else
        {
            if(time<(prevtime[21]+deadch))
            {
                prevtime[21]=time;
                return true;
            }
            else
            {
                prevtime[21]=time;
                ncdhit[21]=true;
                return false;
            }

        }
    }
    else
if((PNch[20]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]))//ch20
    {
        if(prevtime[20]==0)
        {
            prevtime[20]=time;
            ncdhit[20]=true;
            return false;
        }
        else
        {
            if(time<(prevtime[20]+deadch))
            {
                prevtime[20]=time;
                return true;
            }
            else
            {
                prevtime[20]=time;
                ncdhit[20]=true;
                return false;
            }
```

```
		}
	}
	else
if((PNch[19]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]))//ch19
	{
		if(prevtime[19]==0)
		{
			prevtime[19]=time;
			ncdhit[19]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[19]+deadch))
			{
				prevtime[19]=time;
				return true;
			}
			else
			{
				prevtime[19]=time;
				ncdhit[19]=true;
				return false;
			}
		}
	}
	else
if((PNch[18]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]))//ch18
	{
		if(prevtime[18]==0)
		{
			prevtime[18]=time;
			ncdhit[18]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[18]+deadch))
			{
				prevtime[18]=time;
				return true;
			}
			else
			{
				prevtime[18]=time;
				ncdhit[18]=true;
				return false;
			}
		}
	}
	else
if((PNch[17]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]))//ch17
```

```
	{
		if(prevtime[17]==0)
		{
			prevtime[17]=time;
			ncdhit[17]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[17]+deadch))
			{
				prevtime[17]=time;
				return true;
			}
			else
			{
				prevtime[17]=time;
				ncdhit[17]=true;
				return false;
			}
		}
	}
	else
if((PNch[16]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]))//ch16
	{
		if(prevtime[16]==0)
		{
			prevtime[16]=time;
			ncdhit[16]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[16]+deadch))
			{
				prevtime[16]=time;
				return true;
			}
			else
			{
				prevtime[16]=time;
				ncdhit[16]=true;
				return false;
			}
		}
	}
	else
if((PNch[15]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]))
//ch15
	{
		if(prevtime[15]==0)
		{
			prevtime[15]=time;
```

```
                        ncdhit[15]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[15]+deadch))
                        {
                                prevtime[15]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[15]=time;
                                ncdhit[15]=true;
                                return false;
                        }
                }
        }
        else
if((PNch[14]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch
[14]))//ch14
        {
                if(prevtime[14]==0)
                {
                        prevtime[14]=time;
                        ncdhit[14]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[14]+deadch))
                        {
                                prevtime[14]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[14]=time;
                                ncdhit[14]=true;
                                return false;
                        }
                }
        }
        else
if((PNch[13]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch
[14]+PNch[13]))//ch13
        {
                if(prevtime[13]==0)
                {
                        prevtime[13]=time;
                        ncdhit[13]=true;
                        return false;
                }
```

```
		else
		{
			if(time<(prevtime[13]+deadch))
			{
				prevtime[13]=time;
				return true;
			}
			else
			{
				prevtime[13]=time;
				ncdhit[13]=true;
				return false;
			}
		}
	}
	else
if((PNch[12]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch
[14]+PNch[13]+PNch[12]))//ch12
	{
		if(prevtime[12]==0)
		{
			prevtime[12]=time;
			ncdhit[12]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[12]+deadch))
			{
				prevtime[12]=time;
				return true;
			}
			else
			{
				prevtime[12]=time;
				ncdhit[12]=true;
				return false;
			}
		}
	}
	else
if((PNch[11]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch
[14]+PNch[13]+PNch[12]+PNch[11]))//ch11
	{
		if(prevtime[11]==0)
		{
			prevtime[11]=time;
			ncdhit[11]=true;
			return false;
		}
		else
		{
			if(time<(prevtime[11]+deadch))
```

```
                        {
                                prevtime[11]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[11]=time;
                                ncdhit[11]=true;
                                return false;
                        }
                }
        }
        else
if((PNch[10]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNc
h[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch
[14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]))//ch10
        {
                if(prevtime[10]==0)
                {
                        prevtime[10]=time;
                        ncdhit[10]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[10]+deadch))
                        {
                                prevtime[10]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[10]=time;
                                ncdhit[10]=true;
                                return false;
                        }
                }
        }
        else
if((PNch[9]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]))//ch9
        {
                if(prevtime[9]==0)
                {
                        prevtime[9]=time;
                        ncdhit[9]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[9]+deadch))
                        {
                                prevtime[9]=time;
                                return true;
```

```
                }
                else
                {
                        prevtime[9]=time;
                        ncdhit[9]=true;
                        return false;
                }
        }
    }
    else
if((PNch[8]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]))//ch8
    {
        if(prevtime[8]==0)
        {
                prevtime[8]=time;
                ncdhit[8]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[8]+deadch))
                {
                        prevtime[8]=time;
                        return true;
                }
                else
                {
                        prevtime[8]=time;
                        ncdhit[8]=true;
                        return false;
                }
        }
    }
    else
if((PNch[7]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]))//ch 7
    {
        if(prevtime[7]==0)
        {
                prevtime[7]=time;
                ncdhit[7]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[7]+deadch))
                {
                        prevtime[7]=time;
                        return true;
                }
                else
                {
```

```
                        prevtime[7]=time;
                        ncdhit[7]=true;
                        return false;
                }
        }
    }
    else
if((PNch[6]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]))//ch 6
    {
        if(prevtime[6]==0)
        {
            prevtime[6]=time;
            ncdhit[6]=true;
            return false;
        }
        else
        {
            if(time<(prevtime[6]+deadch))
            {
                prevtime[6]=time;
                return true;
            }
            else
            {
                prevtime[6]=time;
                ncdhit[6]=true;
                return false;
            }
        }
    }
    else
if((PNch[5]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]+PNch[5]))/
/ch 5
    {
        if(prevtime[5]==0)
        {
            prevtime[5]=time;
            ncdhit[5]=true;
            return false;
        }
        else
        {
            if(time<(prevtime[5]+deadch))
            {
                prevtime[5]=time;
                return true;
            }
            else
            {
                prevtime[5]=time;
                ncdhit[5]=true;
```

```
                        return false;
                }
        }
}
else
if((PNch[4]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]+PNch[5]+PN
ch[4]))//ch 4
{
        if(prevtime[4]==0)
        {
                prevtime[4]=time;
                ncdhit[4]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[4]+deadch))
                {
                        prevtime[4]=time;
                        return true;
                }
                else
                {
                        prevtime[4]=time;
                        ncdhit[4]=true;
                        return false;
                }
        }
}
else
if((PNch[3]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]+PNch[5]+PN
ch[4]+PNch[3]))//ch 3
{
        if(prevtime[3]==0)
        {
                prevtime[3]=time;
                ncdhit[3]=true;
                return false;
        }
        else
        {
                if(time<(prevtime[3]+deadch))
                {
                        prevtime[3]=time;
                        return true;
                }
                else
                {
                        prevtime[3]=time;
                        ncdhit[3]=true;
                        return false;
```

```
                        }
                }
        }
        else
if((PNch[2]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]+PNch[5]+PN
ch[4]+PNch[3]+PNch[2]))//ch 2
        {
                if(prevtime[2]==0)
                {
                        prevtime[2]=time;
                        ncdhit[2]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[2]+deadch))
                        {
                                prevtime[2]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[2]=time;
                                ncdhit[2]=true;
                                return false;
                        }
                }
        }
        else
if((PNch[1]!=0)&&crnum<(PNch[29]+PNch[28]+PNch[27]+PNch[26]+PNch[25]+PNch[24]+PNch
[23]+PNch[22]+PNch[21]+PNch[20]+PNch[19]+PNch[18]+PNch[17]+PNch[16]+PNch[15]+PNch[
14]+PNch[13]+PNch[12]+PNch[11]+PNch[10]+PNch[9]+PNch[8]+PNch[7]+PNch[6]+PNch[5]+PN
ch[4]+PNch[3]+PNch[2]+PNch[1]))//ch 1
        {
                if(prevtime[1]==0)
                {
                        prevtime[1]=time;
                        ncdhit[1]=true;
                        return false;
                }
                else
                {
                        if(time<(prevtime[1]+deadch))
                        {
                                prevtime[1]=time;
                                return true;
                        }
                        else
                        {
                                prevtime[1]=time;
                                ncdhit[1]=true;
                                return false;
                        }
```

```
			}
		}
		else//ch 0
		{
			if(PNch[0]!=0)
			{
				if(prevtime[0]==0)
				{
					prevtime[0]=time;
					ncdhit[0]=true;
					return false;
				}
				else
				{
					if(time<(prevtime[0]+deadch))
					{
						prevtime[0]=time;
						return true;
					}
					else
					{
						prevtime[0]=time;
						ncdhit[0]=true;
						return false;
					}
				}
			}
		}
}

void NeutFunc::PrintChar(int sub[8])
{
		int ans=0, mult=1;
		char print;

		for(int i=0;i<8;i++) //converts the binary bit to an integer
		{
			ans+=sub[i]*mult;
			mult*=2;
		}

		print = (char) ans; //converts the int to a char represented by that int
value
		output.write((char *) &print,1); //outputs the char byte to a binary file
}

void NeutFunc::SortOutput(NeutInfo ni) //list is organized from smallest time to
largest time
{
		NeutInfo ns,ns1;
		int size = neutlist.size(); //gets current list size
		int midpoint = size/2;
		list<NeutInfo>::reverse_iterator niter; //iterator used to go through the
list
```

```
	if(neutlist.size()+1>neutlist.max_size()) //checks to ensure that adding
one more neutron to the list will not exceede the max list size
	{
		cout<<"ERROR: There are too many neutrons for the list to sort.
Please turn off sorting"<<endl;
		exit(1); //if adding one more neutron to the list does exceede the
max list size, then the program exits to avoid a fatal error
	}

	if(neutlist.empty()) //if the list is empty, i simply add the information.
no need to sort
	{
		ns = ni;
		neutlist.push_front(ns); //adds info to list
	}
	else
	{
		ns = neutlist.front();
		ns1 = neutlist.back();
		if(ni.time<ns.time) //if the current time is less than the first
time in the list, the current time is added in front of that time
		{
			ns = ni;
			neutlist.push_front(ns);
		}
		else if(ni.time>ns1.time) //if the current time is greater than the
last time in the list, then the current time is added to the end. it is expected
that this case occurs the most
		{
			ns = ni;
			neutlist.push_back(ns);
		}
		else //looks for where to write time given that the previous
conditions failed
		{
			for(niter=neutlist.rbegin();niter!=neutlist.rend();niter++)
//iterates starting at the end of the list because it is more likely that the
current time will be written somewhere close to the end
			{
				if(niter->time<ni.time)
				{
					ns = ni;
					neutlist.insert(niter.base(),ns);
//niter.back() converts niter to a forward iterator, and then points to the niter-
- location. insert then places our current time one in front of the spot indicated
by niter.base()
					break;
				}
			}
		}
	}
}

void NeutFunc::PrintSorted() //prints all elements in the list, thus emptying the
list
```

```
{
	NeutInfo ns;

	while(!neutlist.empty()) //while the list isn't empty
	{
		ns = neutlist.front(); //gets the element in the front of the list
(should be smallest time value in the list)
		Output_Handler(ns); //sends time to output handler for file output
		neutlist.pop_front(); //deletes first element
	}
}
```

APPENDIX E

MCNP deck to find the appropriate window

```
INL/LANL Photofission Induced Neutron Coincidence/Multiplicity Measurements
C Scott Stewart June 2012
C 544g HEU with polyethylene moderation
C
C 0 in z is the floor, 0 in x and 0 in y define the point of the PND closest to the Varitron
on the Left
C at the cosmic ray shielding polyethylene edge furthest away from the opening for the
PND.
C
C Z = verticle up and down from ground
C X = distance perpendicular to the photon beam
C Y = distance parallel to the photon beam
C
C
C
C
C --------------------------------
C Cell Cards
C --------------------------------
C
C ==== Detector ====
100   1   -0.950    -200 +201 +217                imp:p=1 imp:n=1 $polyethylene block
101   4   -2.699    -202 +203 -219 +207             imp:p=1 imp:n=2 $side Al
102   5   -1.640    -203 +204 -216 +209             imp:p=1 imp:n=2 $side boroflex
103   3   -8.650    -204 +205 -216 +209             imp:p=1 imp:n=2 $side Cd
104   4   -2.699    -202 (-207:+219)              imp:p=1 imp:n=2 $top/bot Al
105   5   -1.640    -203 ((-219 +218):(+207 -208))     imp:p=1 imp:n=2 $top/bot boroflex
106   3   -8.650    -203 ((-218 +216):(+208 -209))     imp:p=1 imp:n=2 $top/bot cd
107   1   -0.950    -205 +221 -215 +209             imp:p=1 imp:n=2 $side poly
109   1   -0.950    -221 +206 -215 +209             imp:p=1 imp:n=4 $inner side
polyethylene (different for importance schemes)
108   1   -0.950    -206 +209 -210                imp:p=1 imp:n=2 $bot polyethylene plug
110   4   -1.350    -205 -216 +215                imp:p=1 imp:n=2 $electronics
111   4   -2.699    -206 ((-215 +214):(+210 -211))     imp:p=1 imp:n=2 $top/bot caps He3
112   2   3.294e-4  -206 ((-214 +213):(+211 -212))     imp:p=1 imp:n=2 $top/bot He3
REF
113   2   3.294e-4  -206 -213 +212                 imp:p=1 imp:n=8 $He3 active length
114   7   -0.00119 (-201 +202):(-302 +202):(-217 +202) imp:p=1 imp:n=2 $air in cutout
```

```
C
C ==== "Left" of Varitron Translated Detectors ====
C Detector 2
115 LIKE 100 BUT TRCL 100
116 LIKE 101 BUT TRCL 100
117 LIKE 102 BUT TRCL 100
118 LIKE 103 BUT TRCL 100
119 LIKE 104 BUT TRCL 100
120 LIKE 105 BUT TRCL 100
121 LIKE 106 BUT TRCL 100
122 LIKE 107 BUT TRCL 100
123 LIKE 108 BUT TRCL 100
124 LIKE 109 BUT TRCL 100
125 LIKE 110 BUT TRCL 100
126 LIKE 111 BUT TRCL 100
127 LIKE 112 BUT TRCL 100
128 LIKE 113 BUT TRCL 100
129 LIKE 114 BUT TRCL 100
C Detector 3
130 LIKE 100 BUT TRCL 101
131 LIKE 101 BUT TRCL 101
132 LIKE 102 BUT TRCL 101
133 LIKE 103 BUT TRCL 101
134 LIKE 104 BUT TRCL 101
135 LIKE 105 BUT TRCL 101
136 LIKE 106 BUT TRCL 101
137 LIKE 107 BUT TRCL 101
138 LIKE 108 BUT TRCL 101
139 LIKE 109 BUT TRCL 101
140 LIKE 110 BUT TRCL 101
141 LIKE 111 BUT TRCL 101
142 LIKE 112 BUT TRCL 101
143 LIKE 113 BUT TRCL 101
144 LIKE 114 BUT TRCL 101
C Detector 4
145 LIKE 100 BUT TRCL 102
146 LIKE 101 BUT TRCL 102
147 LIKE 102 BUT TRCL 102
148 LIKE 103 BUT TRCL 102
149 LIKE 104 BUT TRCL 102
150 LIKE 105 BUT TRCL 102
151 LIKE 106 BUT TRCL 102
152 LIKE 107 BUT TRCL 102
153 LIKE 108 BUT TRCL 102
```

```
154 LIKE 109 BUT TRCL 102
155 LIKE 110 BUT TRCL 102
156 LIKE 111 BUT TRCL 102
157 LIKE 112 BUT TRCL 102
158 LIKE 113 BUT TRCL 102
159 LIKE 114 BUT TRCL 102
C Detector 5
160 LIKE 100 BUT TRCL 103
161 LIKE 101 BUT TRCL 103
162 LIKE 102 BUT TRCL 103
163 LIKE 103 BUT TRCL 103
164 LIKE 104 BUT TRCL 103
165 LIKE 105 BUT TRCL 103
166 LIKE 106 BUT TRCL 103
167 LIKE 107 BUT TRCL 103
168 LIKE 108 BUT TRCL 103
169 LIKE 109 BUT TRCL 103
170 LIKE 110 BUT TRCL 103
171 LIKE 111 BUT TRCL 103
172 LIKE 112 BUT TRCL 103
173 LIKE 113 BUT TRCL 103
174 LIKE 114 BUT TRCL 103
C Detector 6
500 LIKE 100 BUT TRCL 104
501 LIKE 101 BUT TRCL 104
502 LIKE 102 BUT TRCL 104
503 LIKE 103 BUT TRCL 104
504 LIKE 104 BUT TRCL 104
505 LIKE 105 BUT TRCL 104
506 LIKE 106 BUT TRCL 104
507 LIKE 107 BUT TRCL 104
508 LIKE 108 BUT TRCL 104
509 LIKE 109 BUT TRCL 104
510 LIKE 110 BUT TRCL 104
511 LIKE 111 BUT TRCL 104
512 LIKE 112 BUT TRCL 104
513 LIKE 113 BUT TRCL 104
514 LIKE 114 BUT TRCL 104
C
C ==== "Right" of Varitron Detectors ====
C Detector 1
400 LIKE 100 BUT TRCL 110
401 LIKE 101 BUT TRCL 110
402 LIKE 102 BUT TRCL 110
```

403 LIKE 103 BUT TRCL 110
404 LIKE 104 BUT TRCL 110
405 LIKE 105 BUT TRCL 110
406 LIKE 106 BUT TRCL 110
407 LIKE 107 BUT TRCL 110
408 LIKE 108 BUT TRCL 110
409 LIKE 109 BUT TRCL 110
410 LIKE 110 BUT TRCL 110
411 LIKE 111 BUT TRCL 110
412 LIKE 112 BUT TRCL 110
413 LIKE 113 BUT TRCL 110
414 LIKE 114 BUT TRCL 110
C Detector 2
415 LIKE 100 BUT TRCL 111
416 LIKE 101 BUT TRCL 111
417 LIKE 102 BUT TRCL 111
418 LIKE 103 BUT TRCL 111
419 LIKE 104 BUT TRCL 111
420 LIKE 105 BUT TRCL 111
421 LIKE 106 BUT TRCL 111
422 LIKE 107 BUT TRCL 111
423 LIKE 108 BUT TRCL 111
424 LIKE 109 BUT TRCL 111
425 LIKE 110 BUT TRCL 111
426 LIKE 111 BUT TRCL 111
427 LIKE 112 BUT TRCL 111
428 LIKE 113 BUT TRCL 111
429 LIKE 114 BUT TRCL 111
C Detector 3
430 LIKE 100 BUT TRCL 112
431 LIKE 101 BUT TRCL 112
432 LIKE 102 BUT TRCL 112
433 LIKE 103 BUT TRCL 112
434 LIKE 104 BUT TRCL 112
435 LIKE 105 BUT TRCL 112
436 LIKE 106 BUT TRCL 112
437 LIKE 107 BUT TRCL 112
438 LIKE 108 BUT TRCL 112
439 LIKE 109 BUT TRCL 112
440 LIKE 110 BUT TRCL 112
441 LIKE 111 BUT TRCL 112
442 LIKE 112 BUT TRCL 112
443 LIKE 113 BUT TRCL 112
444 LIKE 114 BUT TRCL 112

C Detector 4
445 LIKE 100 BUT TRCL 113
446 LIKE 101 BUT TRCL 113
447 LIKE 102 BUT TRCL 113
448 LIKE 103 BUT TRCL 113
449 LIKE 104 BUT TRCL 113
450 LIKE 105 BUT TRCL 113
451 LIKE 106 BUT TRCL 113
452 LIKE 107 BUT TRCL 113
453 LIKE 108 BUT TRCL 113
454 LIKE 109 BUT TRCL 113
455 LIKE 110 BUT TRCL 113
456 LIKE 111 BUT TRCL 113
457 LIKE 112 BUT TRCL 113
458 LIKE 113 BUT TRCL 113
459 LIKE 114 BUT TRCL 113
C Detector 5
460 LIKE 100 BUT TRCL 114
461 LIKE 101 BUT TRCL 114
462 LIKE 102 BUT TRCL 114
463 LIKE 103 BUT TRCL 114
464 LIKE 104 BUT TRCL 114
465 LIKE 105 BUT TRCL 114
466 LIKE 106 BUT TRCL 114
467 LIKE 107 BUT TRCL 114
468 LIKE 108 BUT TRCL 114
469 LIKE 109 BUT TRCL 114
470 LIKE 110 BUT TRCL 114
471 LIKE 111 BUT TRCL 114
472 LIKE 112 BUT TRCL 114
473 LIKE 113 BUT TRCL 114
474 LIKE 114 BUT TRCL 114
C Detector 6
531 LIKE 100 BUT TRCL 115
516 LIKE 101 BUT TRCL 115
517 LIKE 102 BUT TRCL 115
518 LIKE 103 BUT TRCL 115
519 LIKE 104 BUT TRCL 115
520 LIKE 105 BUT TRCL 115
521 LIKE 106 BUT TRCL 115
522 LIKE 107 BUT TRCL 115
523 LIKE 108 BUT TRCL 115
524 LIKE 109 BUT TRCL 115
525 LIKE 110 BUT TRCL 115

```
526 LIKE 111 BUT TRCL 115
527 LIKE 112 BUT TRCL 115
528 LIKE 113 BUT TRCL 115
529 LIKE 114 BUT TRCL 115
C ==== Sample ====
187   1   -0.950    -253:-254:-255:-256            imp:p=2 imp:n=2 $2" polyethylene
outside box
188   1   -0.950    -252 +251 +250                 imp:p=2 imp:n=2 $polyethylene box
189   9   -19.35    -250:-251                     imp:p=4 imp:n=2 $2 HEU Zipper Plates
C
C ==== Floor ====
C 190   6   -2.250    -220                        imp:p=1 imp:n=1 $concrete floor
C 194   10  -7.859    -298                         imp:p=1 imp:n=1 $steel floor
C 195   6   -2.250    -297                        imp:p=1 imp:n=1 $concrete wall
C ==== Universe ====
191   7   -0.00119  -299 +301 +303 +253
              +252 +254 +255 +256             imp:p=2 imp:n=2 $air near detectors
192   7   -0.00119  +299 -300                    imp:p=1 imp:n=1 $air
193   0            +300                       imp:p=0 imp:n=0 $void

C --------------------------------
C Surface Cards
C --------------------------------
C
C ==== First "Left" of Varitron Detector ====
200   RPP   0.00 12.7   0.00 15.2   0.00 127         $Polyethylene Block
201   RCC   7.62 7.60 0.00   0.00 0.00 127      5.08 $Cutout
202   RCC   7.62 7.60 0.00   0.00 0.00 112.88   5.08 $outer Al
203   RCC   7.62 7.60 0.00   0.00 0.00 112.88   4.86 $outer Boroflex
204   RCC   7.62 7.60 0.00   0.00 0.00 112.88   3.91 $outer Cd
205   RCC   7.62 7.60 0.00   0.00 0.00 112.88   3.81 $outer Poly
221   RCC   7.62 7.60 0.00   0.00 0.00 112.88   2.54 $inner Poly
206   RCC   7.62 7.60 0.00   0.00 0.00 112.88   1.27 $outer He3
207   PZ    4.87   $ Al bot/Boroflex bot
208   PZ    5.82   $ Boroflex bot/Cd bot
209   PZ    5.92   $ Cd bot/Polyethylene bot plug
210   PZ    8.46   $ Polyethylene bot plug/He3 bot cap
211   PZ    12.06  $ He-3 bot cap/bot REF
212   PZ    13.83  $ bot REF/active he3
213   PZ    79.29  $ active he3/top REF
214   PZ    81.06  $ top REF/he3 top cap
215   PZ    84.66  $ he3 top cap/electronics
216   PZ    106.96 $ electronics/cd top
218   PZ    107.06 $ cd top/boroflex top
```

```
219   PZ    108.01 $ boroflex top/al
217   RPP   9.84 12.7   3.6 11.6   0.00 127 $air cutout
C
C ==== Sample ====
250   RPP   60.25 65.35   44.06 44.33   53.4 63.6    $1 HEU Zipper Plate (5.1x10.2x.27
cm)
251   RPP   60.25 65.35   46.87 47.14   53.4 63.6    $1 HEU Zipper Plate (5.1x10.2x.27
cm)
252   RPP   57.71 67.89   41.52 49.68   50.86 66.14  $1" polyethylene box
253   RPP   57.71 67.89   36.44 41.52   50.86 66.14  $front 2" poly
254   RPP   57.71 67.89   49.68 54.76   50.86 66.14  $back 2" poly
255   RPP   52.63 57.71   41.52 49.68   50.86 66.14  $left side 2" poly
256   RPP   67.89 72.97   41.52 49.68   50.86 66.14  $right side 2" poly
C
C ==== Translation Cutouts ====
301   RPP   0.00 12.80    0.00 91.2   0.00 127       $left detectors
302   RPP   12.7 12.80    0.00 15.2   0.00 127       $air gap in front of detectors
303   RPP   112.80 125.60 0.00 91.2   0.00 127       $right detectors
C
C ==== Room Features ====
C 220   RPP   -200 200   -200 316.28   -100 -30         $Concrete floor
C 297   RPP   216.28 316.28   -200 200   -30 200     $Conrete wall near detectors
C 298   RPP   -42.8 170.56   -60.445 151.645   -21 0 $Steel floor
299   RPP   0 125.60   0 91.2     0 127            $air near detectors
C
C ==== Universe ====
300   RPP   -200 200   -200 200   -200 200

C --------------------------------
C Data Cards
C --------------------------------
MODE N
PRINT
PHYS:N 20 J J J 21
C
C ==== Cutoff Card ====
CUT:N 800000 J 0 0
C
C ==== Translation Cards ====
TR100   0 15.2 0
TR101   0 30.4 0
TR102   0 45.6 0
TR103   0 60.8 0
TR104   0 76.0 0
```

```
TR110   125.60 0 0        -1  0  0  0  1  0  0  0  1
TR111   125.60 15.2 0      -1  0  0  0  1  0  0  0  1
TR112   125.60 30.4 0      -1  0  0  0  1  0  0  0  1
TR113   125.60 45.6 0      -1  0  0  0  1  0  0  0  1
TR114   125.60 60.8 0      -1  0  0  0  1  0  0  0  1
TR115   125.60 76.0 0      -1  0  0  0  1  0  0  0  1
C
C ==== Source Cards ====
SDEF PAR=SF X=d2 Y=d1 Z=d3 TME=d4 CEL=189
SI2   60.25 65.35
SP2   0 1
SI1   44.06 47.14
SP1   0 1
SI3   53.4 63.6
SP3   0 1
SI4   0 400
SP4   0 1
C
C ==== Tallies ====
F14:N (113 128 143 158 173 513
     413 428 443 458 473 528) $He-3 tubes
T14   2000 398I 800000
FM14  -1 2 103
SD14  1
C
F24:N 189 $HEU
T24   2000 398I 800000
FM24  -1 9 -6
SD24  1
FQ0  M T S
C
C ==== Material Cards ====
C
C m1 = polyethylene (0.95 g/cc)
m1    06000.70c   0.33333
      01001.70c   0.66667
mt1   poly.60t
C
C m2 = He-3 gas (0.001641 g/cc) at 297.2 K (75 degrees F) and 10 atm (3.294e20
atoms/cc)
m2    02003.70c   1.0
C
C m3 = cadmium (8.65 g/cc)
m3    48106.70c   0.0125
```

```
     48108.70c   0.0089
     48110.70c   0.1249
     48111.70c   0.1280
     48112.70c   0.2413
     48113.70c   0.1222
     48114.70c   0.2873
     48116.70c   0.0749
C
C m4 = aluminum (2.699 g/cc)
m4    13027.70c   1.0
C
C m5 = boroflex (1.64 g/cc)
m5    01001.70c   0.327
     05010.70c   0.281
     08016.70c   0.196
     14028.70c   0.181
     14029.70c   0.009
     14030.70c   0.006
C
C m6 = concrete (2.25 g/cc)
m6    6000.70c   1.60e-3
     8016.70c   4.350e-2
     11023.70c  5.50e-4
     13027.70c  1.60e-3
     14000.60c  1.52e-2
     16032.70c  5.00e-5
     20000.66c  3.10e-3
     26000.55c  3.80e-4
     1001.70c   7.606e-3
mt6   lwtr.60t
C
C m7 = air (0.00119 g/cc) at 297.2 K (75 degrees F)
m7    07014.70c   .7779
     07015.70c   .0029
     08016.70c   .2097
     18040.70c   .0093
     06000.70c   .0002
C
C m8 = DU (19 g/cc)
m8    92235.70c  -7.1640E-03
     92238.70c  -9.9284E-01
C
C m9 = ZPPR HEU (19.35 g/cc)
m9    92235.70c  -9.3300E-01
```

```
     92234.70c  -9.0600E-03
     92236.70c  -4.3800E-03
     92238.70c  -5.3560E-02
C
C m10 = Steel (7.859g/cc)
m10   06000.70c   .020
     26054.70c   .057
     26056.70c   .899
     26057.70c   .021
     26058.70c   .003
NPS 1E9
```