AN ABSTRACT OF THE THESIS OF

Noah C. Harris for the degree of Master of Science in Radiation Health Physics presented on October 30, 2020.

Title: Developing Software Tools to Characterize Electrochemical Loss Scenarios via Gamma Detection for Various Measurement Points, Source Geometries, and Process Times.

Abstract approved: ____

Haori Yang

Faithful modeling of the expected gamma signals inside an electrochemical facility at various key measurement points is important for understanding what detection limits are available for the next generation of safeguards technologies. Gamma Detector Response and Analysis Software (GADRAS) and the Separation Safeguards Performance Model (SSPM) were used to build software tools for predicting the difference in radiation signatures between normal plant use and plutonium diversion scenarios, for various operational configurations. These tools allow for automated analysis of the large data sets generated by the SSPM, producing tables and charts for both total gamma counts over time, and channel-by-channel gamma spectrum analysis for specified time-slices. Preliminary application of this analysis suite shows a 99% confidence in detecting a 0.2 SQ protracted electrorefiner diversion over the course of one year, and a 99% confidence in detecting a 1.4 SQ protracted transuranic diversion over the course of one year.

Developing Software Tools to Characterize Electrochemical Loss Scenarios via Gamma Detection for Various Measurement Points, Source Geometries, and Process Times

by

Noah C. Harris

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented October 30, 2020

Commencement June 2021

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to the professors in the school of nuclear science and engineering at Oregon State University, especially his research advisor Dr. Haori Yang, and to the researchers at Sandia National Laboratories, primarily Dr. Ben Cipiti, Dr. Nathan Shoman, and Dr. Gregory Thoreson for their guidance and mentorship. He would also like to express gratitude to his wonderful wife Breanna and son Nathaniel for their constant support and motivation throughout the program. Finally, he would like to thank his mother, Dana Harris, for always pushing him to do his best and to keep learning.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF APPENDIX FIGURES

Dedicated to Dana Harris

1. INTRODUCTION

The purpose of this document is to act as a guide to those interested in the limits of gamma detection as a tool for nuclear bulk material accounting in a pyroprocessing facility, including the use of software developed during this project expressly for this purpose. For those interested in the installation and use of this software, please refer to section 4.2 of this document.

This work is being carried out as part of an agreement, known as a “123 Agreement”, between the United States government and South Korea (Republic of Korea) to develop energy-related nuclear technology. With the development of new industrial processes, electrochemical processes in this case, comes the requirement of new safeguards. This work will hopefully aid someone in designing the systems for such a facility, and give a reference for what kinds of uncertainties may be expected in a gamma non-destructive analysis (NDA) module within a larger overall electrochemical materials accounting system.

Bulk accounting, such as required in an aqueous or electrochemical facility, is more difficult than traditional nuclear accounting, in that the material is harder to track than discrete quantities such as drums, boxes, etc. In addition, an electrochemical facility faces numerous challenges compared to aqueous facilities including a lack of flush-outs, material holdup, lack of input accountability and representative salt samples (due to salt inhomogeneity), and others [3]. These factors, coupled with the international interest in constructing electrochemical facilities in the near feature, has led to a push for the development of novel techniques for safeguarding these facilities. The United States Department of Energy has been leading this charge with their Material Protection Accounting and Control Technologies (MPACT) campaign, in direct support of the aforementioned “123 Agreement” with the South Korean government [8]. MPACT is currently pursuing a set of goals called Milestone 2020 which is the grouping into which the work of this thesis falls.

The ultimate goal of the MPACT campaign is the creation of Virtual Facility Distributed Test Bed which is meant bring together experimental and modeling capabilities across the national laboratory and university complex to provide a one-stop-shop for advanced Safeguards and Security by Design (SSBD). Experimental testing alone of safeguards and security technologies would be cost prohibitive, but testbeds and laboratory processing facilities with safeguards measurement opportunities, coupled with modeling and simulation, provide the ability to generate modern, efficient safeguards and security systems for new facilities [3].

The Virtual Test Bed concept is built upon the Separation and Safeguards Performance Model (SSPM), which is a virtual model of an electrochemical processing facility created by Sandia National Laboratory based off of models developed at Argonne National Laboratory, which also the birthplace of the electrochemical (sometimes called pyroprocessing) approach to nuclear reprocessing. The work of this thesis is to take data from the SSPM and use radiation transport codes to simulate detector response. By looking at differences between the data for loss scenarios vs normal facility operations, we can explore the efficacy of using gamma detectors to detect those loss scenarios. Gamma detection will be just one component of the virtual safeguards test bed, and MPACT's hope is that the unification of many simultaneous approaches will lead to a truly robust safeguards outcome.

Preliminary efforts towards this goal have already been documented and submitted in an article for the Journal of Nuclear Materials Management [10], and this document is meant to be an elaboration of those efforts.

2. LITERATURE REVIEW

Although electrochemical (pyroprocessing) of spent fuel is a relatively new technology, only recently being seriously developed at Argonne since the late 2000's, and without any current large-scale facilities, there is already a wealth of nascent information on the subject. The technology is currently being considered for application by the Republic of Korea. However, before that happens there must be advancement in the safeguards technologies pertaining to any proposed pyroprocessing facility. The US government's Department of Energy has an active and vigorous research campaign known as MPACT which is exploring a plethora of avenues for the advancement of technologies which may be useful in this area. MPACT's efforts are beginning to pay off, and some of the technologies they have been fostering, such as microcalorimetry, may have an impact far beyond

2.1 "123 AGREEMENT"

The political impetus for this particular project's funding springs from a treaty between the governments of the United States and the Republic of Korea (ROK), who have entered into a cooperative agreement to help advance mutual interests in the realm of nuclear power, known colloquially as a "123 Agreement". The two countries agreed in 2011 to a 10-year joint study on pyroprocessing, to examine ways to deal with the ROK's spent fuel challenge.

As a result of developing its nuclear power sector, The ROK of course finds itself with a growing inventory of spent fuel. Because it is a country which has a very high population density, creative solutions must be employed [18]. The Korean Atomic Energy Research Institute (KAERI) and the ROK government have been promoting pyroprocessing as a technology that could reduce the volume of high-level radioactive waste requiring deep disposal by a factor of up to 20, and the underground area required for a geologic repository by a factor of up to 100 [19]. In a bid on future reprocessing technology, KAERI has opted to develop the "Advanced spent fuel Conditioning Process" (ACP), and has even built a test facility, the ACP-Facility or ACPF, in the basement of the Irradiated Material Examination Facility (IMEF) at KAERI.

The subject of this thesis will be part of the unified effort in the development of proper safeguards for any facilities implementing the ACP, and will be in support of the objective of the agreement between the US and the ROK, as one of many approaches constituting the US DOE's MPACT campaign. In exchange for the development of this safeguards technology, the agreement will help strengthen U.S. nonproliferation policy by ensuring South Korean commitment, building protection mechanisms, and signaling to the global community that the United States is prepared to work constructively with strong advocates and proponents of nonproliferation [22].

2.2 ELECTROCHEMICAL REPROCESSING (PYROPROCESSING)

Electrochemical processing (pyroprocessing) is a new reprocessing technique first developed at Argonne National Lab around 2010. Since that time, the details of this technique have been well researched and worked out, and developing the safeguards technology remains the final piece of the puzzle so that these types of facilities may be brought online, e.g. in the ROK.

Pyroprocessing facilities use molten salts and electrochemical operations, known as electrorefining or electrowinning, to separate actinides from spent nuclear fuel. The technology was originally designed for processing metallic fuels but has also been adapted for use with oxide fuels. Variation in the flowsheet design, beyond what is used as the basis for this work, is certainly possible and depends somewhat on operator and country needs, engineering issues, and any safeguards or security considerations. A referenced article [20] provides more detail on flowsheet options and describes the AMPYRE (Argonne Model for Pyrochemical Recycling) and DyER (Dynamic Electrorefiner) models. Additional information on electrochemical flowsheets can be found in references [6] and [25], but a brief description is provided here:

The flowsheet is based on a throughput of 100 metric tons of spent nuclear fuel per year (MT/yr). Fuel assemblies are initially disassembled in a hot cell, and the fuel rods are de-clad from their protective alloy. The spent nuclear fuel (SNF) is chopped into smaller sized particles and loaded into porous metal baskets for processing.

Oxide fuel first goes through an oxide reduction (OR) step to convert most of the fuel to a metallic form while liberating oxygen from the oxide fuel form. During this step some of the trapped fission product gases in the fuel are also outgassed and captured. For a metal fuel reprocessing operation, this step is not required. This oxide reduction is accomplished by lowering the baskets into a molten salt and applying an electric potential across the basket and a cathode. As current is passed, the metal oxides reduce to metals, and the oxide ions are released into the molten salt and transported to the cathode where they are oxidized to produce oxygen gas.

The newly metallic fuel in the baskets then gets transferred to an electrorefiner (ER) vessel, where the baskets are lowered into another molten salt. As before, an electric potential is applied between the basket and a cathode—this potential drives fuel into the salt phase, and extracts actinides in the salt onto the cathode. Separate cathodes are used for uranium (U) recovery and combined uranium and transuranics (UTRU) recovery. The products are then melted and poured to consolidate them into an ingot form for storage and/or future use for fuel fabrication.

The current flowsheet we're using in the SSPM assumes the use of two cathodes that operate at the same time in the ER vessel. The U cathode is a solid cathode that removes U only, as the material builds up as dendrites on the cathode, and then the cathode is removed and scraped to get the product. The UTRU cathode can also be solid, is usually a liquid cadmium cathode. This is often a crucible containing liquid cadmium that is lowered into the ER salt. When a potential is applied, the U and transuranics will transport into the cadmium in a metallic phase. But the key point is that both the U and UTRU extraction can occur at the same time in the same vessel.

The U cathode processor is a furnace that melts down the U dendrites and distills off the salt—the salt gets retuned to the ER vessel. The leftover U ingot then becomes a product of waste form. The UTRU cathode processor is also a furnace that distills off first the salt, then the cadmium, so that the UTRU is remaining and cast into an ingot.

The UTRU drawdown is a separate process. The purpose of drawdown ultimately is to remove the rare earth fission products. But in order to do that, residual UTRU must first be removed. So this process removes the UTRU using electrolysis, and then the same electrolysis process is used to remove the rare earths. The leftover salt, which is mostly clean of fission products, is returned to the ER vessel. A certain amount of the ER salt needs to be processed continuously to maintain rare earth content below some maximum threshold. This is needed so that the U/TRU product does not contain too many fission product poisons since there will always be some small amount of rare earths going into the UTRU product.

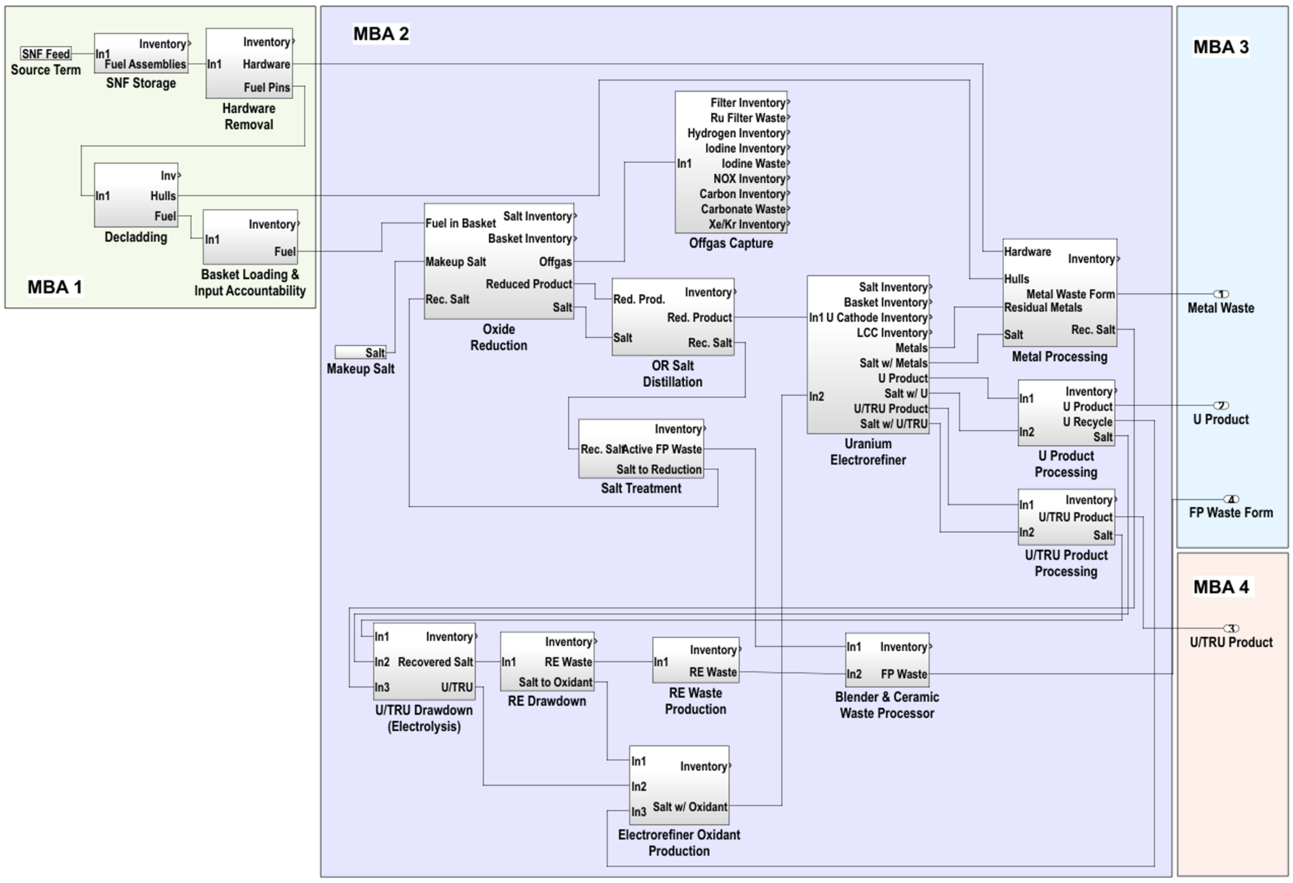


Fig. 2.1: Diagram of the SSPM electrochemical flow chart. [2]

The removed UTRU metal from drawdown is not a good quality for fuel fabrication, because it contains a high amount of rare earths, so it gets recycled back to the ER salt. The best way to do that is to convert it to its oxidant form UTRU-Cl3. The UTRU just gets added back in to prevent more waste. There are many variants on this process design. The UTRU metal could also be returned to the baskets to be recycled back in. And the plant could use an external source of UCl3 for the oxidant rather than recycling. That's somewhat of a cost-benefit analysis.

Separate fission product recovery processes are used to remove active metals from the OR salt and rare earth fission products from the ER salt. Residual noble metals stay in the basket and are recovered and placed into a metal waste form along with cladding and assembly hardware.

2.3 NEUP UNIVERSITY SUPPORT FOR MPACT 2020 MILESTONE

The work of this thesis follows along with the second fiscal of year of work (task 4) of three years total outlined in the technical narrative for the NEUP support project for MPACT [8]. This period of time is meant to be used to develop a system for simulating radiation-based signatures using the SSPM data. Thus, a greater part of focus has been spent on creating a robust toolset for accomplishing this task, including an automated pipeline for both GADRAS-integration and subsequent analysis. Although the use of these tools for analysis is not meant to be performed until the third fiscal year of the project (FY 21), some time has still been spent on analyzing the data produced, primarily so that the output of the tools may be judged and thus to establish the usefulness and effectiveness of the software pipeline.

2.4 MPACT

Material Protection Accounting and Control Technologies (MPACT) is a United States Department of Energy campaign whose mission is to develop innovative technologies and analysis tools to enable next generation nuclear materials management for existing and future U.S. nuclear fuel cycles, to manage and minimize proliferation and terrorism risk. [21]

The scope of this work includes the operator's Material Control and Accountancy (MC&A) system and Physical Protection System (PPS). The purpose of these systems is to provide assurance that nuclear material is adequately protected and detect and respond to theft or sabotage by sub-national groups. International safeguards measurements and verification are not the subject of this work, but many of the measurement technologies and analyses presented here can also be applied for international safeguards. [3] For example, the work of MPACT is being used to directly support the terms of the "123 Agreement" between the US and ROK detailed above [8].

As a key part of their mission, the MPACT group is seeking to develop and and demonstrate advanced material controls and accounting technologies to fill important gaps and keep pace with upcoming technologies. This is accomplished via a comprehensive diversified group working across many different domains, both in the national lab system and at universities across the country. This is necessary due to the sheer number of disciplines involved to produce the instrumentation that holds promise for next-generation safeguards applications. Examples of the instrumentation groups currently working on MPACT projects include:

(1) The microcalorimetry group at Los Alamos National Lab (LANL) is currently working on developing an ultra-high resolution gamma spectroscopy nondestructive method of detecting changes in concentration of key isotopes throughout the process. The SOFIA (Spectrometer Optimized for Facility Integrated Applications) microcalorimeter instrument was used to measure spent fuel samples from real-world facilities and purified U and Pu reference materials. The enhanced energy resolution of microcalorimetry as compared to germanium detectors was found to improve NDA capabilities by increasing peak-to-background ratio and ability to resolve closely-spaced gamma ray energy peaks. The microcalorimeter measurements can potentially measure certain actinide and fission product peaks with 1% counting statistics [5][3].

(2) High Dose Neutron Detection (LANL): Modeling work was carried out using the High Dose Neutron Detector (HDND) to determine detection of small amounts of TRU in the U ingot in the case of diversion or off-normal operation.[20][21] The calculations found that a 0.005% weight contamination of plutonium (Pu) in the U ingot would be detectable with greater than 95% detection probability using the HDND with a counting time as low as 10 minutes [16][7].

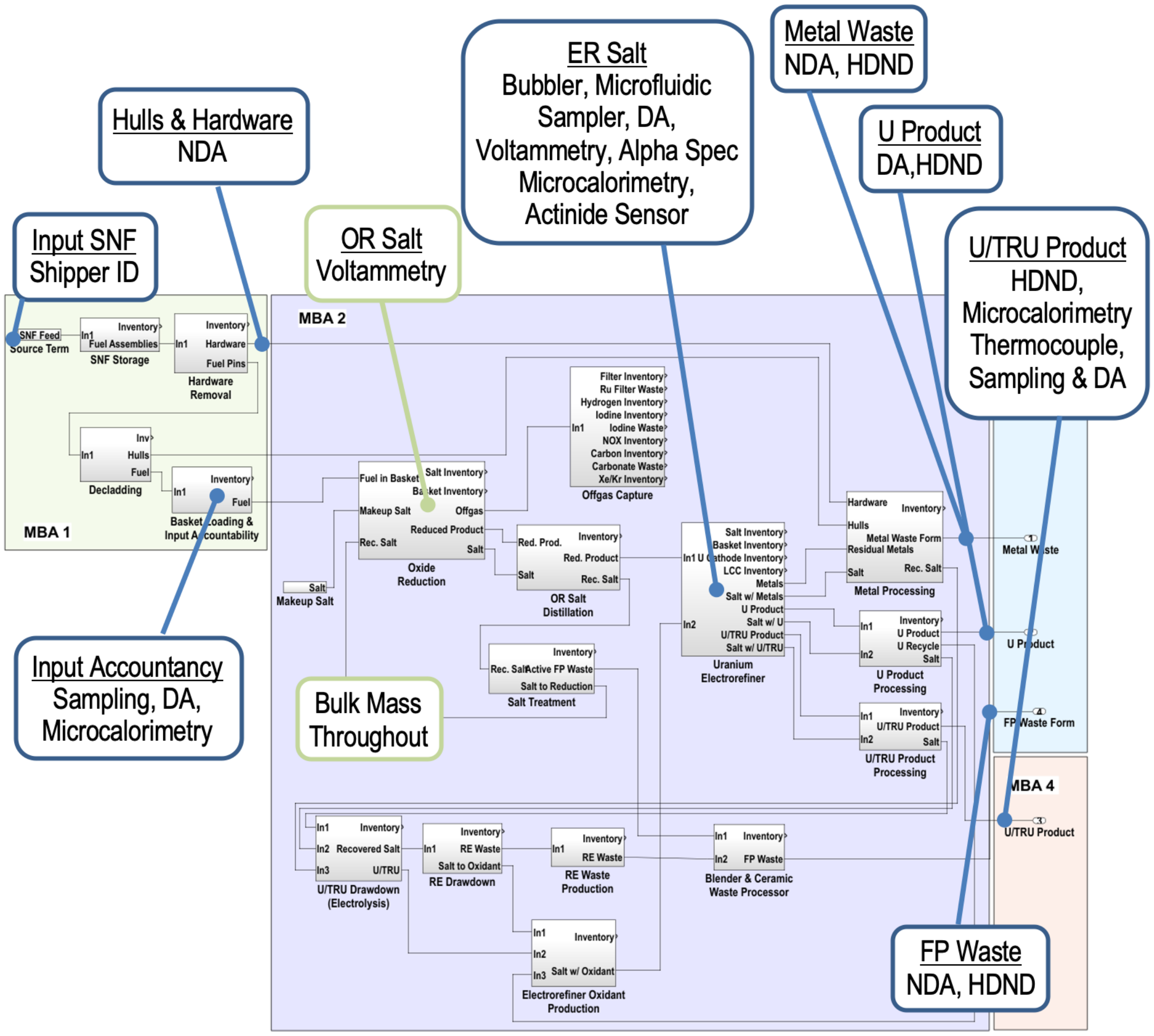


Fig. 2.2: Diagram of the SSPM with various projected applications for MPACT instrumentation projects. [2]

(3) Actinide sensors and voltammetry are being developed both at Idaho National Laboratory (INL) and by our collaborators at Virginia Tech. ER voltammetry appears the most promising for an in-situ measurement of actinides and other properties of the salt with a measurement uncertainty less than 1%. [27]

(4) A microfluidic sampler was developed by Argonne National Laboratory (ANL) for generation of uniform salt samples for analysis. Acquiring representative salt samples was identified as a key challenge area for electrochemical safeguards, and this technology serves to fulfill that need. In addition, commercial scale plants will benefit from an automated way to collect samples. The microfluidic salt sampler has demonstrated capabilities for high-throughput sample generation with coupled analysis. [9][17][3].

There are many measurement points available in the SSPM schema, and it is important to understand how traditional NDA techniques fit into the larger picture. Microcalorimetry and High Dose Neutron Detector (HDND) are novel NDA technologies being developed by other research groups for use at the electrorefiner and product ingots. As a preview of the results, it was found that for the two diversion scenarios explored in this paper, the metal waste and uranium/transuranic product ingot NDA key measurement points (KMPs) had the most predictive power, and the fission product waste was surprisingly less helpful.

2.5 VIRTUAL FACILITY DISTRIBUTED TEST BED

One of the primary deliverables of the MPACT campaign is the Virtual Facility Distributed Test Bed, which is meant to incorporate all the of the simulation-based advances by the MPACT group into a unified virtual test environment. The focus of this effort will be the SSPM, and most of the advances by the group will be incorporated into the test bed by adding onto the SSPM software ecosystem, or by informing the inputs to the test bed system via physical laboratory testing environments. The Virtual Facility Distributed Test Bed is also capable of modeling physical plant security by means of Unity models built by researchers at Los Alamos [3] These elements come together to provide a unified Security and Safeguards By Design (SSBD) simulation capabilities. The work of this project, as outlined in the NEUP technical narrative [8], is intended to be integrated into the Virtual Facility Distributed Test Bed by providing analysis support for the NDA component of the safeguards approach. Integrating this work into the SSPM will require some additional software beyond MATLAB running on a machine. A GADRAS installation will be required of course, as well as Microsoft Visual Studio, in which the API calls, logic, and interface are implemented. This analysis suite is also dependent on Microsoft Excel. The hope is that the flexibility and automation of analysis is enough to justify the larger software overhead.

Other electrochemical measurements should prove easier to integrate into the virtual test bed. An example of work which will fit more cleanly into the test bed as a MATLAB Simulink block is being carried out by collaborators at Virginia Tech, who are working on an electrode sensor model for the electrorefiner unit. [8][27]

3. MATERIALS AND METHODS

This section is meant to enumerate the different software used for manipulating data from the SSPM, running transport and detector response codes, and ultimately analyzing the results. Additionally there is information about the statistical methods used for quantitative analysis of the results, and there is information about the reasoning that went into choosing the particular subset of SSPM and geometrical parameters that were decided upon for this project, given the massive size of possible parameter combinations.

3.1 SOFTWARE ECOSYSTEM

The bulk of the work done on this project is in the manipulation and processing of SSPM data using various software platforms and methods. Here is described the various softwares used for the project and how they interoperate with one another.

3.1.1 MATLAB

The SSPM is built in Simulink, which is essentially a very complex plugin for the main MATLAB program, which is comprised of the workspace and scripting, and allows us to examine the isotopics sampled from various KMPs in the signal flows of the SSPM. MATLAB also provides the ability to write to excel files, which is how data is moved out of the SSPM to be processed. A script ‘copyToExcel.m’ (Appendix C) was written, which allows copying of data from workspace variables to an excel file which then allows for further manipulation by the Visual Studio programs. In order to change which data run one is reading from, change the ‘load()’ call at the top of the script to reflect which ‘.mat’ file to pull data from. To change which time-slices are sampled from the data series one can change the time slices enumerated at the start of each KMP’s respective section. It must be verified that the Excel rows allocated for writing data are numerous enough to house the number of time slices entered. This is determined by the constants in the for loop structures of ‘copyToExcel.m’. Lastly, this script can be set to not trigger the C# program by commenting out line 78 the bottom of the script. This changes a flag variable which will be read by the Visual Studio C# add-in. If the flag is ‘createGeometry’, then our C# add-in will run, if it is 0 the add-in will not run.

3.1.2 SSPM

The Separation Safeguards Performance Model was developed by Ben Cipiti et. al. at Sandia National Laboratory for the purposes of modeling a pyroprocessing plant. The model was developed in Simulink, which is a MATLAB-based graphical programming environment for modeling, simulating and analyzing multi-domain dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries.

Simulink’s block layout means that the model can easily be understood as a flow chart between the different functional areas of the plant. This is very useful for navigating the layers of complexity which are well encapsulated by this graphical programming language.

The SSPM models movement of material through a pyroprocessing plant. This starts with spent nuclear fuel which is then shredded and dissolved into molten salts before running through first an OR, then an electrorefiner, then various tail-end processes.

3.1.3 EXCEL

Excel is an excellent way to store the data generated by the SSPM software because it is a fast and convenient way to manipulate the many 1676 element vectors, as well as providing for graphing needs with a robust interface and lots of customizability. It also plays along nicely with Visual Studio via the Visual Studio Tools for Office (VSTO) interface. Overall it is a good piece of connective and analytical software in the pipeline.

The workflow for this project entails taking data from the Visual Studio (VS) program, and shuffling the data around with Excel to develop the graphs by creating a chart and specifying the proper data series, and other chart customization options. Then new data can be swapped in keeping the same chart template, which indicates the difference between control and loss scenarios with a dotted vs solid line on the graphs respectively.

3.1.4 VISUAL STUDIO

Microsoft Visual Studio (VS) is where the software heavy lifting for this project is done. VS allows writing to Xtended Markup Language (XML) to create '.1dm' GADRAS files after data is exported from the SSPM, using an Excel VSTO add-in. It also allows use of the GADRAS C# Applications Programming Interface (API) for running the '.1dm' files through transport simulations (which creates a '.pcf' file). Then VS is used to read from the '.pcf' files and move data to an intermediary file, which can finally be imported back into Excel (via Excel add-in) to make charts, analyze the outputs, etc. The user interface which controls the behavior of the analysis software is accomplished by means of the VS command-line tools and Windows forms.

3.1.5 GADRAS

GAmma Detector Response and Analysis Software (GADRAS), is a software suite developed at Sandia National Lab with capabilities related to radiation transport and detector response. Its primary function is the simulation of gamma-ray and neutron signals from sources with various geometries and shielding configurations. This data can then be included as part of an 'inject' calculation, which incorporates information about the detector and its spatial relationship to the source, as well as information about geographical location and coarse-grained environmental properties, such as geographical location, which can help determine potassium-40 and cosmic ray background contributions.

GADRAS also contains an analysis suite which can be applied to the pre-computed spectra ('.pcf') files generated as previously described. These tools include single-regression analysis, multiple-regression analysis, radionuclide identification, and special-nuclear-material (SNM) analysis, along with a few others which are not relevant to this project such as flux computation and full-spectrum-analysis (FSA) model fitting. These tools may be useful in providing insight on the spectra produced by feeding SSPM data into GADRAS.

GADRAS provides a C# API which can be used to turn isotopics data into a GADRAS '.1dm' model file. This is a file which specifies composition and geometry of the material to be placed near a simulated radiation detector.

3.2 METHODS

The main work of this project has consisted of creating a software analysis pipeline whereby isotopic data created by the SSPM is processed and run through GADRAS to generate simulated detector responses. These responses may then be subject to various analysis and evaluated for their usefulness with regards to electrochemical safeguards applications.

Because the SSPM represents a single year in 6480 hours of plant operations, and for each hour represents the isotopic makeup in a given plant unit by a 1676-element vector, the amount of information generated is quite formidable. And this is only for a single run of the simulation. There are many different loss parameters that may be tweaked including diversion location, diversion length, fuel enrichment, fuel burnup, and fuel cooling time. The dual task of this project is to first strategize the best way to tackle this huge parameter space on the macro level and then secondly using the software to automate analysis on the micro (individual simulation) level.

3.2.1 RUNNING AND MODIFYING SSPM

After opening MATLAB, the SSPM Simulink file may be opened using the file hierarchy on the left hand side. This will open a Simulink window with the electrochemical flowchart represented in Simulink's visual block-based programming. Before a simulation may be run, model parameters must first be selected. This is accomplished by opening 'SSPMeChem_GUI_MPACTrev1.mlapp' from the main MATLAB window in the same way the Simulink file may be opened. This will bring up a GUI editor window. In this window, the "Run" button is clicked to open the parameter selection screen. Here one is able to select various parameters including fuel enrichment and burnup, cooling time, simulation time, diversion location, diversion period, diversion percentage, and there is even the potential to create a custom fuel swap sequence if desired, although that feature is not used for the purposes of this thesis. Now clicking “Generate Parameters for Standalone Run” will generate the parameters, which will be stored in the MATLAB workspace in the “ModelParams” variable.

Now that the parameters have been generated, the SSPM Simulink file will be able to use them for an electrochemical simulation. Returning to the Simulink screen, the green “Run” button at the top may be pressed to initialize and run the simulation. The built-in output of the SSPM can be found in the MATLAB workspace, as the SSPM will generate workspace variables during its execution.

3.2.2 MEASURING DIVERTED SPECIAL NUCLEAR MATERIAL (SNM)

The amount of plutonium or uranium diverted over the course of a simulation may be measured by sampling the negative signal inside a given diversion block, and sending it through this chain of Simulink blocks: Selector -> Sum -> Integrator -> Scope. This chain first selects only the plutonium isotopes, then adds them together (we want to consider the diversion of all Pu isotopes), then converts the per-time signal into a total mass (in kg) via the integrator, then displays the data, which is how the figures B.5-B.12 were obtained. Appendix figure B.1 shows the Simulink setup for UTRU diversion detection, along with the preferences for the selector block.

3.2.3 OBTAINING AND PROCESSING ISOTOPIC VECTORS

Capturing data from the SSPM is a straightforward process. First in the block diagram, a toWorkspace block is added (using the 3-D array setting with a sample time of 1 hr) and given whatever variable name is desired. Then a signal line is attached to a preexisting signal line in the diagram to sample the value at that point.

The simulation is then run. Once the simulation is paused or stopped, the workspace variables will be updated and data values will be available for navigation. Multiple variables may be selected and saved as a group, making saving and reopening simulation records much easier. As many ‘toWorkspace’ blocks may be placed as desired, because even approximately 20 blocks does not significantly slow down the simulation. Although it is recommended that a user ensures that their sample time is not set to an extremely small value, which may indeed result in a slowdown.

Once this data is available in a MATLAB workspace, the ‘copyToExcel.m’ (found in appendix C) may be used, which goes through the various KMP data series and writes the vectors corresponding to a set of time-slices, also specified in that script, to an excel file (isotopics.xlsx) for further processing.

3.2.4 WRITNG TO 1DM File

SSPM data may be provided to GADRAS via an XML file with the extension .1dm. This file tells GADRAS how to construct a 1-dimensional radiation transport model, with arbitrarily defined custom materials comprised of any number of specified structural materials and radionuclides. Various materials may be combined into a single model, separated into distinct shells (in the spherical case), shown in the cross-sectional view of the “Model” tab. This allows for the modeling of shielded sources, in which the source may be a homogeneous mixture of various different radionuclides of varying compositional percentage.

As mentioned before, the data displayed in the ‘Model’ tab is summarized in an XML file with extension .1dm, which may be populated with data from the excel file into which SSPM data is copied by using the C# Add-In described above. The specifics of this XML files are designated Official-Use-Only and thus will not be disclosed, but suffice to say that isotopes and their corresponding mass-fractions are provided for each material layer (which are then normalized by GADRAS and as such may be used for any specified mass of the resultant custom material).

Three different source term geometries have been employed for this analysis: spherical, cylindrical, and slab-shaped. The cylindrical configuration is a long skinny rod, and is not a true cylinder but approximated in GADRAS using a long skinny rectangular prism with square cross section. The slab configuration is also a rectangular prism, with two of the dimensions having substantial length, and the third being quite small compared to the first two. All three of these geometrical configurations are lead shielded, with a concentric outer lead layer that matches each particular geometry. Different thicknesses of lead shielding are used at different KMP's, to ensure optimal detector response. Included in appendix B.2 and B.3 are pictures of the various source term geometries in the GADRAS model window.

These parameters are chosen via a dialogue window which appears as part of the Excel add-in (called as isotopics.xlsx opens) after the MATLAB script writes the isotopic vectors to the spreadsheet. Additionally, this dialogue allows the user to select parameters for the GADRAS transport and inject calculations, such as detector type, dimensions, distance from source, etc.

3.2.5 GADRAS TRANSPORT CALCULATIONS

After SSPM data has been written to the '.1dm' files, the Excel add-in will call another binary which uses the specified parameters to initiate a transport calculation which produces a .gam files in that same directory. These files contain information about the photon and neutron energies and fluxes for that particular model. These .gam files will be used as the source terms for GADRAS "Inject" calculations, in which detector response is simulated.

The result of a GADRAS inject calculation is a pre-computed-spectrum '.pcf' file, which appears in the directory belonging to the GADRAS detector with which the inject calculation was run. These '.pcf' files may then be analyzed via the GADRAS "Analysis" tab. Or in the case of the automated system developed for this project, a third Visual Studio program will write from the .pcf file to a intermediary data file, which will then ultimately be read by a second GADRAS Excel Add-In representing the final piece of software and terminal point of analysis for this pipeline.

3.2.6 DATA ANALYSIS

3.2.6.1 TOTAL GAMMA CPS

A main avenue of analysis that has been pursued on this project has been the total gamma counts-per-second (CPS) for NDA radiation detection at the various KMPs. At the relevant KMPs there is a detectable difference at the point of an abrupt diversion, or an increasing difference over the course of a longer diversion.

Once the GADRAS transport and inject calculations have been run, the output is stored in a '.pcf' filetype. This can be read by GADRAS or an associated program also developed at Sandia known as Cambio. GADRAS displays the '.pcf' data in table form. In order to create the final graphs and have an easier time representing the data visually, the '.pcf' data is put back into Excel to create the charts. This is accomplished by once again leveraging the GADRAS C# API using the ReadHeader API example. This example project is used as the basis for the 'GammaAnalysis' Visual Studio project, which as part of its capability writes a specified '.pcf' file's records' total counts to an intermediary file to be used by a final Excel Add-In (via totalGamma.xlsx) for the final analysis, which consists mainly in charting the total gamma counts for various KMPs over the course of an operational year of the facility.

Because multiple KMPs are dealt with on a single chart, the most optimal solution to the problem of visual noise is to represent all measurements on the same KMP with a single color, and to make the control scenario a dotted line whereas the diversion scenario would be a solid line. A sample of the charts output by this process can be found in appendix A (figures A.1-A.5).

In order to generate detection probabilities with associated confidence intervals, the null hypothesis will be tested. That is to say, given a background count (the control gamma CPS) what additional (or in our case subtractive) amount of counts will give the desired confidence that we have facility misuse at play. This process is carried out in greater detail in section 4.5.2.

3.2.6.2 PEAK ANALYSIS

An alternative to using total gamma count as the primary metric for our NDA analysis is the use of gamma peak ratio analysis. This technique distinguishes between different energy gammas and is concerned with how the energy spectrum changes in response to diversion scenarios in the electrochemical facility.

The 'GammaAnalysis' component of the software tools also has the capability to initiate a channel-by-channel analysis of a specified gamma spectrum in addition to its total count functionality. The difference here is that the spectrum is being examined for only a single KMP at a single time-slice. Thus, the logical flow of using the software consists in using the total gamma functionality of 'GammaAnalysis' to identify good KMP/time-slice candidates for diversion discrepancy, which can then be explored individually using the channel-by-channel spectrum analysis capability. A figure of a chart generated by the peak analysis mode can be found in appendix B.4.

3.2.7 PARAMETER SPACE SELECTION

As stated in the introduction to this section. The possible parameter space of loss scenarios generated by the SSPM is quite large, and must be reasonably reduced if there is to be a chance of making some sort of comprehensive judgement in the time span allotted for the project. A good initial reduction of this parameter space can be achieved by looking at which diversion scenarios result in a substantial amount of plutonium being lost from the facility. An initial survey for this thesis can be found in section 4.4 and appendix figures B.5-B.12.

3.2.7.1 PROTRACTED VS. ABRUPT LOSS SCENARIOS

Given the results presented in graphs such as figure A.3, it can be shown that this method is extremely effective at detecting abrupt loss scenarios for common diversion points such as the ER, UTRU processing, etc. The confidence interval on these detections is effectively 100% for testing of the null hypothesis. This is very good news for the use of this technique as a safeguard against abrupt diversion. However, safeguards are only as effective as their weakest link, and the possibility of a protracted diversion event is much more pernicious and likely given a malicious state-sponsored misuse of a facility. Thus the decision has been made to focus the main share of this project on protracted loss scenarios and their associated probabilities of detection, which seems to be a more interesting and salient focus.

3.2.7.2 KEY MEASUREMENT POINTS

As covered in section 2.3, Dr. Cipiti of MPACT has laid out his predictions for which KMPs will benefit most from NDA gamma techniques. Figure 2.2 shows that it is predicted that the most benefit will be obtained by using NDA at the metal waste and fission product waste KMPs. As revealed in section 4, the U and UTRU product processing may also be prime candidates for NDA. While the results of this project support the recommendation to use the metal waste KMP, it was found, surprisingly, that the fission product waste KMP turned out to be far less useful than anticipated.

3.2.7.3 FUEL CHARACTERISTICS

The SSPM parameter GUI (SSPMeChem_GUI_MPACTrev1.mlapp) offers 9 different possible fuel burnup/enrichment combinations. For 33 GWd/MTHM: 2.6%, 3.3% and 4% enrichment. For 45 GWd/MTHM: 3.3%, 3% and 4.7% enrichment. And for 60 GWd/MTHM: 4.03%, 4.73% and 5.43% enrichment. Alongside this there are 5 different cooling times: 1 year, 5 years, 10 years, 25 years, and 50 years. Together, these parameters allow for 45 different combinations of fuel input parameters. Given the time it takes to run the SSPM, it was opted to simplify this complexity by collapsing the burnup/enrichment options into only three. This parameter was chosen because of the simulations I had already run, the smallest output variance was found when changing the enrichment, rather than changing the burnup or cooling time. Thus it was opted to only use 33 GWd/MTHM at 2.6%, 45 GWd/MTHM at 3%, and 60 GWd/MTHM at 5.43%, calling these options ‘low’, ‘middle’, and ‘high’ respectively.

3.2.7.4 TIME SLICES

The MATLAB script takes as an input both the name of the output '.pcf' file to be created and the time slices for each KMP that are desired. These time-slices have been selected by hand i.e. going through the time-series in the MATLAB workspace and manually choosing a set of around 30 time-slices for each KMP. It is fine to do this manually, because it really only has to be done once and then one can run many analysis scenarios without having to choose them again. It is difficult to see which time-slices actually contain significant quantities of material for a given KMP (due to flows of material around the plant) without going in and choosing them by hand.

4. RESULTS

After several different iterations on the process, a fairly compact and reasonable automation implementation has been settled upon, despite having to leverage so many different technologies. This section describes that final product and how it is used. Additionally, the initial analysis of the gamma NDA prospects will be presented. This analysis shows quite promising results for NDA applications to pyroprocessing safeguards.

4.1 FINAL SOFTWARE SCHEME

Depicted below is the data and control flow through the various software components:

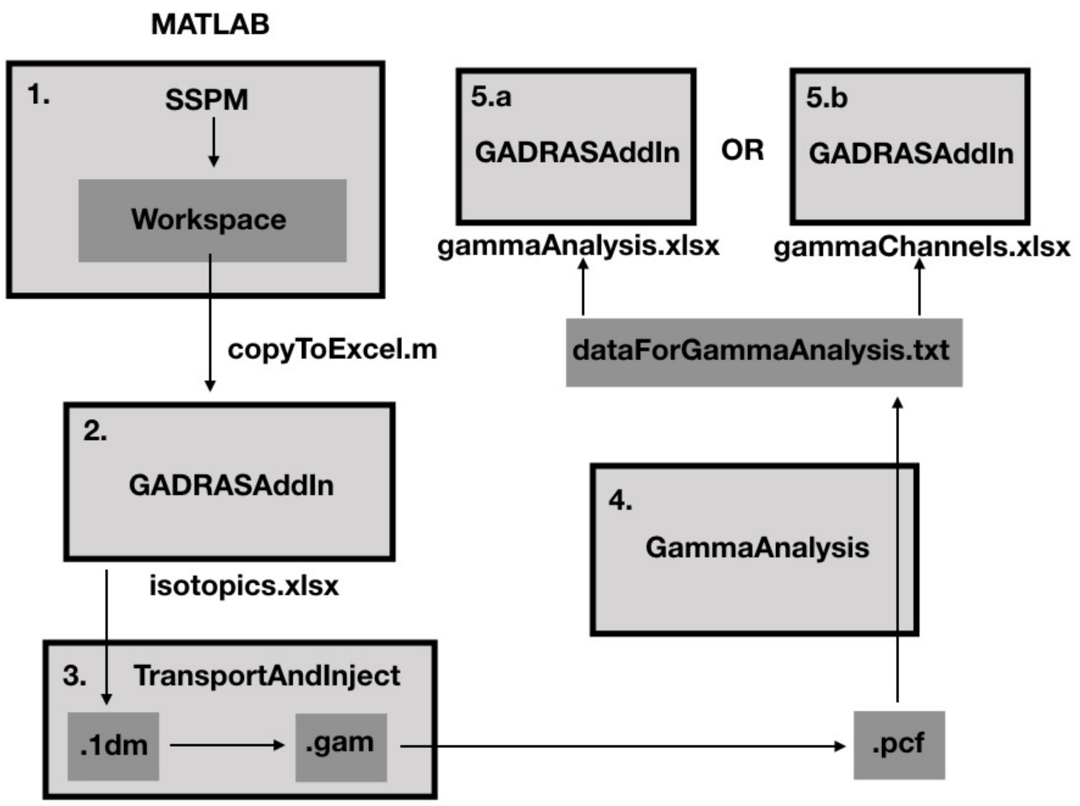


Fig. 4.1: Control/data flow through the various software components.

For additional information on software data/control flow, read section 4.2.2 below.

4.2 SOFTWARE USAGE GUIDE

4.2.1 INSTALLATION

(1) Install the latest version of Microsoft Visual Studio, including the VSTO framework and .NET 3.5, which requires an external download. Here is a link describing the process. https://docs.microsoft.com/en-us/dotnet/framework/install/dotnet-35-windows-10.

(2) Install GADRAS 18.8.x the installation drive should be C:\ by default.

(3) If the folder 'C:\GADRAS\Source\ApiTestSources\ApiTest_ParallelGADRAS' does not exist, then create it.

(4) Open the GammaNDA zip file and move the resulting folder to your desktop.

(5) Open the 'GADRASAddIn' Visual Studio project and click run to build and register this add-in with the local copy of Excel.

(6) Open the SSPM and add any needed 'ToWorkspace' blocks to the main signal flow in order to generate the variables that the 'copyToExcel.m' script will utilize. This is described in section 3.2.3. By default it will look for variables named 'UConf', 'UTRUConf', 'FPWasteInv', and 'MetalConf', for U, UTRU, fission product, and metal waste KMPs respectively.

(7) (Optional) Modify the diversion blocks of the SSPM to track how much plutonium is being diverted for a given diversion scenario. This process is described in detail in section 3.2.2.

4.2.2 USAGE

(1) Open the SSPM MPACT GUI (SSPMeChem_GUI_MPACTrev1.mlapp), choose your simulation parameters. My strategy for this step is described in sections 3.2.7.3 and 4.4.

(2) Run the SSPM to obtain diversion scenario data and write SSPM data to MATLAB workspace variables. Save these variables to a .mat file for later use if you would like. Make

sure that any .mat files you would like to load into MATLAB (for the next step) are located on the desktop so that the script can find them.

(3) In ‘copyToExcel.m’ choose the .mat file you would like to load (line 1), or comment out this line to use whatever variables are currently in the MATLAB workspace. Choose the name you want your pcf file to have (lines 7 & 8, these string will concatenate together).

(4) Run ‘copyToExcel.m’ in the MATLAB workspace. After the data has been copied to isotopics.xlsx, it should open automatically and trigger the Excel add-in.

(5) Once excel opens, a window should appear. Choose your geometry/transport/inject parameters here. Click “Run Transport and Inject” when you are ready to start the calculations. The dialogue window should disappear. Nothing visible should happen for a few moments while the .1dm files are being written, then a console window should open to track the progress of the transport and inject calculations.

(6) There is an unknown bug which in some cases causes the program to crash in between the transport and inject phases. You will know this has happened if the .pcf file does not appear in the detector folder and the ‘GammaAnalysis’ window does not appear prompting you to enter in your loss and control scenario IDs. If this happens, open the TransportAndInject Visual Studio project. In the ‘Program.cs’ file, change the ‘true’ boolean on line 18 to ‘false’. This will prevent the transport sequence from running and jump straight to the inject calculation. You will need to specify your own parameters for the inject calculation. Comment out the existing inject call and replace it with your own with parameters inserted by hand. Run the project in the Visual Studio editor. The inject calculation should now work correctly. Once it has finished running, return the boolean on line 18 to ‘true’, comment out your new inject function and uncomment the old ones. Always make sure to build the project ‘Toolbar->Build->Build Solution’ before your next analysis run to ensure normal operation.

(7) Once you have successfully completed the inject calculation. A new program ‘GammaAnalysis’ should open in a console window. This window may also be opened by running the ‘GammaAnalysis’ project from the Visual Studio editor. This command window

will prompt you to enter the names of your loss (diversion) scenario and control scenario .pcf files found in the relevant detector folder after their inject calculations have been run. If you do not yet have both a control and a diversion .pcf, just close the window and generate another one. If you have both and are ready to analyze, then enter in their filenames (just names, not full paths) without the '.pcf' at the end. The console will then ask if you would like to do total gamma or channel (spectrum) analysis. Enter 't' for total gamma and 'c' for channel analysis. If you choose the channel mode, you will also be asked to specify the KMP and time-slice for the spectrum comparison. Currently the options are 'U', 'UTRU', 'MW', and 'FP'.

(8) This will cause either 'totalGamma.xlsx' or 'gammaChannels.xlsx' to be automatically opened, depending on the analysis mode you chose. The data will be arranged and a chart will automatically be generated. Once the chart is generated, it is advisable to set cell A1 to zero and save as new file for future reference. This will prevent the analysis add-in from being run every time you open the file. Make sure the original (i.e. 'totalGamma.xlsx' or 'gammaChannels.xlsx') still has "totalGamma" or "gammaChannels" in cell A1 so it can still run.

Additional notes and warnings:

It is currently inadvisable to choose time-slices for 'copyToExcel.m' which are below 1000, as the system will process them *after* every other record, so they will appear at the bottom of the final Excel file. If it is really necessary to include these times, they may be moved by hand in the final analysis, but this extra step may be avoided by simply excluding them.

4.3 DATA PROCESSING TIMES

4.3.1 SSPM

On the machine used for this project, which has a 3.4 GHz processor, it takes approximately 6.5 hours to run the SSPM. This would not benefit from additional cores or a Graphical Processing

Unit (GPU), as the SSPM is coded as a single-threaded calculation and was not (probably cannot be) coded to take advantage of Simulink’s GPU acceleration feature.

4.3.2 PCF ANALYSIS

On the machine used for this project, which has a 3.4 GHz processor, it takes approximately 20 minutes to process around 130 individual records (for a particular measurement point and time). Given quite large variances in computer performance due to particularities among data sets, this comes out to a data processing time on the order of 10 seconds per record for the whole processing pipeline. This pipeline can benefit from parallel cores, as the GADRAS transport and inject API supports up to eight cores for each records processing operation.

4.4 PRELIMINARY SURVEY OF KEY MEASUREMENT POINTS

A qualitative investigation of diversion behavior at the various key measurement points (KMPs) enables a better understanding of which measurement points are most likely to be exploited by an attacker, and thus which ones we should focus on for this analysis. Trials of all KMPs were performed over the course of a year (using an undisclosed but substantial loss fraction, enrichment, burnup, and cool down time) to determine the upper limits of how much plutonium would be diverted by such an extreme diversion campaign. Sampling of the diverted plutonium content was achieved via the method outlined in section 3.2.2.

This survey identified the following diversion locations as significant (results can be found in appendix scope images (B.5-B.12): SNF Storage, Shredded Fuel, ER, UTRU, U, Oxidant Production, UTRU Drawdown. The following diversion locations were deemed insignificant: Metal Waste, RE Drawdown, FP Waste, Blender/Ceramic Waste.

Multiple attempts were made to utilize the multiple-regression analysis capabilities of GADRAS on the product ingot spectra, yet all these attempts indicated a marked difficulty in detecting individual concentrations of special nuclear material isotopes. E.g. one of these measurements was taken with a variable width iron shield in a spherical geometry. Below around

15 cm, the detector receives too many counts and fails to function correctly. At around 20 we have a chi-squared value of 260, which decreases to around 160 at 40 cm of shielding before beginning to increase again. Obviously these values mean that there is no meaningful correlation between the special nuclear materials isotopes to be identified and the spectral data generated by our source terms. This is an indication of the effective impossibility of using NDA for a direct measurement of the SNM isotopes, and an indication that this is not an avenue which should be considered.

Several calculations were carried out on the results of spectra with different shielding constraints, and show that the lead, copper, iron, and aluminum all behave in essentially the same way with regards to the resulting spectrum that we observe. Since lead is a common shielding material in the industry, due to it's compactness and availability, This is the material assumed to be used for shielding throughout the rest of this analysis.

4.5 EXPLORATIONS OF VARIOUS SIGNIFICANT QUANTITY DIVERSIONS

In light of the preliminary plutonium diversion survey conducted on the various possible diversion locations described above, the most probable targets for facility misuse by a malicious actor have been identified. The software developed in this thesis will be used to investigate these cases more deeply, and a preliminary trial of these capabilities is documented in the proceeding sections.

4.5.1 QUALITATIVE FINDINGS

The four diversion locations chosen for analysis are at (1) the electrorefiner (2) UTRU ingot processing (3) UTRU drawdown, and (4) the oxidant production unit. These four were chosen from the survey images found in appendix images B.5-B.12. The first two are the most important locations as they require far smaller diversion percentages to obtain one significant quantity of plutonium over the course of a year. As discussed in section 3.2.7.1, it was opted to explore protracted scenarios over abrupt scenarios, as the former have much stricter detection limits, while the latter are fairly easy to detect. The relevant charts are presented in appendix images A.1 and A.2 Looking at these images, a few things may be noted. First, for the electrorefiner diversion, note that the most important KMPs for detecting the facility misuse are the metal waste KMP and, perhaps surprisingly, the U product KMP. Also surprising is how the fission product waste KMP does not seem very useful. The UTRU product displays an oscillatory behavior for this diversion. For the UTRU diversion, we note that only the UTRU KMP is relevant. This makes sense, as the only KMP downstream of a UTRU diversion is the UTRU KMP itself. Among all the scenarios explored, this UTRU diversion case is perhaps the most difficult detect, and would warrant the most extra support from other safeguarding methods when designing a pyroprocessing facility. The UTRU drawdown and oxidant production diversion scenarios are presented in appendix charts A.4 and A.5. By inspection we can see that the UTRU drawdown discrepancy is much larger than the oxidant production discrepancy. This coupled with the fact that these diversions require a much higher percentage of the throughput to be diverted (5.8% and 10% respectively), place them at a lower threat level than ER or UTRU diversions, yet perhaps still worth the implementation of NDA monitors.

4.5.2 QUANTITATIVE FINDINGS

For the protracted electrorefiner diversion, there are a few noteworthy points. The first is that metal waste form and uranium product processing KMPs appear to be the most useful for detecting a protracted diversion, as their values steadily diverge from the control scenario over the course of the diversion. By the end of the diversion period the metal waste showed a discrepancy of 2100 CPS and the uranium processing a discrepancy of 2800 CPS. The latter is over 10 standard deviations with detectors operating at around 33,000 CPS. The standard deviation (for testing the null hypothesis) is found in [15] as the square root of the total gamma counts (182 CPS). If proportionality between diversion amount and count discrepancy is assumed, then it can be calculated that we have a 99% confidence in detecting a 0.2 SQ diversion via gamma NDA.

The fission product KMP did not prove to be very useful for detecting this ER diversion, and the transuranic KMP displayed a strange oscillatory behavior, which could still be useful in detecting anomalous facility use, but has not yet been statistically analyzed.

For the transuranic product diversion, it can be seen that the only relevant KMP is indeed the transuranic NDA measurement. This makes sense as this diversion scenario is very far “downstream” and should not affect all of the KMPs in the way that the electrorefiner diversion does. For the duration of this diversion, a consistent discrepancy between the control and diversion scenarios with an average CPS difference of approximately 400 is seen. The detectors are running a total CPS of around 24,000. Thus the standard deviation (for testing the null hypothesis) is 155 CPS. For just one measurement point we obtain a confidence interval of around 93% that the diversion is occurring [15]. Over the course of this diversion we may have as many as 60 measurement points. Research into the statistical implications of having so many measurement points is ongoing. To obtain a confidence interval of 99% our standard deviation could be multiplied by 2.58 [15]. 2.58*219 = 565. This discrepancy would correspond to a diversion of 1.4 SQ if proportionality between diversion amount and count rate is assumed.

5. DISCUSSION

Explored here are the implications and horizons of this research, as well as other details which are relevant to understanding the totality of the landscape pertaining to these approaches.

5.1 SIGNIFICANCE OF RESULTS

The results obtained, and that this software system will further elucidate in the future, seem very promising with regards to pyroprocessing safeguards applications. The probabilities for detecting an abrupt loss scenario are very good and well within the desired limits put forth by the International Atomic Energy Agency (IAEA), including the timeliness criteria (further discussed in section 5.6). However, this is also the scenario which is most easily detected by other safeguards mechanisms, and as a result, the one which a malicious agent would be least likely to pursue. The protracted scenario is much more interesting. Based on preliminary findings, detection limits which are within IAEA expectations can be obtained for all protracted diversion scenarios modeled by the SSPM with the exception of a UTRU ingot diversion, which is still close to those standards (99% confidence for a 1.4 SQ/a diversion). It is possible that even better results are possible with the spectrum peak analysis capabilities. These results can be obtained by the use of high-purity germanium detectors stationed at only three KMPs throughout the site (although the fission product waste KMP is featured prominently throughout the study, it is probably not necessary to obtaining these results). Thus, in conjunction with other pyroprocessing safeguards methodologies being developed, it seems there is no reason that extremely robust safeguarding systems cannot be brought online to enable the next generation of nuclear reprocessing technologies to safely flourish.

5.2 TRANSFER TO SANDIA NATIONAL LABORATORY

Now that this set of analysis tools is in its rudimentary stage of completion, the process of transferring the code over to Sandia National Laboratory personnel has begun. Although the code does have many external dependencies and requires a fairly convoluted software environment, work is being done to investigate how this burden might be lessened. A fairly obvious possibility is compiling the software into binary, as it is currently operated in Visual Studio project form to provide maximum flexibility. When the feature set is more hammered down and users are not desiring to add more functionality on the fly, it would be quite advantageous to compile binaries to be used on new machines, along with the required dynamic link libraries, so that each end user is not required to set up or manage a Visual Studio project, which could lead to software incompatibilities. Another important consideration is the version of GADRAS used. This software is written for GADRAS 18.8.11, and using a newer version of GADRAS may introduce bugs.

5.3 POTENTIAL FURTHER DEVELOPMENTS

There are multiple avenues for further research along these lines, both on an analytical and an instrumentation level. An important first expansion would be the inclusion of neutron count analysis alongside the gamma counting. This would be fairly straightforward as GADRAS allows for a neutron detector to be "attached" to any gamma detector being used, and the neutron calculations are performed at the same time as the gamma calculations. The neutron transport results are stored in the same .gam files used by the gamma detectors, and the addition of total neutron counts to the 'GammaAnalysis' software would not be very difficult. Of course, it would be important to keep in mind the HDND project also being developed by MPACT (more details in section 2.3), and whether or not developing this neutron capability would be redundant in the face of that work.

Another further development which may be beneficial is the inclusion of highly-enriched uranium (HEU) into the SNM for which we are attempting to detected significant diverted quantities. The work of this thesis has focused on plutonium, as it is more coveted as a bomb-making material than HEU due to substantially less being required for the fabrication of a weapon. However, this certainly does not rule out the possibility of a significant quantity of HEU being diverted from a pyroprocessing facility in the same manner, and it would be interesting to see if detection limits on such a diversion could possibly be lower than they are for the plutonium case. The main diversion point of interest here would naturally be the U product processing diversion location and KMP.

With regards to further developments of the usability and dependability of the software tools themselves, making the software easier to install on new machines would also provide some benefit. Some difficulties have already been encountered in transferring the codes to Sandia National Laboratory personnel, and those experiences are actively being used to inform an iterative cycle which is leading to better packaging and self-containment of the software tools.

5.4 TOTAL GAMMA CPS VS SPECTRUM PEAK ANALYSIS

The use of total gamma CPS NDA systems vs an energy-specific peak analysis NDA system presents an interesting balance of factors for facility designers. Although including peak ratio analysis does introduce a new dimension of information (energy) along which facility misuse could be detected, it also necessitates a lower gamma exposure than a detector which is being used for total count analysis. Pileup and other effects which could compromise the resolution of an energy spectrum are not important considerations for a system relying on total gamma count, meaning that the detector may be exposed to a greater number of gamma counts, which will lead to better counting statistics. It is also plausible that NDA systems relying on total gamma count may be less expensive, as resolution, which is often a quite important driving cost for detectors, is not an important factor. The difference in price between NaI and HPGe detectors is substantial, and being able to recommend the former over the latter for a comparable increase in safeguards efficacy, would make NDA detection a very attractive choice for a facility designer.

5.5 DEFEAT OF TOTAL GAMMA CPS VIA HOT SUBSTITUTION

One of the objections which may be raised to the use of total gamma count NDA safeguards is the possibility of a malevolent agent defeating the safeguard by means of a “hot substitution”. That is, because this method does not discriminate between gamma energy levels, the attacker may substitute in new radionuclides during the diversion procedure, in such a way that the total gamma count detected at the critical KMPs is not changed. While this is certainly a worthwhile objection, and should be considered in full, here are a few possible responses to this objection.

One possible response is that although the attacker may very well substitute in whatever radionuclides they like, the probability that just any radionuclide across the table of isotopes would actually reach the desired KMP as the original would is substantially smaller. Unless they are direct chemical analogues of the original material (which would fool a system sensitive to gamma energy levels anyways) the substituted material will very likely not have the same chemical properties as the original, and thus flow through the plant in a different way, setting off not just the NDA detection system, but likely many other safeguards systems as well.

Another response is that although this hot substitution is a viable threat, the necessity of executing this additional dangerous and complicated maneuver will help to decrease the chance of a diversion attack occurring. The general purpose of safeguards is to deter attackers, and whether or not an NDA system, even one vulnerable to hot substitution, represents a substantial enough deterrence to would-be attackers is an interesting question for someone designing safeguards systems in such a facility.

5.6 SIGNIFICANT QUANTITIES AND TIMELINESS

The exposition of this project has thus far taken for granted the various standards and definitions employed by the IAEA. The two most important definitions are those of *significant quantity*, already used extensively, and *timeliness* of detection, which has not yet been discussed.

A significant quantity (SQ) of material, officially defined as 8 kg of plutonium or 25 kg of U-235 specifically. The difference between these two numbers results from the amount of material which is lost in the conversion and manufacturing process during weapon fabrication for highly-enriched uranium vs plutonium. This is taken into account by the IAEA and reflected in the ultimate value [29].

Timeliness refers to the acceptable window within which a diversion must be detected, and is dependent upon additional steps required to fabricate a bomb given characteristics of diverted material. The standards are currently 1-month detection for unirradiated direct -use (no further processing required), 3-months for irradiated direct-use, and 12-months for indirect-use (further processing required). [28]

The gamma NDA system preforms excellently with regards to the timeliness requirement. This is because radiation detectors are a type of “process monitoring” which may be collected very frequently, close to real-time, reflected by the fact that measurements can be taken from the SSPM for each hour of operation. This is extremely good compared to some other IAEA evaluation techniques, such as requiring human inspectors to visit a facility and take inventory. Furthermore, the UTRU to be gained from a pyroprocessing facility, despite having high plutonium content, may be designated as indirect-use given high amounts of undesirable spontaneous neutron emitter Cm 244 [8]. Although this same argument does not hold for uranium, the fuel being processed at an electrochemical facility will almost certainly be LEU except in extraordinary situations. In light of these, I can safely say that the systems investigated in this project do extremely well with regards to IAEA timeliness standards.

As regards significant quantities, electrochemical plants are in theory more proliferation-resistant, because the IAEA significant quantity does not always take into account certain aspects of the material (such as UTRU containing high amounts of Cm 244) as described above. Instead opting to emphasize “the need for safeguards to protect against the diversion and misuse of separated plutonium applies essentially equally to all grades of plutonium. “ [8]. The presence of curium in this product will almost certainly necessitate further processing and material loss by any bad actors, therefore necessitating an increase in material diverted if a bomb is to be manufactured. This reality means that our results may be overestimating the threshold detection limits for SQ loss scenarios, i.e. the safeguards systems in reality may perform even better than expected based on this work.

5.7 VERIFICATION AND VALIDATION

Even though it has been shown that in a virtual facility, i.e. the SSPM, there can be detected a difference in gamma counts between control and diversion scenarios, it is a nontrivial matter to port that knowledge over to a real world pyroprocessing facility such as the ACPF. The problem lies in how well the SSPM can truly be used as a substitute for an actual facility. Due to random error and the impossibility of a completely faithful mathematical model of reality, the SSPM data will, to a greater or lesser extent, diverge from the actual numbers achieved in a real world facility.

This speaks to a greater issue of verification and validation faced by this project. Luckily the tail-end analyses in GADRAS are resting upon a widely-used and well tested software suite, but the SSPM itself is very much a new and untested model. This is due to its fairly recent development, combined with a lack of full-scale real world facilities to provide data against which it may be validated. The difficulty in initially applying safeguards whose development is dependent on empirical real-world plant results is that not such data exists. So we have a chicken-and-egg problem in which plants may not be constructed without safeguards, but safeguards may not be fully developed without plants.

There are a couple of possible redemptive stances. The first is that the models used to construct the SSPM (AMPYRE and DyER, see section 2.2) have been validated against scaled-down versions of these pyroprocessing systems. This gives them some, if not full credibility as the structural basis for the SSPM. The second point of solace is that given the sheer range of technologies developed by the MPACT group, the first electrochemical plant will likely be over-outfitted and used as a laboratory for such technologies. Under such intense scrutiny and having so many different (likely redundant) measurement instruments running, the likelihood of a malicious diversion drops significantly. This “laboratory-plant” will then provide the full-scale real-world data necessary to validate models and refine the safeguards approaches, so that they may be used more cost-effectively in any subsequently constructed pyroprocessing facilities.

6. CONCLUSIONS

The tools developed during this project will be useful for anybody who is researching, designing, or implementing safeguards for a pyroprocessing facility. Based on the cursory results obtained by use of these tools, the conclusion may be reached that simple gamma NDA, while perhaps not as new and exciting as other methods being pioneered for pyroprocessing safeguards applications, can be an excellent mechanism for the detection of facility misuses. Based on the preliminary findings, detection limits which are within IAEA expectations can be obtained for all loss scenarios modeled by the SSPM, especially protracted and abrupt scenarios at the electrorefiner, which has perhaps posed the greatest challenge to safeguarding [8]. The exception of a UTRU ingot diversion, which is still close to those standards (99% confidence for a 1.4 SQ/a diversion), is mitigated by the fact that electrochemical UTRU product is very radiologically hot, and therefore not extremely attractive to potential bad actors. It is possible that even better results are within reach through use of the spectrum peak analysis capabilities developed at the end of the thesis writing period, whose impacts have not yet been fully evaluated. Furthermore, these results can be obtained by the use of high-purity germanium detectors stationed at only three KMPs throughout the site (although the fission product waste KMP is evaluated and featured prominently throughout the study, it is probably not necessary to place a detector there). Thus, in conjunction with other pyroprocessing safeguards methodologies being developed, it appears there is no reason that extremely robust safeguarding systems cannot be brought online to enable the next generation of nuclear reprocessing technologies to safely flourish.

Hopefully these tools will be utilized by such professionals as can benefit from them, perhaps as a part of MPACT's Virtual Facility Test Bed or otherwise, and they can be incorporated into analysis workflows for members of the international nuclear community who are interested in pyroprocessing as an avenue for improving their nuclear energy production.

This software is, at the time of writing, in the process of being transferred over to personnel at Sandia National Laboratory, so that they may safely keep it and use it as they see fit. Although the author will no longer be working on the project in any official capacity, he urges anyone reading this who is interested in using the software, or is perhaps having issues with the software, to get in contact so that an attempt may be made to correct the issues. If there is a problem which cannot be troubleshot by the contents of this thesis, particularly section 4.2, please contact for additional support at noah.christopher.harris@gmail.com.

BIBLIOGRAPHY

[1] Cho, I. J., D. H. Kook, K. C. Kwon, E. P. Lee, W. M. Choung, and G. S. You. "Design and Verification of Shielding for the Advanced Spent Fuel Conditioning Process Facility." Health Physics 94, no. 5 Suppl 2 (2008): S65-71.

[2] Cipiti, B. (2020, June). *Milestone 2020 - Overview*. Presented at the MPACT Spring 2020 Virtual Meeting.

[3] Cipiti, B., Browne M., Reim, M. 2021. “The MPACT 2020 Milestone: Safeguards and Security by Design of Future Nuclear Fuel Cycle Facilities.” Journal of Nuclear Materials Management xx-xx

[4] Cipiti, Benjamin B., and Mancel Jordan Parks. “Integration of Materials Accountancy and Process Monitoring Data with Physical Protection.” No. SAND2016-10373C. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[5] Croce, M. et al. 2021. Electrochemical Safeguards Measurement Technology Development at LANL, *J. Nucl. Mater. Manage,* Vol. X, No. X.

[6] Frigo, Arthur A., Dale R. Wahlquist, and James L. Willit. “A conceptual advanced pyroprocess recycle facility.” No. ANL/CMT/CP-111325. Argonne National Lab.(ANL), Argonne, IL (United States), 2003.

[7] Fugate, M. L., Key, B.P., and Tutt, J.R. 2019. Monitoring an Electro-Refining Process: Revision 1, LA-UR-19-29553, Los Alamos National Laboratory.

[8] Haori, Y., Zhang, J., & Cipiti, B. (2018). Integration of Nuclear Material Accounting Data and Process Monitoring Data for Improvement on Detection Probability in Safeguarding

[9] Hoyt, N.C., Launiere, C.A., and Stricker, E.A. 2021. In-Process Monitoring of Molten Salt Composition by Voltammetry and Automated Sampling-based Techniques, *J. Nucl. Mater. Manage,* Vol. X, No. X.

BIBLIOGRAPHY (Continued)

[10] Harris, N.C. et al. 2021. University Research to Support the MPACT 2020 Milestone, *J. Nucl. Mater. Manage,* Vol. X, No. X.

[11] Higham, Desmond J., and Nicholas J. Higham. MATLAB guide. Society for Industrial and Applied Mathematics, 2016.

[12] Horne, Steven M., Gregory G. Thoreson, Lisa A. Theisen, Dean J. Mitchell, Lee Harding, and Wendy A. Amai. GADRAS-DRF 18.5 User's Manual. No. SAND2015-5176. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.

[13] Humberto Garcia, Wen-Chiao Lin, Reed Carlson, and Idaho National Laboratory. "Evaluating Safeguards Benefits of Process Monitoring as Compared with Nuclear Material Accountancy." 2014, 01 July 2014.

[14] Kim, Ho-Dong, T.H Lee, J.S Yoon, S.W Park, S.Y Li, T.K Menlove, H. Miller, M.C Tolba, A. Zarucki, R. Shawky, S. Kamya, and 555 North Kensington Avenue, La Grange Park, Il 60526 American Nuclear Society. "Safeguards System for the Advanced Spent Fuel Conditioning Process Facility." 2007, 01 July 2007.

[15] Knoll, Glenn F. Radiation detection and measurement. John Wiley & Sons, 2010.

[16] Lafreniere, P., Fugate, M., and Key, B. 2021. Advanced Integration of Safeguards Measurements, *J. Nucl. Mater. Manage,* Vol. X, No. X.

[17] Launiere, C. et al. 2019. Benchtop Assembly and Testing of Flow Cell Pneumatic Droplet Generator and Sampling Line, M3NT-19AN040104011, Argonne National Laboratory.

[18] Lee, Sung-Ho, Dong-Sup So, Byung-Doo Lee, Hyun-Jo Kim, and Ho-Jun Park. "Safeguards Approach to ACPF (Facility Level)." In Proceedings of the Korean Nuclear Society Conference, pp. 1183-1184. Korean Nuclear Society, 2005.

[19] Lim, Eunjung. "South Korea's Nuclear Dilemmas." Journal for Peace and Nuclear Disarmament 2, no. 1 (2019): 297-318.

BIBLIOGRAPHY (Continued)

[20] Maggos, L.E., Williamson, M.A., and Pereira, C. 2021. Flowsheet and Facility Design to Support Safeguards and Security by Design (SSBD) for Future Nuclear Fuel Cycle Facilities, *J. Nucl. Mater. Manage,* Vol. X, No. X.

[21] Miller, Michael. “Material Protection, Accounting and Control Technologies (MPACT) Campaign Overview and Advanced Instrumentation Development.” Lecture, Instrumentation and Control Review Meeting, September 17, 2014.

[22] Philip Baxter, “Approaches to Nuclear Cooperation: A Review of the U.S.-ROK Agreement,” Science & Diplomacy, Vol. 4, No. 3 (September 2015*).

[23] Seo, Hee, Seong-Kyu Ahn, Chaehun Lee, Jong-Myeong Oh, and Seonkwang Yoon. "ASNC Upgrade for Nuclear Material Accountancy of ACPF." Nuclear Inst. and Methods in Physics Research, A 880 (2018): 58-66.

[24] “Simulink.” Simulink Documentation. Accessed September 25, 2020. https://www.mathworks.com/help/simulink/.

[25] Williamson, M. A., and J. L. Willit. "Pyroprocessing flowsheets for recycling used nuclear fuel." Nuclear Engineering and Technology 43, no. 4 (2011): 329-334.

[26] You, Gil-Sung, Choung, Won-Myung, Ku, Jeong-Hoe, Cho, Il-Je, Kook, Dong-Hak, Kwon, Kie-Chan, Lee, Eun-Pyo, and Lee, Won-Kyung. "DESIGN AND CONSTRUCTION OF AN ADVANCED SPENT FUEL CONDITIONING PROCESS FACILITY (ACPF)." Nuclear Engineering and Technology 41, no. 6 (2009): 859-66.

[27] Zhang, Jinsuo. "Electrochemistry of actinides and fission products in molten salts—Data review." Journal of Nuclear Materials 447, no. 1-3 (2014): 271-284.

[28] Fortakov, V. Nuclear verification: What it is, how it works, the assurances it can provide. No. INIS-XA--183. 1998.

[29] International Atomic Energy Agency. IAEA Safeguards Glossary-2001 Edition. International Atomic Energy Agency, 2002.

APPENDICES

These appendices contain loss scenario comparison charts generated during my analysis and other supporting data/images, as well as the MATLAB and C# codes written for the project.

APPENDIX A: GAMMA CHARTS

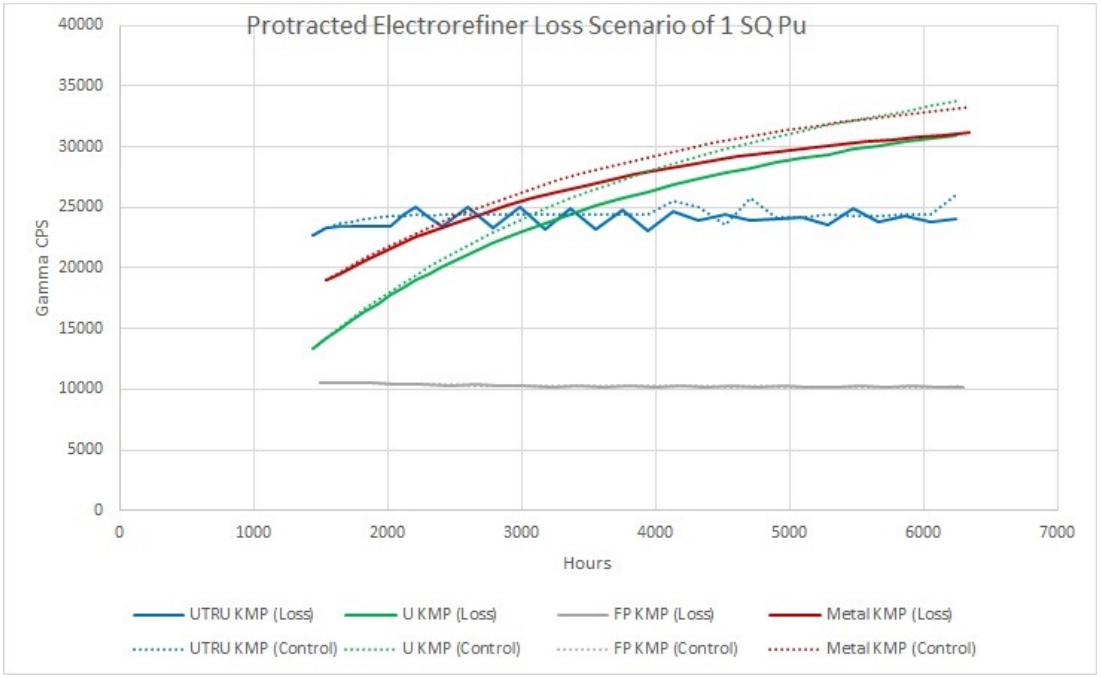


Figure A.1: Protracted electrorefiner loss of 1 SQ.

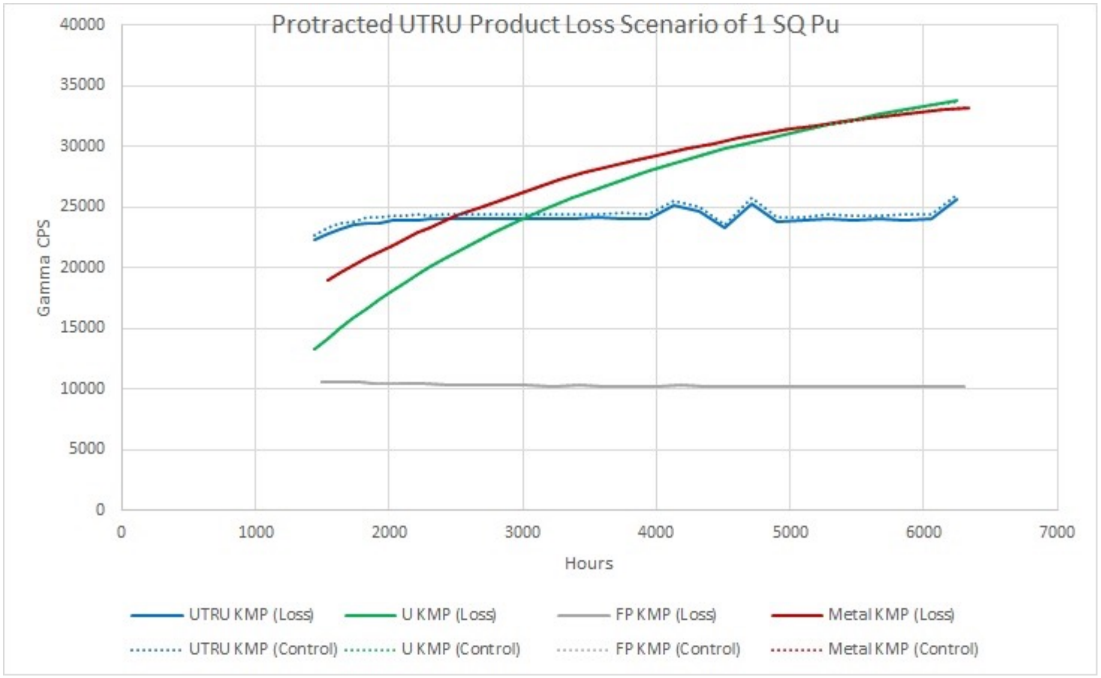


Figure A.2: Protracted UTRU product loss of 1 SQ.

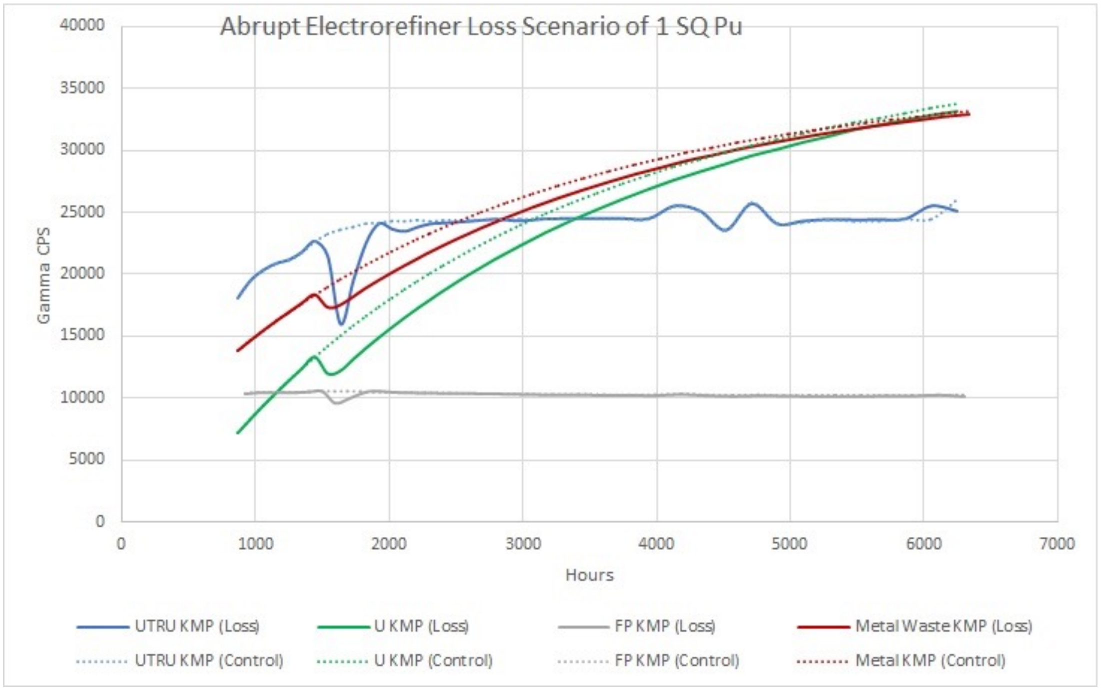


Figure A.3: Abrupt electrorefiner loss of 1 SQ.

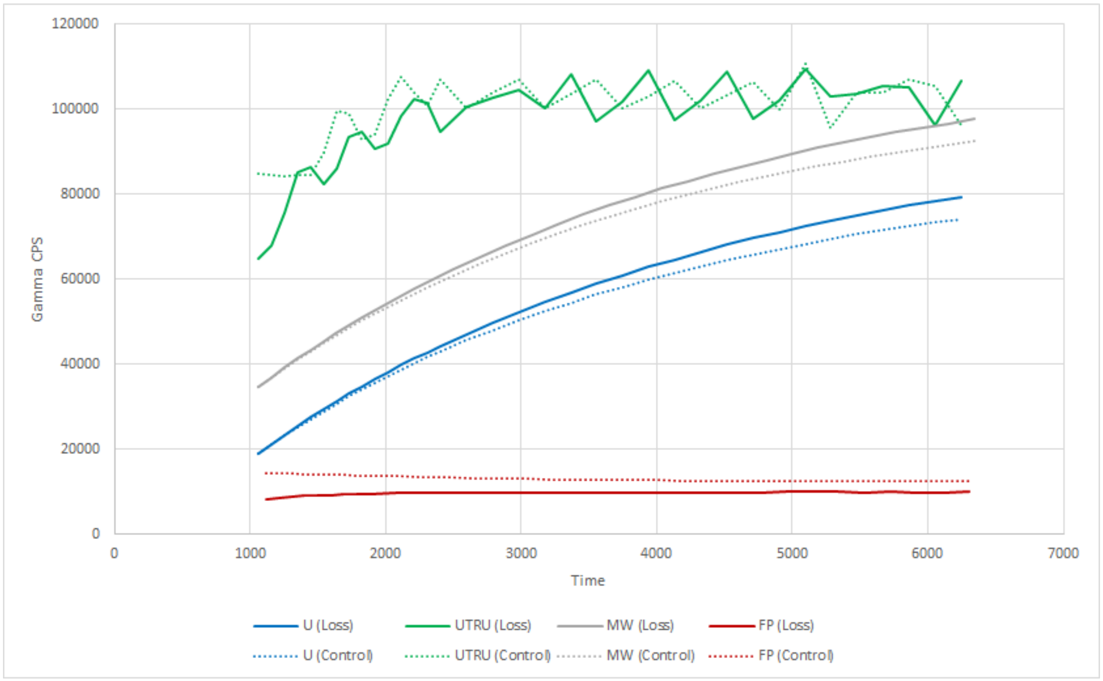


Figure A.4: Protracted UTRU drawdown loss of 1 SQ.

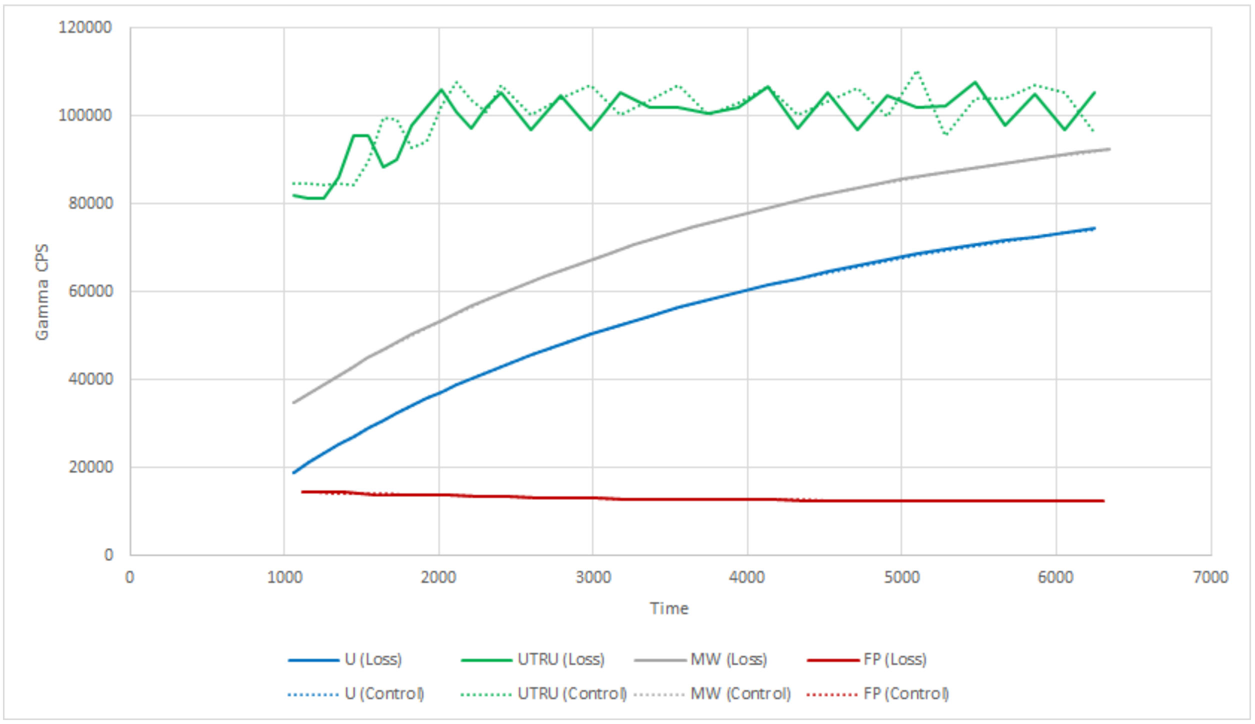
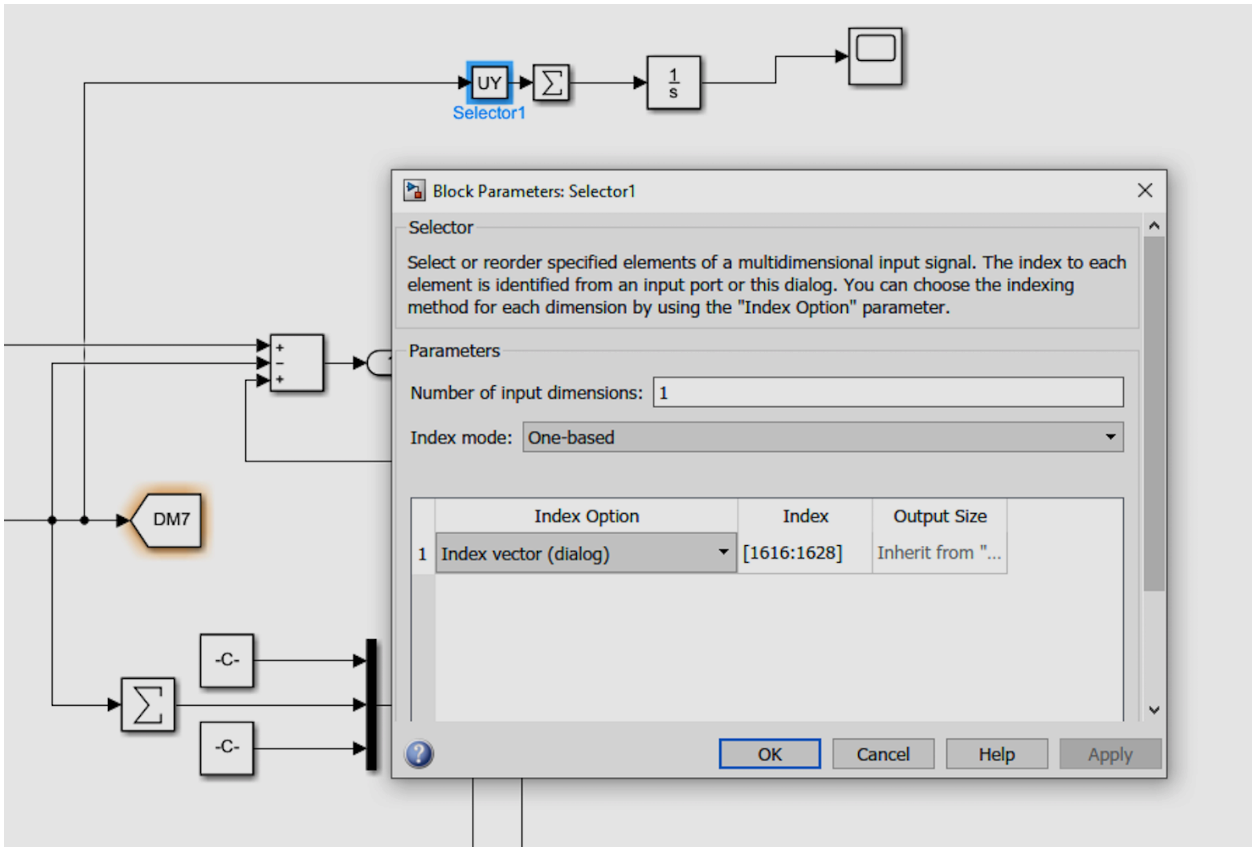


Figure A.5: Protracted oxidant production loss of 1 SQ.

APPENDIX B: SUPPORTING DATA & FIGURES



Block Parameters: Selector1

Selector

Select or reorder specified elements of a multidimensional input signal. The index to each element is identified from an input port or this dialog. You can choose the indexing method for each dimension by using the "Index Option" parameter.

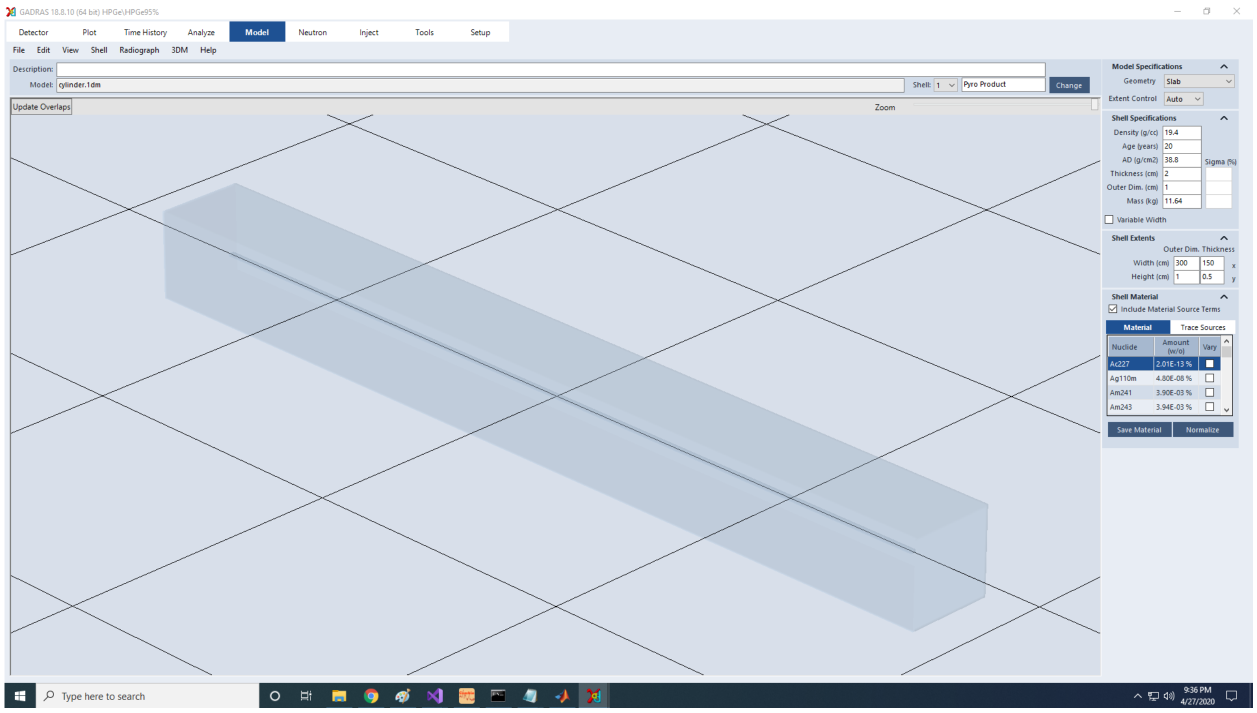
Parameters

Number of input dimensions: 1

Index mode: One-based

	Index Option	Index	Output Size
1	Index vector (dialog)	[1616:1628]	Inherit from "...

Fig. B.1: Simulink setup for UTRU diversion detection, along with the preferences for the selector block.



GADRAS 18.8.10 (64 bit) HPGe\HPGe95%

Description:
Model: cylinder.1dm
Shell: 1
Pyro Product

Model Specifications
Geometry: Slab
Extent Control: Auto

Shell Specifications
Density (g/cc): 19.4
Age (years): 20
AD (g/cm2): 38.8
Thickness (cm): 2
Outer Dim. (cm): 1
Mass (kg): 11.64
Sigma (%)
☐ Variable Width

Shell Extents

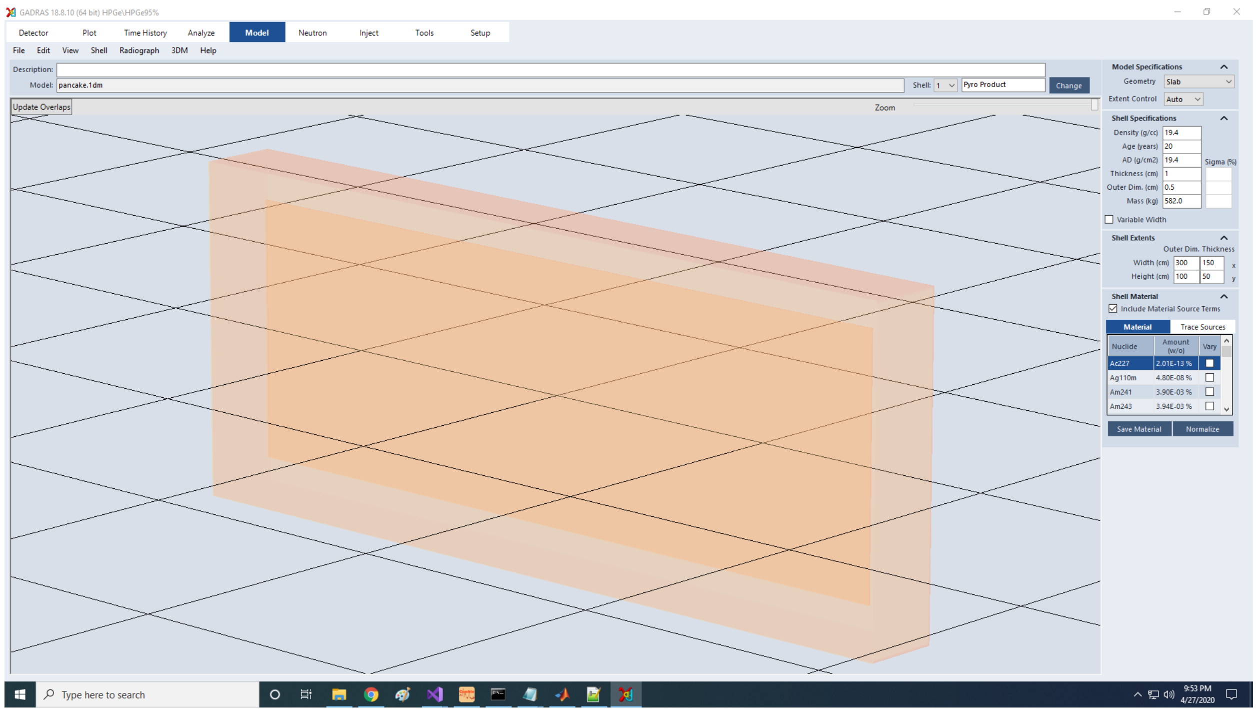
	Outer Dim.	Thickness	
Width (cm)	300	150	x
Height (cm)	1	0.5	y

Shell Material
☑ Include Material Source Terms
Material | Trace Sources

Nuclide	Amount (w/o)	Vary
Ac227	2.01E-13 %	☐
Ag110m	4.80E-08 %	☐
Am241	3.90E-03 %	☐
Am243	3.94E-03 %	☐

9:36 PM
4/27/2020

Fig. B.2: Cylindrical GADRAS geometry.



GADRAS 18.8.10 (64 bit) HPGe\HPGe95%

Description:
Model: pancake.1dm
Shell: 1
Pyro Product

Model Specifications
Geometry: Slab
Extent Control: Auto

Shell Specifications
Density (g/cc): 19.4
Age (years): 20
AD (g/cm2): 19.4
Thickness (cm): 1
Outer Dim. (cm): 0.5
Mass (kg): 582.0
Sigma (%)
☐ Variable Width

Shell Extents

	Outer Dim.	Thickness	
Width (cm)	300	150	x
Height (cm)	100	50	y

Shell Material
☑ Include Material Source Terms
Material | Trace Sources

Nuclide	Amount (w/o)	Vary
Ac227	2.01E-13 %	☐
Ag110m	4.80E-08 %	☐
Am241	3.90E-03 %	☐
Am243	3.94E-03 %	☐

9:53 PM
4/27/2020

Fig. B.3: Slab GADRAS geometry.

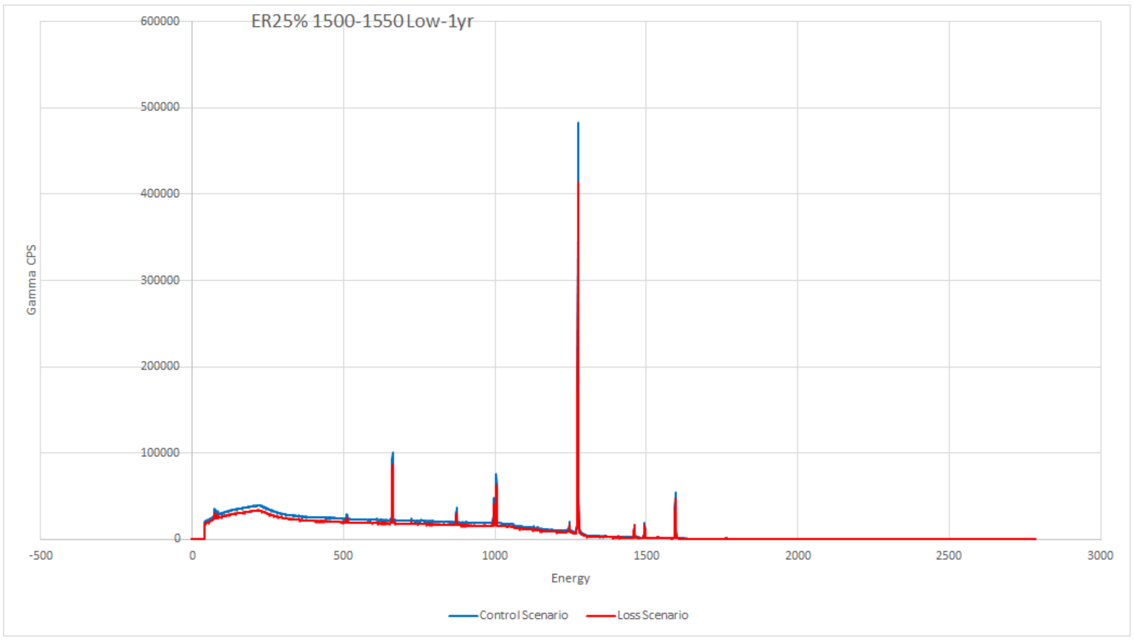


Fig. B.4: Example output from peak analysis mode. You can see the red data series overlaid on top of the red one, representing the difference in the gamma peaks between the diversion and control scenarios.

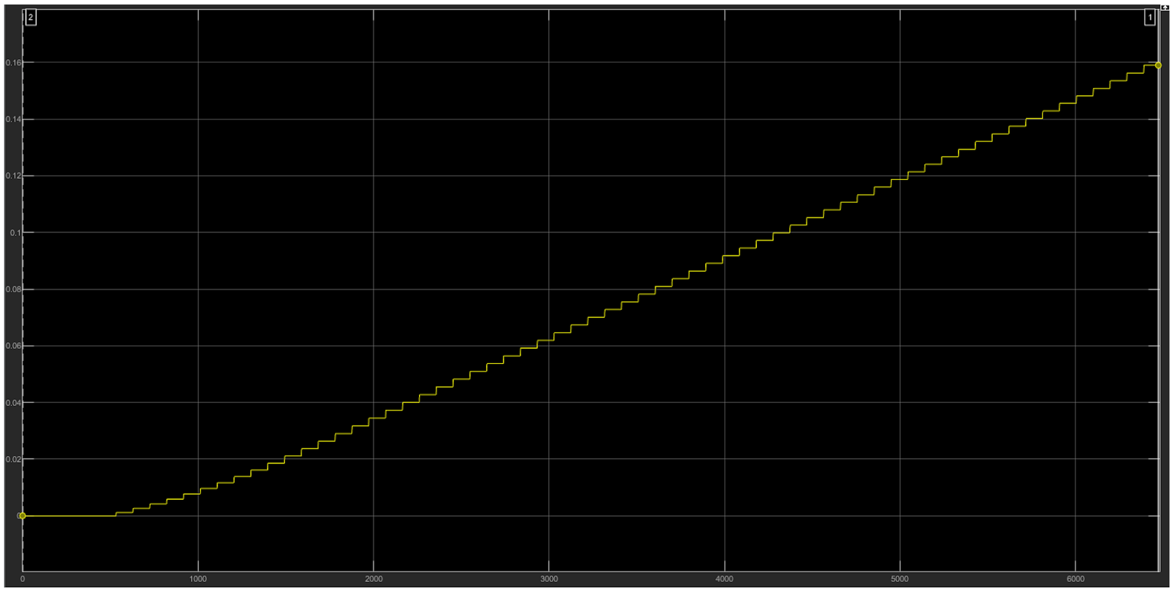


Fig. B.5 Protracted plutonium loss at spent fuel storage.

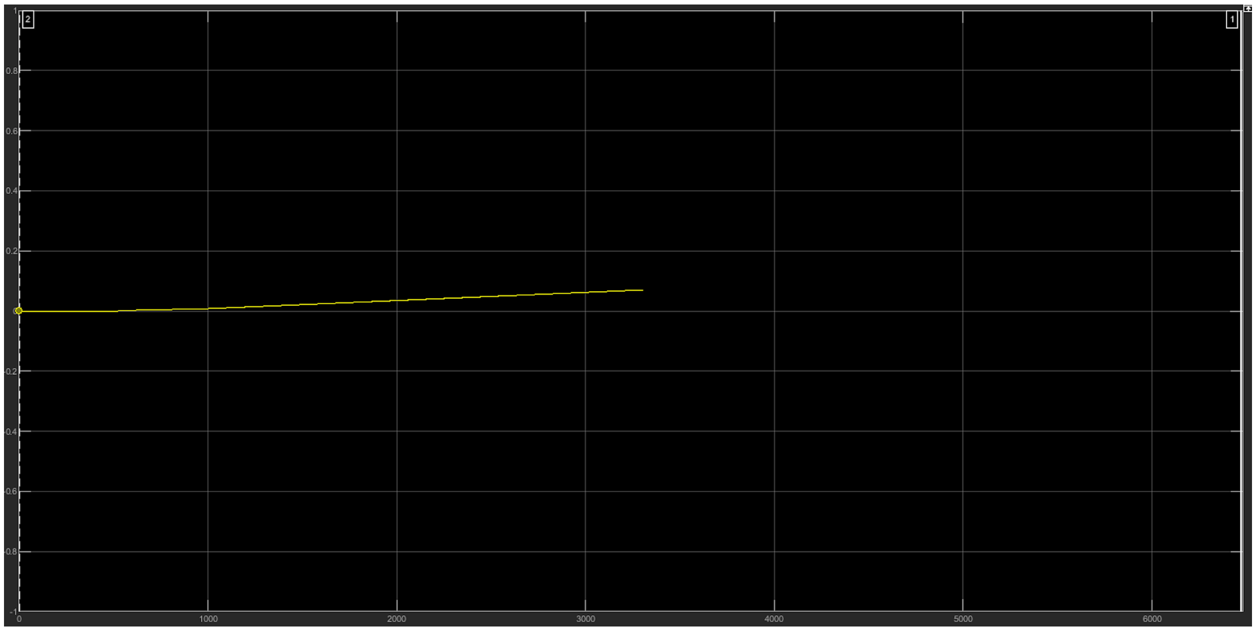


Fig. B.6 Protracted plutonium loss at fission product waste.

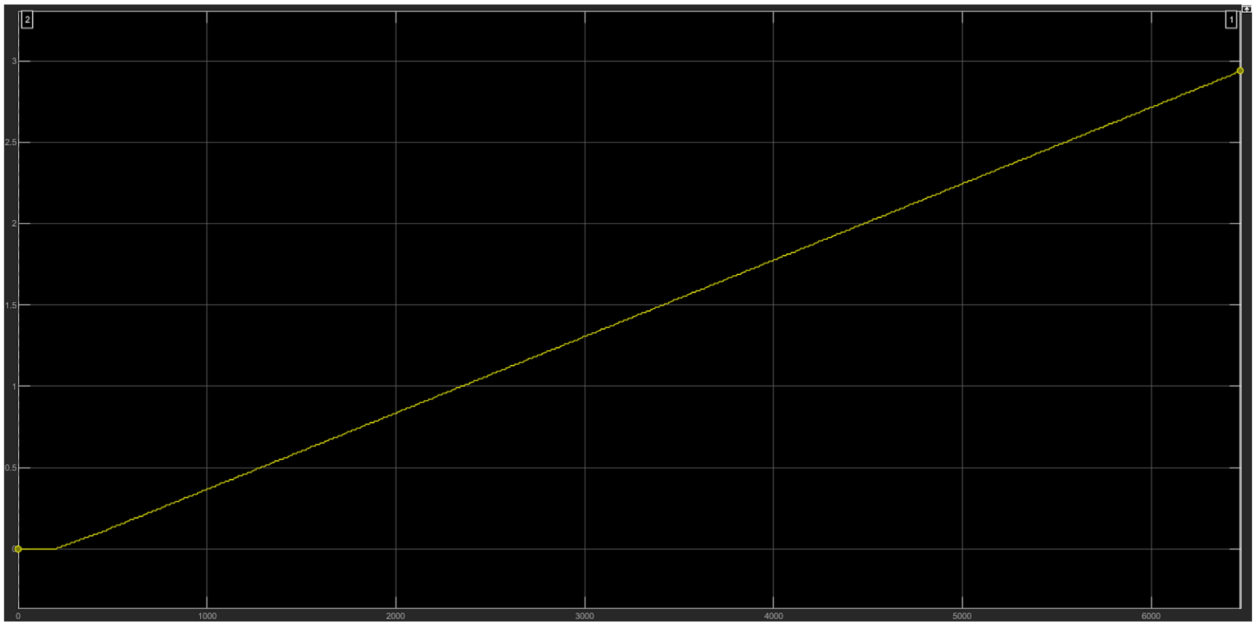


Fig. B.7 Protracted plutonium loss at metal waste.

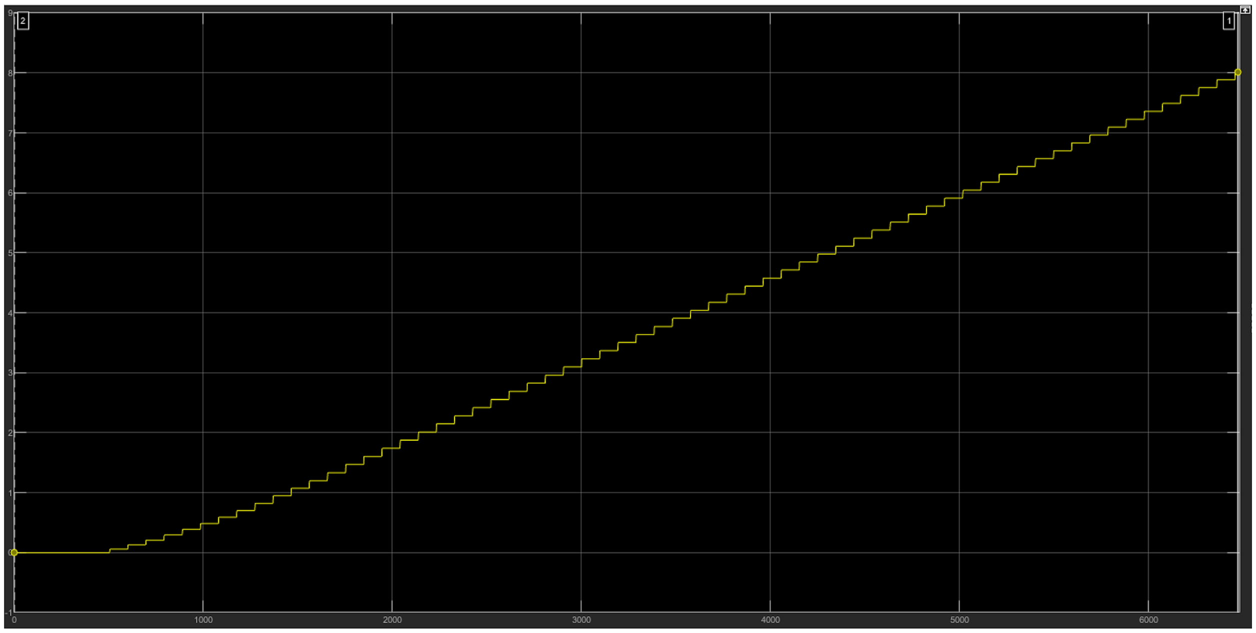


Fig. B.8 Protracted plutonium loss at oxidant production.

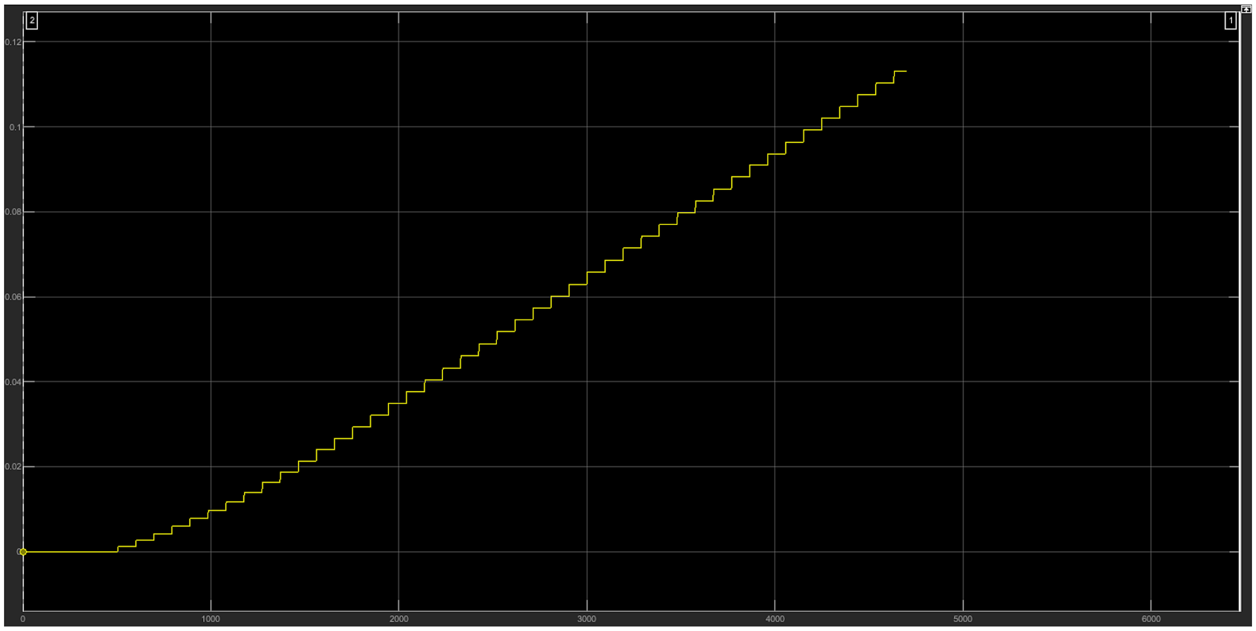


Fig. B.9 Protracted plutonium loss at rare earth drawdown.

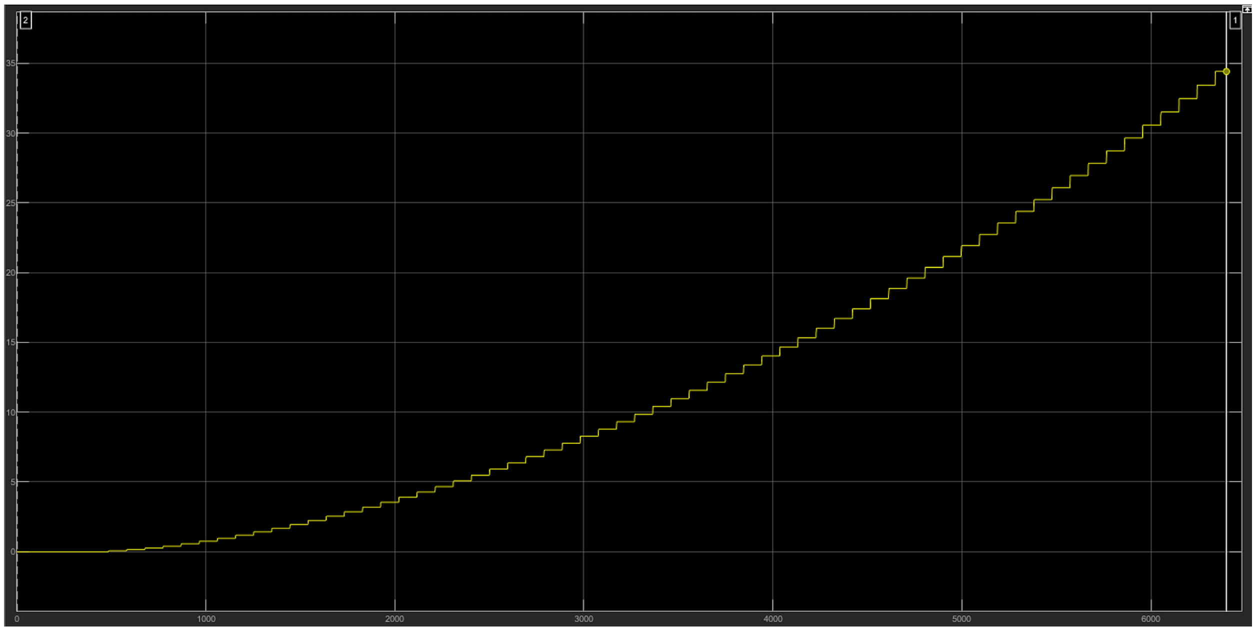


Fig. B.10 Protracted plutonium loss at UTRU drawdown.

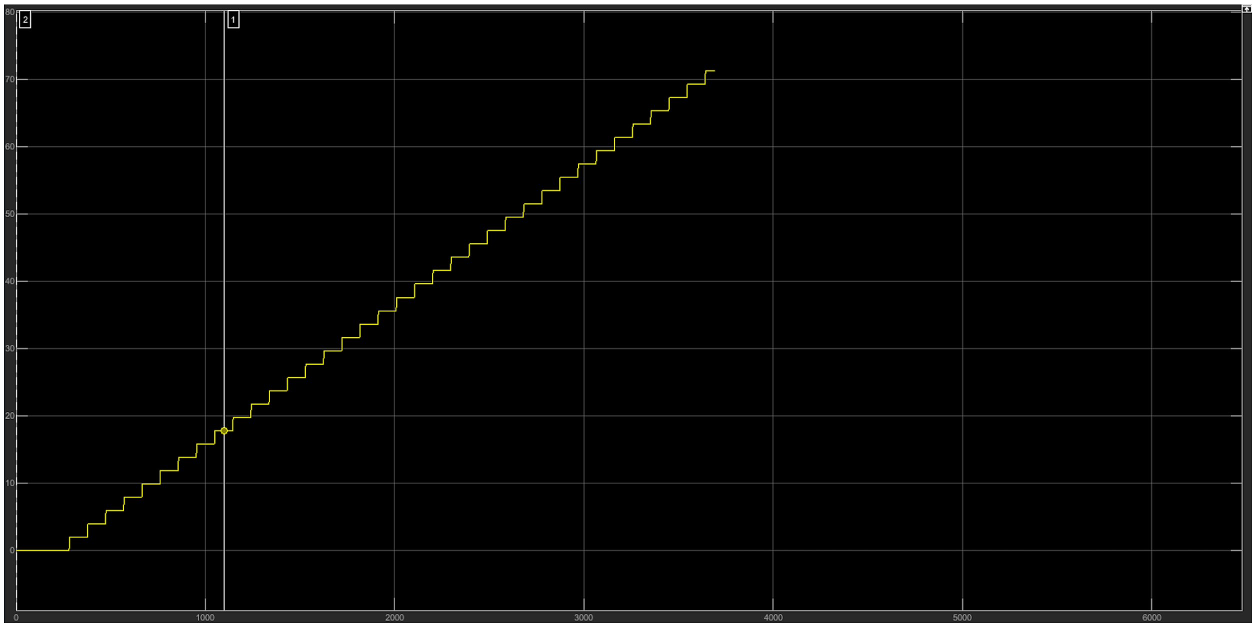


Fig. B.11 Protracted plutonium loss at shredded fuel inventory.

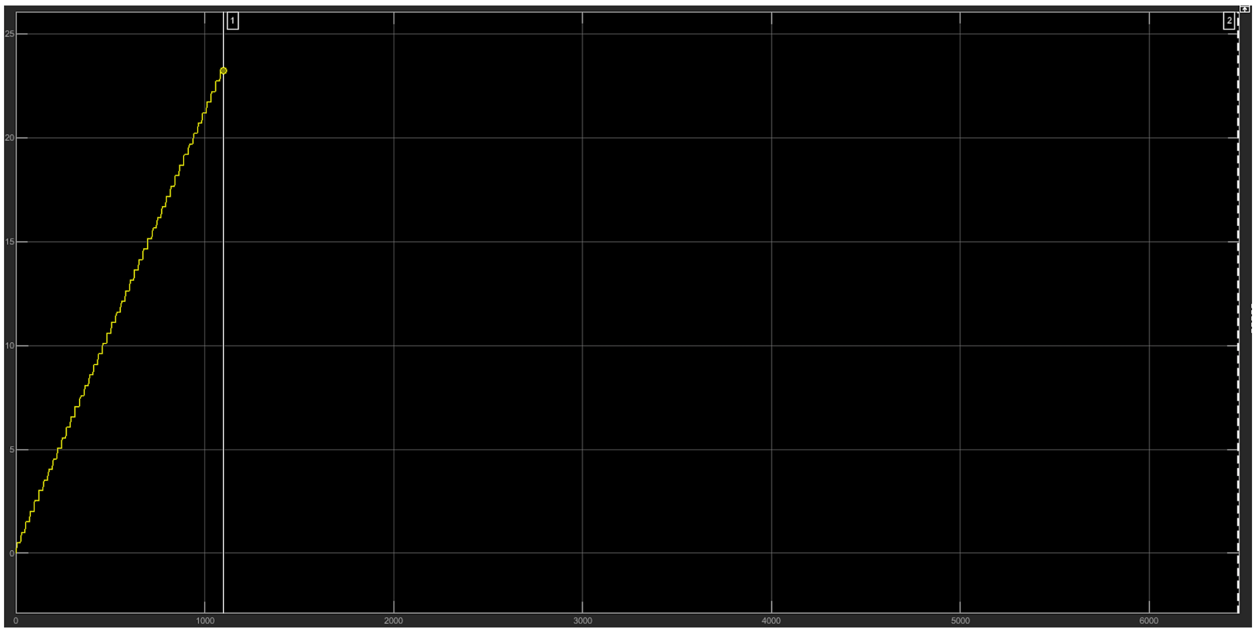


Fig. B.12 Protracted plutonium loss at spent fuel storage.

APPENDIX C: CODE

########################### copyToExcel.m ###########################

```
load('215.mat');
%load('run2.mat');
%variables
filename = 'isotopics.xlsx';


%diversionString = 'ER25';
%rangeString = 'r1500-1550';

diversionString = 'control-high-5yr';
rangeString = '';


%sampling density = 10; %sampling density *per KMP

%C = {'hello' 'yes' 'no' 'goodbye'}; %sample of cell array data
%C = {'b7684'};
%writecell(C,filename,'Sheet',1,'Range','A7');
%writematrix(UConf(7684,:),filename,'Sheet',1,'Range','B7');


%need to come up with the best way of identifying these intervals,
%perhaps there is a "leading isotope", which is present in some
%amount at every interval, such as Caesium

%clear previous isotopic data before writing new isotopic data
X = strings(200,1677);
writematrix(X,filename,'Sheet',1,'Range',sprintf('A5'));

%UConf
numlist = {1060, 1156, 1252, 1348, 1444, 1540, 1636, 1732, 1828, 1924, 2020, 2115, 2212,
2308, ...
  2404, 2596, 2788, 2980, 3172, 3364, 3556, 3748, 3940, 4132, ...
  4324, 4516, 4708, 4900, 5092, 5284, 5476, 5668, 5860, 6052, 6244};
for a = 1:length(numlist)
  txt = sprintf(strcat('%d~U~',diversionString,rangeString), numlist{a});
  C = {txt};
  writecell(C,filename,'Sheet',1,'Range',sprintf('A%d',a+4));
  writematrix(UConf(numlist{a},:),filename,'Sheet',1,'Range',sprintf('B%d',a+4));
end

%UTRUConf
numlist = {1060, 1156, 1252, 1348, 1444,1540,1636,1732,1828,1924,2020,2115,2212,2308, ...
  2404, 2596, 2788, 2980, 3172, 3364, 3556, 3748, 3940, 4132, ...
  4324, 4516, 4708, 4900, 5092, 5284, 5476, 5668, 5860, 6052, 6244};
for a = 1:length(numlist)
  txt = sprintf(strcat('%d~UTRU~',diversionString,rangeString), numlist{a});
  C = {txt};
  writecell(C,filename,'Sheet',1,'Range',sprintf('A%d',a+49));
  writematrix(UTRUConf(numlist{a},:),filename,'Sheet',1,'Range',sprintf('B%d',a+49));
end

%FPWasteInv
numlist = {1115, 1210, 1306, 1402, 1499, 1594, 1690, 1786, 1883, 2075, 2266, 2459, 2651,
2843, ...
  3035, 3227, 3420, 3611, 3803, 3995, 4187, 4378, 4571, 4762, 4954, ...
  5146, 5338, 5531, 5723, 5915, 6107, 6300};
for a = 1:length(numlist)
  txt = sprintf(strcat('%d~FP~',diversionString,rangeString), numlist{a});
  C = {txt};
  writecell(C,filename,'Sheet',1,'Range',sprintf('A%d',a+99));
  writematrix(FPWasteInv(numlist{a},:),filename,'Sheet',1,'Range',sprintf('B%d',a+99));
end


%Metal Waste
numlist = {1060, 1155, 1252, 1347, 1444, 1539, 1635, 1827, 2019, 2211, 2307, 2499, 2691,
2883, 3075, ...
  3267, 3459, 3651, 3843, 4035, 4227, 4419, 4611, 4803, 4995, 5187, ...
  5379, 5571, 5763, 5955, 6147, 6339};
for a = 1:length(numlist)
  txt = sprintf(strcat('%d~MW~',diversionString,rangeString), numlist{a});
  C = {txt};
  writecell(C,filename,'Sheet',1,'Range',sprintf('A%d',a+149));
  writematrix(MetalConf(numlist{a},:),filename,'Sheet',1,'Range',sprintf('B%d',a+149));
end


C = {'createGeometry'};
writecell(C,filename,'Sheet',1,'Range','A1');

fprintf('Done!\n');

winopen('isotopics.xlsx');

%exit;

#########################################################################
```

C:\Users\student\Desktop\GADRASAddIn\GADRASAddIn\Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GADRASAddIn
{
    public partial class Form1 : Form
    {



        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

            checkedListBox1.Items.Add("Metal Waste", CheckState.Checked);
            checkedListBox1.Items.Add("Fission Product Waste",
              CheckState.Checked);
            checkedListBox1.Items.Add("U Ingot", CheckState.Checked);
            checkedListBox1.Items.Add("U/TRU Ingot", CheckState.Checked);
        }

        //comboBox1 is detector
        //comboBox2 is geometry
        //textBox1 is detector height (dimension)
        //textBox3 is detector width (dimension)
        //textBox4 is detector length (dimension)
        //textBox2 is shielding
        //textBox10 is time
        //textBox5 is detector distance
        //textBox8 is detector height (configuration)
        //textBox6 is elevation
        //textBox7 is latitude
        //textBox9 is longitude

        private void button1_Click(object sender, EventArgs e)
        {
            bool u = false;
            bool utru = false;
            bool fp = false;
            bool mw = false;

            for (int i=0; i < checkedListBox1.CheckedItems.Count; i++)
            {
                if (checkedListBox1.CheckedItems[i].ToString() == "U Ingot")
                {
                    u = true;
                }
                if (checkedListBox1.CheckedItems[i].ToString() == "U/TRU Ingot")
                {
                    utru = true;
                }
                if (checkedListBox1.CheckedItems[i].ToString() == "Fission 
                  Product Waste")
                {
                    fp = true;
                }
                if (checkedListBox1.CheckedItems[i].ToString() == "Metal Waste")
                {
                    mw = true;
                }
            }


            this.Hide();
            string detector;
            switch (comboBox1.Text)
            {
                case "HPGe":
                    detector = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";
                    break;
                case "NaI":
                    detector = "C:\\GADRAS\\Detector\\3x3\\NaI MidScat";
                    break;
                default:
                    detector = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";
                    break;
            }

            string geometry = comboBox2.Text;
            //default to spherical geometry
            if (geometry == "")
            {
                geometry = "Sphere";
            }

            Globals.ThisAddIn.Make1DMsAndRunTransportAndInject(detector, 
              geometry, textBox1.Text, textBox3.Text, textBox4.Text, 
              textBox2.Text, textBox10.Text, textBox5.Text, textBox8.Text, 
              textBox6.Text, textBox7.Text, textBox9.Text, u, utru, fp, mw, 
              textBox11.Text);

        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void comboBox3_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void label8_Click(object sender, EventArgs e)
        {

        }

        private void checkedListBox1_SelectedIndexChanged(object sender,
          EventArgs e)
        {

        }

        private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox4_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox10_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox5_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox8_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox6_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox7_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox9_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox11_TextChanged(object sender, EventArgs e)
        {

        }
    }
}

```

C:\Users\student\Desktop\GADRASAddIn\GADRASAddIn\ThisAddIn.cs

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Linq;
using Excel = Microsoft.Office.Interop.Excel;
using Office = Microsoft.Office.Core;
using Microsoft.Office.Tools.Excel;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using System.Xml;
using Microsoft.Office.Interop.Excel;
using Microsoft.Office.Core;
using System.Globalization;

namespace GADRASAddIn
{
    public partial class ThisAddIn
    {




        private void ThisAddIn_Startup(object sender, System.EventArgs e)
        {
            //simply adds our main function as a handler for the workbook activate event
            this.Application.WorkbookActivate += Application_WorkbookActivate;
        }


        private void Application_WorkbookActivate(Excel.Workbook Wb)
        {
            //here the worksheet had been activated and we will be able to obtain data from it
            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];



            //check to see if run flag written by MATLAB is 1, if so run the program
            Excel.Range flagRange = worksheet.Range["A1"];
            if (flagRange.Cells[1, 1].Text == "createGeometry")
            {

                Form1 myForm = new Form1();
                myForm.ShowDialog();

                //this is the old way to activate the next step
                //BuildModelsAndRunTransportAndInject();
            }

            if (flagRange.Cells[1, 1].Text == "totalGamma")
            {
                CreateTotalGammaChart();
            }

            if (flagRange.Cells[1, 1].Text == "gammaChannels")
            {
                CreateGammaChart();
            }

            //only need spherical for now, will ask if other geometries are
              relevant
            //rn cylinder and slab are only relevant for ingots
            //WriteCylinderXML("test", 5, true);
            //WriteSlabXML("test", 47, true);
            //WriteSphereXML("test", 47, "15", true);
        }




        public void CreateTotalGammaChart()
        {
            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            //reading from the data transfer file
            StreamReader sr = new StreamReader(Environment.GetFolderPath
              (Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\
              \dataForGammaAnalysis.txt");

            worksheet.Range["A2", "J500"].Clear();

            string line = sr.ReadLine();
            //loss scenario first
            int index = 2;
            while (line != "$")
            {
                //first write the time in
                worksheet.Range["A" + index].Cells[1, 1].Value2 = line.Split('~')
                  [0];


                if (line.Split('~')[1] == "U")
                {
                    worksheet.Range["B" + index].Cells[1, 1].Value2 = line.Split
                     ('~')[4];
                }
                if (line.Split('~')[1] == "UTRU")
                {
                    worksheet.Range["C" + index].Cells[1, 1].Value2 = line.Split
                     ('~')[4];
                }
```

```
                    if (line.Split('~')[1] == "MW")
                    {
                        worksheet.Range["D" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    if (line.Split('~')[1] == "FP")
                    {
                        worksheet.Range["E" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    index++;
                    line = sr.ReadLine();
                }
                //save this for later
                int maxIndex1 = index;
                index = 2;
                line = sr.ReadLine();
                //then do control scenario
                while (line != null)
                {
                    worksheet.Range["F" + index].Cells[1, 1].Value2 = line.Split('~')[0];


                    if (line.Split('~')[1] == "U")
                    {
                        worksheet.Range["G" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    if (line.Split('~')[1] == "UTRU")
                    {
                        worksheet.Range["H" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    if (line.Split('~')[1] == "MW")
                    {
                        worksheet.Range["I" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    if (line.Split('~')[1] == "FP")
                    {
                        worksheet.Range["J" + index].Cells[1, 1].Value2 = line.Split('~')[4];
                    }
                    index++;
                    line = sr.ReadLine();
                }

                int maxIndex2 = index;

                sr.Close();
                //File.Delete("C:\\Users\\student\\Desktop\
                  \dataForGammaAnalysis.txt");


                worksheet.Shapes.AddChart2(201, Excel.XlChartType.xlLine).Select();
                Excel.Chart cha = Application.ActiveChart;
                cha.ApplyChartTemplate(Environment.GetFolderPath
                  (Environment.SpecialFolder.Desktop) + "\\..\\AppData\\Roaming\
                  \Microsoft\\Templates\\Charts\\TotalGammaChart.crtx");
                //cha.ApplyChartTemplate("C:\\Users\\student\\AppData\\Roaming\
                  \Microsoft\\Templates\\Charts\\tryAgain.crtx");
                //cha.SetSourceData(worksheet.Range["Sheet1!$C$2:$E$6"]);

                cha.ChartTitle.Text = "Loss Scenario Over Time";
                cha.Axes(Excel.XlAxisType.xlValue,
                  Excel.XlAxisGroup.xlPrimary).AxisTitle.Text = "Gamma CPS"; //Set Y-
                  Axis
                cha.Axes(Excel.XlAxisType.xlCategory,
                  Excel.XlAxisGroup.xlPrimary).AxisTitle.Text = "Time"; //Set X-Axis

                //LOSS SCENARIO
                cha.SeriesCollection(1).XValues = worksheet.Range["Sheet1!$A$2:$A$" +
                   maxIndex1];
                cha.SeriesCollection(1).Values = worksheet.Range["Sheet1!$B$2:$B$" +
                  maxIndex1];
                cha.SeriesCollection(1).Name = "U (Loss)";

                cha.SeriesCollection(2).XValues = worksheet.Range["Sheet1!$A$2:$A$" +
                   maxIndex1];
                cha.SeriesCollection(2).Values = worksheet.Range["Sheet1!$C$2:$C$" +
                  maxIndex1];
                cha.SeriesCollection(2).Name = "UTRU (Loss)";

                cha.SeriesCollection(3).XValues = worksheet.Range["Sheet1!$A$2:$A$" +
                   maxIndex1];
                cha.SeriesCollection(3).Values = worksheet.Range["Sheet1!$D$2:$D$" +
                  maxIndex1];
                cha.SeriesCollection(3).Name = "MW (Loss)";

                cha.SeriesCollection(4).XValues = worksheet.Range["Sheet1!$A$2:$A$" +
                   maxIndex1];
                cha.SeriesCollection(4).Values = worksheet.Range["Sheet1!$E$2:$E$" +
                  maxIndex1];
                cha.SeriesCollection(4).Name = "FP (Loss)";


                //CONTROL SCENARIO
                cha.SeriesCollection(5).XValues = worksheet.Range["Sheet1!$F$2:$F$" +
                   maxIndex2];
                cha.SeriesCollection(5).Values = worksheet.Range["Sheet1!$G$2:$G$" +
                  maxIndex2];
                cha.SeriesCollection(5).Name = "U (Control)";

```

```
            cha.SeriesCollection(6).XValues = worksheet.Range["Sheet1!$F$2:$F$" + 
              maxIndex2];
            cha.SeriesCollection(6).Values = worksheet.Range["Sheet1!$H$2:$H$" + 
              maxIndex2];
            cha.SeriesCollection(6).Name = "UTRU (Control)";

            cha.SeriesCollection(7).XValues = worksheet.Range["Sheet1!$F$2:$F$" + 
              maxIndex2];
            cha.SeriesCollection(7).Values = worksheet.Range["Sheet1!$I$2:$I$" + 
              maxIndex2];
            cha.SeriesCollection(7).Name = "MW (Control)";


            cha.SeriesCollection(8).XValues = worksheet.Range["Sheet1!$F$2:$F$" + 
              maxIndex2];
            cha.SeriesCollection(8).Values = worksheet.Range["Sheet1!$J$2:$J$" + 
              maxIndex2];
            cha.SeriesCollection(8).Name = "FP (Control)";

        }

        public void CreateGammaChart()
        {
            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            //reading from the data transfer file
            StreamReader sr = new StreamReader(Environment.GetFolderPath
              (Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\
              \dataForGammaAnalysis.txt");

            worksheet.Range["A2", "J500"].Clear();

            string line = sr.ReadLine();
            //loss scenario first
            int index = 2;
            while (line != "$")
            {
                //first write the time in
                worksheet.Range["A" + index].Cells[1, 1].Value2 = line.Split('~')
                  [0];

                worksheet.Range["B" + index].Cells[1, 1].Value2 = line.Split('~')
                  [1];

                index++;
                line = sr.ReadLine();
            }
            //save this for later
            int maxIndex1 = index;
            index = 2;
            line = sr.ReadLine();
            //then do control scenario
            while (line != null)
            {
                worksheet.Range["C" + index].Cells[1, 1].Value2 = line.Split('~')[0];

                worksheet.Range["D" + index].Cells[1, 1].Value2 = line.Split('~')[1];

                index++;
                line = sr.ReadLine();
            }

            int maxIndex2 = index;

            sr.Close();
            //File.Delete("C:\\Users\\student\\Desktop\\dataForGammaAnalysis.txt");


            worksheet.Shapes.AddChart2(201, Excel.XlChartType.xlLine).Select();
            Excel.Chart cha = Application.ActiveChart;

            cha.ApplyChartTemplate(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\..\\AppData\\Roaming\\Microsoft\\Templates\\Charts\\GammaChannelsChart.crtx");
            //cha.ApplyChartTemplate("C:\\Users\\student\\AppData\\Roaming\\Microsoft\\Templates\\Charts\\tryAgain.crtx");
            //cha.SetSourceData(worksheet.Range["Sheet1!$C$2:$E$6"]);

            //int someTime = 11213312; //LOL

            cha.ChartTitle.Text = "Loss Scenario Over Energy at ";// + someTime;
            cha.Axes(Excel.XlAxisType.xlValue, Excel.XlAxisGroup.xlPrimary).AxisTitle.Text = "Gamma CPS"; //Set Y-Axis
            cha.Axes(Excel.XlAxisType.xlCategory, Excel.XlAxisGroup.xlPrimary).AxisTitle.Text = "Energy"; //Set X-Axis

            //LOSS SCENARIO
            cha.SeriesCollection(2).XValues = worksheet.Range["Sheet1!$A$2:$A$" + maxIndex1];
            cha.SeriesCollection(2).Values = worksheet.Range["Sheet1!$B$2:$B$" + maxIndex1];
            cha.SeriesCollection(2).Name = "Loss Scenario";


            cha.SeriesCollection(1).XValues = worksheet.Range["Sheet1!$C$2:$C$" + maxIndex2];
            cha.SeriesCollection(1).Values = worksheet.Range["Sheet1!$D$2:$D$" + maxIndex2];
            cha.SeriesCollection(1).Name = "Control Scenario";

```

```
        }

        public void Make1DMsAndRunTransportAndInject(string detector, string geometry, string height, string width, string length, string shielding, string time, string distance, string detHeight, string elevation, string latitude, string longitude, bool u, bool utru, bool fp, bool mw, string coreNumber)
        {

            //also here is where we write to XML now using guidance from Form1

            BuildModels(geometry, float.Parse(shielding, CultureInfo.InvariantCulture.NumberFormat), u, utru, fp, mw);


            ProcessStartInfo startInfo = new ProcessStartInfo(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\\TransportAndInject\\TransportAndInject\\bin\\Debug\\net35\\TransportAndInject.exe");
            startInfo.WindowStyle = ProcessWindowStyle.Normal;
            startInfo.Arguments = detector + " " + geometry + " " + height + " " + width + " " + length + " " + shielding + " " + time + " " + distance + " " + detHeight + " " + elevation + " " + latitude + " " + longitude + " " + coreNumber;// e.g. "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";
            Process.Start(startInfo);

            //set the flag to zero so that we don't run transport on reopen
            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            Excel.Range flagRange = worksheet.Range["A1"];
            flagRange.Cells[1, 1].Value2 = 0;

        }



        private void BuildModels(string geometry, float shielding, bool u, bool utru, bool fp, bool mw)
        {
            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            Excel.Range flagRange = worksheet.Range["A1"];
            string currentString = "";

            //U PRODUCT
            if (u)
            {
                for (var i = 5; i < 50; i++) //change this to alter how much of the excel spreadsheet is processed
                {
                    currentString = worksheet.Range["A" + i].Cells[1, 1].Text;
                    if (currentString != "")
                    {
                        //WriteCylinderXML(currentString, i);
                        //WriteSlabXML(currentString, i);
                        if (geometry == "Sphere")
                            WriteSphereXML(currentString, i, (shielding * 
                    10).ToString());
                        if (geometry == "Cylinder")
                            WriteCylinderXML(currentString, i, (shielding * 
                    10).ToString());
                        if (geometry == "Slab")
                            WriteSlabXML(currentString, i, (shielding * 
                    10).ToString());
                    }
                }
            }


            //U-TRU PRODUCT
            if (utru)
            {
                for (var i = 50; i < 100; i++) //change this to alter how much of 
                   the excel spreadsheet is processed
                {
                    currentString = worksheet.Range["A" + i].Cells[1, 1].Text;
                    if (currentString != "")
                    {
                        if (geometry == "Cylinder")
                            WriteCylinderXML(currentString, i, (shielding * 
                    150).ToString());
                        if (geometry == "Slab")
                            WriteSlabXML(currentString, i, (shielding * 
                    150).ToString());
                        if (geometry == "Sphere")
                            WriteSphereXML(currentString, i, (shielding * 
                    150).ToString());
                        //WriteSphereXML(currentString, i, "100",true);
                    }
                }
            }


            //FISSION PRODUCT WASTE
            if (fp)
            {
                for (var i = 100; i < 150; i++) //change this to alter how much 
                  of the excel spreadsheet is processed
                {
                    currentString = worksheet.Range["A" + i].Cells[1, 1].Text;
                    if (currentString != "")
                    {
                        if (geometry == "Sphere")
                            WriteSphereXML(currentString, i, (shielding * 
                    25).ToString());
```

```
                            if (geometry == "Cylinder")
                                WriteCylinderXML(currentString, i, (shielding * 
                        25).ToString());
                            if (geometry == "Slab")
                                WriteSlabXML(currentString, i, (shielding * 
                        25).ToString());
                        }
                    }
                }


                //METAL WASTE
                if (mw)
                {
                    for (var i = 150; i < 200; i++) //change this to alter how much 
                      of the excel spreadsheet is processed
                    {
                        currentString = worksheet.Range["A" + i].Cells[1, 1].Text;
                        if (currentString != "")
                        {
                            if (geometry == "Sphere")
                                WriteSphereXML(currentString, i, (shielding * 
                        12).ToString());
                            if (geometry == "Cylinder")
                                WriteCylinderXML(currentString, i, (shielding * 
                        12).ToString());
                            if (geometry == "Slab")
                                WriteSlabXML(currentString, i, (shielding * 
                        12).ToString());
                        }
                    }
                }




                //flagRange.Cells[1, 1].Value2 = 0;
                //MessageBox.Show("Done writing XML Files! ");

                //~~~~~~~~~~~~~~~~ RUN TRANSPORT AND INJECT NOW 
                  ~~~~~~~~~~~~~~~~~~~~~~

                /*
                ProcessStartInfo startInfo = new ProcessStartInfo("C:\\Users\\student
                  \\Desktop\\TransportAndInject\\TransportAndInject\\bin\\Debug\
                  \net35\\TransportAndInject.exe");
                startInfo.WindowStyle = ProcessWindowStyle.Normal;
                startInfo.Arguments = "hello";
                Process.Start(startInfo);
                */

                //need to pass arguments to the transport function that we collect 
```

```
              from Form1
            //also this is the old way of calling this
            //Process.Start("C:\\Users\\student\\Desktop\\TransportAndInject\
              \TransportAndInject\\bin\\Debug\\net35\
              \TransportAndInject.exe"); // or we can just directly start the
              TransportAndInject from here

            return;
        }









        private void WriteSlabXML(string s, int index, string shieldWidth = "15",
           bool test = false)
        {
            //MessageBox.Show("going to try and write an XML File! ");

            //XmlTextWriter writer = new XmlTextWriter("C:\\Users\\student\
              \Desktop\\1dm_files\\" + s+"Slab.1dm", System.Text.Encoding.UTF8);
            XmlTextWriter writer;
            if (test)
            {
                writer = new XmlTextWriter(Environment.GetFolderPath
                  (Environment.SpecialFolder.Desktop) + "\\" + s + "~Slab.1dm",
                  System.Text.Encoding.UTF8);
            }
            else
            {
                writer = new XmlTextWriter("C:\\GADRAS\\Source\\ApiTestSources\
                  \ApiTest_ParallelGADRAS\\" + s + "~Slab.1dm",
                  System.Text.Encoding.UTF8);
            }
            writer.WriteStartDocument(true);
            writer.Formatting = Formatting.Indented;
            writer.Indentation = 2;
            writer.WriteStartElement("model");
            writer.WriteStartElement("version");
            writer.WriteString("18.8.10.0");
            writer.WriteEndElement();
            writer.WriteStartElement("description");
            writer.WriteEndElement();
            writer.WriteStartElement("intervals");

            writer.WriteStartElement("interval"); //start product interval
            writer.WriteStartElement("name");
            writer.WriteString("Pyro Product");
            writer.WriteEndElement();
            writer.WriteStartElement("width");
            writer.WriteString("1");
            writer.WriteEndElement();
            writer.WriteStartElement("density");
            writer.WriteString("19.4");
            writer.WriteEndElement();
            writer.WriteStartElement("age");
            writer.WriteString("20");
            writer.WriteEndElement();
            writer.WriteStartElement("state");
            writer.WriteString("solid");
            writer.WriteEndElement();
            writer.WriteStartElement("constituents");

            if (Application.ActiveWorkbook == null)
            {
                return;
            }

            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            Excel.Range range = worksheet.Range["B"+index, "BLM"+index]; //HERE 
              IS THE SPECIFIED RANGE
            Excel.Range isotopeReferenceRange = worksheet.Range["B4", "BLM4"]; // 
              HERE IS THE ISOTOPE REFERENCE RANGE

            for (int i = 1; i < 1677; i++)
            {
                if (isotopeReferenceRange.Cells[1, i].Text != "" && range.Cells 
                  [1, i].Text != "0")
                {
                    //MessageBox.Show(range.Cells[1,i].Text);
                    //MessageBox.Show(range.Cells[3,i].Text);
                    createNode(isotopeReferenceRange.Cells[1, i].Text, 
                     range.Cells[1, i].Text, writer);
                }
            }

            writer.WriteEndElement(); //end constituents

            writer.WriteStartElement("box");
            writer.WriteStartElement("width");
            writer.WriteString("300");
            writer.WriteEndElement();
            writer.WriteStartElement("height");
            writer.WriteString("100");
            writer.WriteEndElement();
            writer.WriteEndElement();

            writer.WriteEndElement(); //end product interval
            //--------------------------------------------------------------
            writer.WriteStartElement("interval"); //start shield interval
            writer.WriteStartElement("name");
```

```
                writer.WriteString("Lead (Pb)");
                writer.WriteEndElement();
                writer.WriteStartElement("width");
                writer.WriteString(shieldWidth);
                writer.WriteEndElement();
                writer.WriteStartElement("density");
                writer.WriteString("11.35");
                writer.WriteEndElement();
                writer.WriteStartElement("age");
                writer.WriteString("20");
                writer.WriteEndElement();
                writer.WriteStartElement("state");
                writer.WriteString("solid");
                writer.WriteEndElement();
                writer.WriteStartElement("constituents");

                writer.WriteStartElement("constituent");
                writer.WriteStartElement("name");
                writer.WriteString("Pb");
                writer.WriteEndElement();
                writer.WriteStartElement("mass_fraction");
                writer.WriteString("1");
                writer.WriteEndElement();
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteStartElement("box");
                writer.WriteStartElement("width");
                writer.WriteString("330");
                writer.WriteEndElement();
                writer.WriteStartElement("height");
                writer.WriteString("131");
                writer.WriteEndElement();
                writer.WriteEndElement();
                writer.WriteEndElement(); //end shield interval

                writer.WriteEndDocument();
                writer.Close();
                //MessageBox.Show("XML File created ! ");
            }


            private void WriteCylinderXML(string s, int index,string shieldWidth = 
              "30", bool test = false)
            {
                //MessageBox.Show("going to try and write an XML File! ");

                //XmlTextWriter writer = new XmlTextWriter("C:\\Users\\student\
                  \Desktop\\1dm_files\\" + s + ".1dm", System.Text.Encoding.UTF8);
                XmlTextWriter writer;
                if (test)
                {
```

```
                writer = new XmlTextWriter(Environment.GetFolderPath
                  (Environment.SpecialFolder.Desktop) + "\\" + s +
                  "~Cylinder.1dm", System.Text.Encoding.UTF8);
            } else
            {
                writer = new XmlTextWriter("C:\\GADRAS\\Source\\ApiTestSources\
                  \ApiTest_ParallelGADRAS\\" + s + "~Cylinder.1dm",
                  System.Text.Encoding.UTF8);
            }
            writer.WriteStartDocument(true);
            writer.Formatting = Formatting.Indented;
            writer.Indentation = 2;
            writer.WriteStartElement("model");
            writer.WriteStartElement("version");
            writer.WriteString("18.8.10.0");
            writer.WriteEndElement();
            writer.WriteStartElement("description");
            writer.WriteEndElement();
            writer.WriteStartElement("intervals");

            writer.WriteStartElement("interval"); //start product interval
            writer.WriteStartElement("name");
            writer.WriteString("Pyro Product");
            writer.WriteEndElement();
            writer.WriteStartElement("width");
            writer.WriteString("2.5");
            writer.WriteEndElement();
            writer.WriteStartElement("density");
            writer.WriteString("19.4");
            writer.WriteEndElement();
            writer.WriteStartElement("age");
            writer.WriteString("20");
            writer.WriteEndElement();
            writer.WriteStartElement("state");
            writer.WriteString("solid");
            writer.WriteEndElement();
            writer.WriteStartElement("constituents");

            if (Application.ActiveWorkbook == null)
            {
                return;
            }

            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            Excel.Range range = worksheet.Range["B" + index, "BLM" + index]; //
              HERE IS THE SPECIFIED RANGE
            Excel.Range isotopeReferenceRange = worksheet.Range["B4", "BLM4"]; //
              HERE IS THE ISOTOPE REFERENCE RANGE

            for (int i = 1; i < 1677; i++)
            {
                if (isotopeReferenceRange.Cells[1, i].Text != "" && range.Cells
                  [1, i].Text != "0")
                {
                    //MessageBox.Show(range.Cells[1,i].Text);
                    //MessageBox.Show(range.Cells[3,i].Text);
                    createNode(isotopeReferenceRange.Cells[1, i].Text,
                     range.Cells[1, i].Text, writer);
                }
            }

            writer.WriteEndElement(); //end constituents

            writer.WriteStartElement("box");
            writer.WriteStartElement("width");
            writer.WriteString("300");
            writer.WriteEndElement();
            writer.WriteStartElement("height");
            writer.WriteString("5");
            writer.WriteEndElement();
            writer.WriteEndElement();

            writer.WriteEndElement(); //end product interval
            //----------------------------------------------------------------
            writer.WriteStartElement("interval"); //start shield interval
            writer.WriteStartElement("name");
            writer.WriteString("Lead (Pb)");
            writer.WriteEndElement();
            writer.WriteStartElement("width");
            writer.WriteString(shieldWidth);
            writer.WriteEndElement();
            writer.WriteStartElement("density");
            writer.WriteString("11.35");
            writer.WriteEndElement();
            writer.WriteStartElement("age");
            writer.WriteString("20");
            writer.WriteEndElement();
            writer.WriteStartElement("state");
            writer.WriteString("solid");
            writer.WriteEndElement();
            writer.WriteStartElement("constituents");

            writer.WriteStartElement("constituent");
            writer.WriteStartElement("name");
            writer.WriteString("Pb");
            writer.WriteEndElement();
            writer.WriteStartElement("mass_fraction");
            writer.WriteString("1");
            writer.WriteEndElement();
            writer.WriteEndElement();

            writer.WriteEndElement();
            writer.WriteStartElement("box");
            writer.WriteStartElement("width");
            writer.WriteString("330");
            writer.WriteEndElement();
            writer.WriteStartElement("height");
            writer.WriteString(shieldWidth);
            writer.WriteEndElement();
            writer.WriteEndElement();
            writer.WriteEndElement(); //end shield interval

            writer.WriteEndDocument();
            writer.Close();
            //MessageBox.Show("XML File created ! ");
        }



        private void WriteSphereXML(string s, int index, string shieldWidth =
          "15", bool test = false)
        {
            //XmlTextWriter writer = new XmlTextWriter("C:\\Users\\student\
              \Desktop\\1dm_files\\" + s+"Slab.1dm", System.Text.Encoding.UTF8);
            XmlTextWriter writer;
            if (test)
            {
                writer = new XmlTextWriter(Environment.GetFolderPath
                  (Environment.SpecialFolder.Desktop) + "\\" + s + "~Sphere.1dm",
                   System.Text.Encoding.UTF8);
            }
            else
            {
                writer = new XmlTextWriter("C:\\GADRAS\\Source\\ApiTestSources\
                  \ApiTest_ParallelGADRAS\\" + s + "~Sphere.1dm",
                  System.Text.Encoding.UTF8);
            }
            writer.WriteStartDocument(true);
            writer.Formatting = Formatting.Indented;
            writer.Indentation = 2;
            writer.WriteStartElement("model");
            writer.WriteStartElement("version");
            writer.WriteString("18.8.10.0");
            writer.WriteEndElement();
            writer.WriteStartElement("description");
            writer.WriteEndElement();
            writer.WriteStartElement("intervals");

            writer.WriteStartElement("interval"); //start product interval
            writer.WriteStartElement("name");
            writer.WriteString("Pyro Product");
            writer.WriteEndElement();
            writer.WriteStartElement("width");
            writer.WriteString("5");
            writer.WriteEndElement();
            writer.WriteStartElement("density");
            writer.WriteString("19.4");
            writer.WriteEndElement();
            writer.WriteStartElement("age");
            writer.WriteString("20");
            writer.WriteEndElement();
            writer.WriteStartElement("state");
            writer.WriteString("solid");
            writer.WriteEndElement();
            writer.WriteStartElement("constituents");

            if (Application.ActiveWorkbook == null)
            {
                return;
            }

            Excel.Worksheet worksheet = Application.ActiveWorkbook.Worksheets[1];
            Excel.Range range = worksheet.Range["B" + index, "BLM" + index]; // HERE IS THE SPECIFIED RANGE
            Excel.Range isotopeReferenceRange = worksheet.Range["B4", "BLM4"]; // HERE IS THE ISOTOPE REFERENCE RANGE

            for (int i = 1; i < 1677; i++)
            {
                if (isotopeReferenceRange.Cells[1, i].Text != "" && range.Cells[1, i].Text != "0")
                {
                    createNode(isotopeReferenceRange.Cells[1, i].Text, range.Cells[1, i].Text, writer);
                }
            }

            writer.WriteEndElement();
            writer.WriteStartElement("sphere");
            writer.WriteEndElement();
            writer.WriteEndElement(); //end product interval
            //--------------------------------------------------------------
            writer.WriteStartElement("interval"); //start shield interval
            writer.WriteStartElement("name");
            writer.WriteString("Lead (Pb)");
            writer.WriteEndElement();
            writer.WriteStartElement("width");
            writer.WriteString(shieldWidth);
            writer.WriteEndElement();
            writer.WriteStartElement("density");
            writer.WriteString("11.35");
            writer.WriteEndElement();
            writer.WriteStartElement("age");
            writer.WriteString("20");
            writer.WriteEndElement();
            writer.WriteStartElement("state");
            writer.WriteString("solid");
            writer.WriteEndElement();
```

```
                writer.WriteStartElement("constituents");

                writer.WriteStartElement("constituent");
                writer.WriteStartElement("name");
                writer.WriteString("Pb");
                writer.WriteEndElement();
                writer.WriteStartElement("mass_fraction");
                writer.WriteString("1");
                writer.WriteEndElement();
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteStartElement("sphere");
                writer.WriteEndElement(); //end shield interval

                writer.WriteEndDocument();
                writer.Close();
            }



            private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
            {
            }

            private void createNode(string name, string massFraction, XmlTextWriter 
              writer)
            {
                writer.WriteStartElement("constituent");
                writer.WriteStartElement("name");
                writer.WriteString(name);
                writer.WriteEndElement();
                writer.WriteStartElement("mass_fraction");
                writer.WriteString(massFraction);
                writer.WriteEndElement();
                writer.WriteEndElement();
            }


            #region VSTO generated code

            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InternalStartup()
            {
                this.Startup += new System.EventHandler(ThisAddIn_Startup);
                this.Shutdown += new System.EventHandler(ThisAddIn_Shutdown);
            }

            #endregion
```

```
    }



}

```

```csharp
using System;
using System.IO;
using System.Collections.Generic;
using System.Diagnostics;
using System.Xml;
using Sandia.Gadras.API;
using System.Globalization;

namespace TransportAndInject
{
    class Program
    {


        static void Main(string[] args)
        {

            //detector geometry height(dimension) width(dimension) length(dimension) shielding time distance height(configuration) elevation latitude longitude
            if (true)
            {
                //this code expects a detector to be passed in via command args
                ParallelTransport.RunTransport(args[0], args[1], args[2], args[3], args[4], args[5], args[6], args[7], args[8], args[9], args[10], args[11], args[12]);
                //i think i need a try/catch for the code below? in case target directory already exists?
                System.IO.Directory.Move("C:\\GADRAS\\Source\\ApiTestSources\\ApiTest_ParallelGADRAS", "C:\\GADRAS\\Source\\AIP_FINALOpen");//move files over to the inject directory
                System.IO.Directory.CreateDirectory("C:\\GADRAS\\Source\\ApiTestSources\\ApiTest_ParallelGADRAS");
            }


            try
            {
                string currentPcf = ParallelInject.RunInject(args[0], args[1], args[2], args[3], args[4], args[5], args[6], args[7], args[8], args[9], args[10], args[11], args[12]);
            }
            catch (System.Exception e)
            {
                string currentPcf = ParallelInject.RunInject(args[0], args[1], args[2], args[3], args[4], args[5], args[6], args[7], args[8], args[9], args[10], args[11], args[12]);
            }


            System.IO.Directory.Delete("C:\\GADRAS\\Source\\AIP_FINALOpen", true);
            //Once transport and inject are complete, we can call the analysis binary
            //ProcessStartInfo startInfo = new ProcessStartInfo("C:\\Users\\student\\Desktop\\TransportAndInject\\TransportAndInject\\bin\\Debug\\net35\\TransportAndInject.exe");
            ProcessStartInfo startInfo = new ProcessStartInfo(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\\GammaAnalysis\\GammaAnalysis\\bin\\Debug\\net35\\GammaAnalysis.exe");
            startInfo.WindowStyle = ProcessWindowStyle.Normal;
            startInfo.Arguments = "hello";// e.g. "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";
            Process.Start(startInfo);


            ApiExampleHelper.ApiExampleHelper.Exit(0, "Success", true);                               //Success


        }



    }

    class ParallelTransport //########################### START TRANSPORT ####################################################
    {
        static int m_numberOfCores = 8;                                                 //Number of cores to use simultaneously
        static string m_modelFileDirectory = "C:\\GADRAS\\Source\\ApiTestSources\\ApiTest_ParallelGADRAS";//..\\..\\..\\ModelFiles";                //   Directory containing model 1dm files
        static string m_apiServicePath = "C:\\GADRAS\\Program\\GadrasAPIServer.exe"; //Path to API server executable
        static string m_tempFolder = "C:\\GADRAS\\Temp";                                 //Path to a dir that is root of all temp dirs used here
        static string m_gadrasRoot = "C:\\GADRAS";                                        //Path to root GADRAS directory that contains data files
        static string m_startingDetector = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";//"ApiTestDetectors\\3x3_ParallelInject";//"3x3\\NaI MidScat";                   //Stargin detector directory for parallel instances
        static List<string> m_modelFiles;                                                     //List of 1dm model files from model file directory

        private static bool m_verbose =
          true;                                           //Whether anything is
          printed to the console during execution.

        #region Helper functions

        static List<string> getFiles(string p_Directory, string p_pattern)
        {
            System.IO.DirectoryInfo di = new System.IO.DirectoryInfo
              (p_Directory);
            System.IO.FileInfo[] fiList = di.GetFiles(p_pattern);
            List<string> files = new List<string>();

            foreach (System.IO.FileInfo fi in fiList)
            {
                files.Add(fi.FullName);
            }

            return files;
        }
        #endregion

        #region Handlers
        static void TransportCompletedHandler(object state,
          System.ComponentModel.RunWorkerCompletedEventArgs e)
        {
            Sandia.Gadras.API.ParallelTransportResults results =
              (Sandia.Gadras.API.ParallelTransportResults)e.Result;
            bool noErrors = results.CompletedSuccessfully;
            string errorMessage = results.ErrorMessage;

            if (m_verbose)
            {
                if (noErrors)
                    Console.WriteLine(string.Format("No errors: {0}",
                     System.IO.Path.GetFileName(results.InputModelFileName)));
                else
                    Console.WriteLine(errorMessage);
            }
        }

        static void ProcessExitHandler(object state, EventArgs e)
        {
            if (m_verbose)
            {
                System.Diagnostics.Process p = (System.Diagnostics.Process)state;
                if (p.ExitCode != 0)
                {
                    Console.WriteLine("[--------------------------------");
                    Console.WriteLine("Service exited with error: {0}",
                     p.ExitCode);
                    Console.Write(p.StandardOutput.ReadToEnd());
                    Console.WriteLine("--------------------------------]");
                }
            }
        }
        #endregion

        //static void Main(string[] args)
        public static void RunTransport(string detector, string geometry, string height, string width, string length, string shielding, string time, string distance, string detHeight, string elevation, string latitude, string longitude, string cores)//string[] args)
        {
            //m_verbose = (args.Length == 0 || args[0] == "-v");                          //Verbose if no args, or first arg is "-v"
            // additional args can be implemented to allow user to specify m_numberOfCores, m_modelFileDirectory/m_modelFiles, m_apiServicePath, m_tempFolder, or m_gadrasRoot

            m_modelFiles = getFiles(m_modelFileDirectory, "*.1dm");                              //Initialize list of model files for transport

            m_startingDetector = detector;

            Console.WriteLine("~~~ Start Transport ~~~");

            m_numberOfCores = int.Parse(cores);

            if (m_verbose)
            {
                Console.WriteLine("Using {0} cores to process {1} files from {2}:", m_numberOfCores, m_modelFiles.Count, System.IO.Path.GetFullPath(m_modelFileDirectory));
                foreach (var modelFile in m_modelFiles)
                {
                    Console.WriteLine("  {0}", System.IO.Path.GetFileName(modelFile));
                }
            }

            // set workspace to be application directory
            string workspaceDirectory = Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
            Sandia.Gadras.API.ParallelFunctions pf = new Sandia.Gadras.API.ParallelFunctions(m_apiServicePath, ProcessExitHandler, m_tempFolder, m_gadrasRoot, m_startingDetector, m_numberOfCores); // initialize parallel functionality

            pf.Transport(m_modelFiles, workspaceDirectory, TransportCompletedHandler); // make the call

            pf.ShutDown();
```

```csharp

            if (m_verbose)
            {
                //Console.WriteLine("Press any key to continue...");
                //Console.ReadKey(true); // Commented for ApiTests
            }
            Console.WriteLine("~~~ End Transport ~~~");
            Console.WriteLine("\n");

            //ApiExampleHelper.ApiExampleHelper.Exit(0, "Success", true);                         //Success, need to comment this out for automatic sequential execution
        }
    } //########################### END TRANSPORT ####################################################












    class ParallelInject //################################## START INJECT ######################################
    {
        static int m_numberOfCores = 8;             //Max number of cores to use in this example
        static string m_gadrasRoot = "C:\\GADRAS";  //Path to root GADRAS directory that contains data files
        static string m_currentDetector = "HPGe\\HPGe95%";
        static string m_gamFileDirectory = "C:\\GADRAS\\Source\\AIP_FINALOpen";// ParallelModelFiles"; // "..\\..\\..\\GamFiles";  //Directory containing source gam files
        static string m_apiServicePath = "C:\\GADRAS\\Program\\GadrasAPIServer.exe"; //Path to API server executable
        static string m_tempFolder = "C:\\GADRAS\\Temp";                              //Path to a dir that is root of all temp dirs used here
        static string m_startingDetector = "HPGe\\HPGe95%";   //Starting detector directory for parallel instances
        static List<string> m_gamFiles;                                               //List of gam source files from gam file directory
        static List<Sandia.Gadras.API.ParallelInjectInput> m_injInputs;            //Inject setup files for each source produced by transport
        static Sandia.Gadras.API.GadrasAPIWrapper m_gadrasAPI;                      //Required for checking source string
```

```
                 and inject setup

        private static bool m_verbose =
          true;                                       //Whether anything is
          printed to the console during execution.



        //static string m_pcfFile = "ControlRecord.pcf";      // PCF file that
          all records are collected to
        static string m_pcfFile = "default.pcf";      // PCF file that all
          records are collected to


        #region Helper functions

        /// <summary>
        /// Get list of files from given directory matching given search pattern
        /// </summary>
        /// <param name="p_Directory">Directory to search for files</param>
        /// <param name="p_pattern">Pattern for matching files</param>
        /// <returns></returns>
        static List<string> getFiles(string p_Directory, string p_pattern)
        {
            System.IO.DirectoryInfo di = new System.IO.DirectoryInfo
              (p_Directory);
            System.IO.FileInfo[] fiList = di.GetFiles(p_pattern);
            List<string> files = new List<string>();

            foreach (System.IO.FileInfo fi in fiList)
            {
                files.Add(fi.FullName);
            }

            return files;
        }

        /// <summary>
        /// Copy given file to detector directory
        /// </summary>
        /// <param name="p_fileName">File to copy to detector directory</param>
        static void copyFileToDetectorDirectory(string p_fileName)
        {
            string detectorDirectory =
                System.IO.Path.Combine(
                Sandia.Gadras.Utilities.Configs.API.DetectorDir,
                Sandia.Gadras.Utilities.Configs.API.CurrentDetector);

            string fullDestination = System.IO.Path.Combine(detectorDirectory,
              System.IO.Path.GetFileName(p_fileName));

            System.IO.File.Copy(p_fileName, System.IO.Path.GetFileName
                  (p_fileName), true);
        }
        /// <summary>
        /// Collects single record pcf files into single record
        /// </summary>
        /// <param name="pcfs"></param> Single record pcf files generated by
          ParallelInject
        static void gatherRecords(List<string> pcfs)
        {
            string mainPCF = System.IO.Path.GetFileNameWithoutExtension
              (m_pcfFile) + ".pcf";
            if (System.IO.File.Exists(mainPCF))
            {
                System.IO.File.Delete(mainPCF);
            }
            int i = 1;
            foreach (string p in pcfs)
            {
                m_gadrasAPI.spectraFileInsertData(p, mainPCF, i);
                System.IO.File.Delete(p);
                i++;
            }
        }

        /// <summary>
        /// Create inject setup
        /// </summary>
        /// <param name="p_gamFile">Gam file to use as source for inject</param>
        /// <returns>Inject setup</returns>
        static Sandia.Gadras.API.InjectSetup makeInjectSetup(string p_sourceName,
           string p_outputPCFFile, int p_outputPCFRecord, string height, string
          width, string length, string shielding, string time, string distance,
          string detHeight, string elevation, string latitude, string longitude)
        {
            Sandia.Gadras.API.InjectSetup injectSetup;
            string sourceName;

            injectSetup = new Sandia.Gadras.API.InjectSetup();

            try
            {
                sourceName = m_gadrasAPI.sourceCheckSourceString(p_sourceName);
            }
            catch (Sandia.Gadras.Utilities.GadrasUserException e)
            {
                Console.WriteLine("source " + p_sourceName + " not valid");
                throw e;
            }

            injectSetup.setDefaults(m_gadrasAPI);

            injectSetup.FileName = p_outputPCFFile;           //PCF filename full
```

```csharp
                  path
            if (string.IsNullOrEmpty(sourceName.Trim()))
            {
                injectSetup.Title = "Background";
            }
            else
            {
                injectSetup.Title = sourceName;
            }
            injectSetup.Record = p_outputPCFRecord;
            injectSetup.Source = sourceName;


            //If blank, will generate background record.
            //injectSetup.ContainsInternalSource = false;                                       //Looks for internal.pcf to include as internal source (common in LaBr detectors)
            //injectSetup.DetectorDeadTimeUs = 12;                                              //Dead time of detector per event (microseconds)
            injectSetup.DetectorHeightCm = float.Parse(detHeight, CultureInfo.InvariantCulture.NumberFormat);                   //Height of detector (cm)
            injectSetup.DistanceToSourceCm = float.Parse(distance, CultureInfo.InvariantCulture.NumberFormat);                  //Distance to source (cm)
            //injectSetup.DwellTimeIsLiveTime = false;                                          //If true, the DwellTimeSec is the live time, otherwise it's the real time
            injectSetup.DwellTimeSec = float.Parse(time, CultureInfo.InvariantCulture.NumberFormat);                            //Length of measurement
            //var eCal = new EnergyCalibration()
            //{
            //    Order0 = 0,
            //    Order1 = 3000,
            //    Order2 = 0,
            //    Order3 = 0,
            //    LowEnergy = 0
            //};
            //injectSetup.EnergyCalibration = eCal;                                             //Actual energy calibration of the recording
            //injectSetup.EnergyCalibrationFile = eCal;                                //Energy calibration of the recording written to file (can simulate uncalibrated detector)
            //var eRes = new EnergyResolution()                                //
            //{                 //
            //    FWHM = 3,    //
            //    Offset = 0,  //
            //    Power = 0,   //
            //    LowEnergySkew = 0,                                  //
            //    HighEnergySkew = 0,                                //
            //    SkewPower = 0,                                          //
            //    SkewExtent = 0                                         //
            //};                 //
            //injectSetup.EnergyResolution = eRes;                                //Energy resolution of the recording
            //injectSetup.IncludePoissonVariations = true;                             //Flag to simulate poisson variance in each channel (simulate measurement as opposed to perfect source)
            var locInfo = new LocationInfo()                                //
            {    //
                Elevation = float.Parse(elevation, CultureInfo.InvariantCulture.NumberFormat),                //
                Latitude = float.Parse(latitude, CultureInfo.InvariantCulture.NumberFormat),                //
                Longitude = float.Parse(longitude, CultureInfo.InvariantCulture.NumberFormat),              //
                Overburden = 0
```

```
                    //
            };
                          //
            //injectSetup.LocationInfo =
              locInfo;                                              //Location
              info for cosmic background (can be used to estimate terrestrial
              background)
            //injectSetup.IncludeCosmicBackground =
              true;                                      //Flag to put cosmic
              background in spectrum (calculated from location)
            //injectSetup.IncludeTerrestrialBackground =
              true;                                 //Flag to put terrestrial
              background inspectrum
            //injectSetup.NeuMeasEnv =
              NeutronMeasurementEnvironment.OUTSIDE_OR_LARGE_BAY;      //
            //var terrestrialBackground = new TerrestrialBackground
              ()                            //
            //
              {
                            //
            //    Attenuation =
              0,                                                          //
            //    K40 =
              0,
                   //
            //    Uranium =
              0,
               //
            //    Th232 =
              0,
                 //
            //    LowEnergyContinuum =
              0,                                                   //
            //    HighEnergyContinuum =
              0                                                   //
            //};
                            //
            //injectSetup.TerrestrialBackground =
              terrestrialBackground;                       //Terrestrial
              background contribution
            //injectSetup.TimeStamp =
              DateTime.Now;                                         //Time
              stamp to put on spectrum
            return injectSetup;
        }
        #endregion

        #region Handlers
        static void InjectCompletedHandler(object state,
          System.ComponentModel.RunWorkerCompletedEventArgs e)
        {
            Sandia.Gadras.API.ParallelInjectResults results =
                (Sandia.Gadras.API.ParallelInjectResults)e.Result;
            bool noErrors = results.CompletedSuccessfully;
            string errorMessage = results.ErrorMessage;

            if (m_verbose)
            {
                if (noErrors)
                    Console.WriteLine(string.Format("No errors: {0}",
                     System.IO.Path.GetFileName
                     (results.InputInjectSetup.FileName)));
                else
                    Console.WriteLine(errorMessage);
            }
        }

        static void ProcessExitHandler(object state, EventArgs e)
        {
            if (m_verbose)
            {
                System.Diagnostics.Process p = (System.Diagnostics.Process)state;
                if (p.ExitCode != 0)
                {
                    Console.WriteLine("[--------------------------------");
                    Console.WriteLine("Service exited with error: {0}",
                     p.ExitCode);
                    Console.Write(p.StandardOutput.ReadToEnd());
                    Console.WriteLine("--------------------------------]");
                }
            }
        }
        #endregion

        //static void Main(string[] args)
        public static string RunInject(string detector, string geometry, string
          height, string width, string length, string shielding, string time,
          string distance, string detHeight, string elevation, string latitude,
          string longitude, string cores)//string[] args)
        {
            //m_verbose = (args.Length == 0 || args[0] == "-
              v");                        //Verbose if no args, or first arg is "-
              v"
            // additional args can be implemented to allow user to specify
              m_numberOfCores, m_gadrasRoot, m_currentDetector,
              m_gamFileDirectory/list of sources, m_apiServicePath, or
              m_tempFolder

            m_startingDetector = m_currentDetector;
            if (detector == "C:\\GADRAS\\Detector\\3x3\\NaI MidScat\\")
            {
                m_startingDetector = "3x3\\NaI MidScat";
                m_currentDetector = "3x3\\NaI MidScat";
            }
```

```csharp

            m_numberOfCores = int.Parse(cores);

            m_gamFiles = getFiles(m_gamFileDirectory, "*.gam");                                    //Initialize list of gam files/source strings for inject

            Console.WriteLine("~~~ Start Inject ~~~");

            Console.WriteLine("Number of processors on this machine: " + Environment.ProcessorCount);
            if (m_verbose)
            {
                Console.WriteLine("Using {0} cores to process {1} files from {2}:", m_numberOfCores, m_gamFiles.Count, System.IO.Path.GetFullPath(m_gamFileDirectory));
                foreach (var gamFile in m_gamFiles)
                {
                    Console.WriteLine("  {0}", System.IO.Path.GetFileName(gamFile));
                }
            }

            m_gadrasAPI = ApiExampleHelper.ApiExampleHelper.setup(true);                            //Instantiate API wrapper to get injectSetup tools

            try
            {
                m_gadrasAPI.detectorSetCurrent(m_currentDetector);                            //Set current detector
            }
            catch (Exception)
            {
                ApiExampleHelper.ApiExampleHelper.Exit(-1, string.Format("Failed to set detector to {0}", m_currentDetector), m_verbose);
            }

            Sandia.Gadras.API.ParallelFunctions pf = new Sandia.Gadras.API.ParallelFunctions(m_apiServicePath, ProcessExitHandler, m_tempFolder, m_gadrasRoot, m_startingDetector, m_numberOfCores); // instantiate parallel functionality

            m_injInputs = new List<Sandia.Gadras.API.ParallelInjectInput>();
            List<string> pcfFiles = new List<string>();
            int i = 1;

            //in order for each pcf file to reflect the scenario it came from in name, we need to change m_pcfFile to reflect the diversion string minus time

            foreach (var gamFile in m_gamFiles)
```

```
            {
                // convert GAM file path into source name
                string sourceName = System.IO.Path.GetFileNameWithoutExtension
                  (gamFile);
                m_pcfFile = sourceName.Split('~')[2];
                //string pcfFile = m_pcfFile + "_" + i.ToString() + ".pcf"; //##
                  THIS IS THE LINE WE CHANGE
                string pcfFile = m_pcfFile + "_" + sourceName  + ".pcf";
                // write to the first record in the output file
                int pcfOutputRecord = 1;
                pcfFiles.Add(pcfFile);
                m_injInputs.Add(new Sandia.Gadras.API.ParallelInjectInput
                  (m_currentDetector, makeInjectSetup(sourceName, pcfFile,
                  pcfOutputRecord,height,  width,  length,  shielding,  time,
                  distance,  detHeight,  elevation,  latitude,  longitude)));
                i++;
            }

            Console.WriteLine("Make the call to Inject");
            long start = DateTime.Now.Ticks;// / TimeSpan.TicksPerMillisecond;
            pf.Inject(m_injInputs, InjectCompletedHandler); // make the call
            Console.WriteLine("Ready to Shutdown");
            long end = DateTime.Now.Ticks;
            pf.ShutDown();
            gatherRecords(pcfFiles);
            Console.WriteLine("Ran " + m_injInputs.Count + " injects, with " +
              m_numberOfCores + " cores, in " + (end - start) /
              TimeSpan.TicksPerMillisecond + "ms");
            long avgInjTime = (end - start) / (m_injInputs.Count *
              TimeSpan.TicksPerMillisecond);
            //Console.WriteLine("Average time " + avgInjTime + "ms/inject, Core
              time: " + avgInjTime*m_numberOfCores + "ms*core/inject ");

            Console.WriteLine("~~~ End Inject ~~~");

            return m_pcfFile;

            //ApiExampleHelper.ApiExampleHelper.Exit(0, "Success",
              true);                              //Success
        }
    } //###################### END INJECT #####################################




}

```

```
using System;
using System.IO;
using System.Collections.Generic;
using Sandia.Gadras.API;
using System.Diagnostics;

namespace GammaAnalysis
{
    class Program
    {


        public static int rowNumber;
        static void Main(string[] args)
        {




            Console.WriteLine("Please enter your detector. Options: HPGe
              (default), NaI");
            string detectorType = Console.ReadLine();
            if (detectorType == "")
            {
                detectorType = "HPGe";
            }
            Console.WriteLine("Please enter the loss scenario ID");
            string loss = Console.ReadLine();
            Console.WriteLine("Please enter the control scenario ID");
            string control = Console.ReadLine();
            Console.WriteLine("What would you like to do? Type 't' for total
              gamma analysis, 'c' for channel analysis");
            string analysisType = Console.ReadLine();

            string kmp = "";
            string time = "";
            if (analysisType == "c")
            {
                Console.WriteLine("Please enter the KMP you want to analyze.
                  Options: U, UTRU, FP, MW");
                kmp = Console.ReadLine();
                Console.WriteLine("Please enter the time you want to analyze");
                time = Console.ReadLine();
            }

            //if no args, then user has to specify the detector they will use

            //this reads the PCF files and writes to the data text file to be
              used by excel analysis
            ReadHeader.Read(loss, control, analysisType, detectorType, kmp,
              time);

            ApiExampleHelper.ApiExampleHelper.Exit(0, "Success", true);       // Success

        }


    }



    class ReadHeader
    {
        static string m_currentDetector = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%";
        static string m_pcfFileName = "project.pcf";
        static int m_pcfRecordIndex = 2;
        private static bool m_verbose = true;                                      //Whether anything is printed to the console during execution.

        public static void Read(string currentPcf, string controlPcf, string analysisType, string detectorType, string kmp, string time) //string[] args
        {
            //m_verbose = (args.Length == 0 || args[0] == "-v");                   //Verbose if no args, or first arg is "-v"

            Sandia.Gadras.API.GadrasAPIWrapper gadrasAPI = ApiExampleHelper.ApiExampleHelper.setup(m_verbose);  //Load settings and instantiate API


            //Add in changing detector type here
            string fullPcfFileName = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%\\" + currentPcf + ".pcf";
            string controlPcfFileName = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%\\" + controlPcf + ".pcf";

            if (detectorType == "HPGe")
            {
                fullPcfFileName = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%\\" + currentPcf + ".pcf";
                controlPcfFileName = "C:\\GADRAS\\Detector\\HPGe\\HPGe95%\\" + controlPcf + ".pcf";

            } else if (detectorType == "NaI")
            {
                fullPcfFileName = "C:\\GADRAS\\Detector\\3x3\\NaI MidScat\\" + currentPcf + ".pcf";
                controlPcfFileName = "C:\\GADRAS\\Detector\\3x3\\NaI MidScat\\" + controlPcf + ".pcf";
            }



            Sandia.Gadras.API.DetectorEnergyCalibration detectorEnergyCalibration
              = new Sandia.Gadras.API.DetectorEnergyCalibration();

            //##########
            gadrasAPI.spectraFileLoadRecord(fullPcfFileName,
             m_pcfRecordIndex).NeutronsCPS.ToString();

            detectorEnergyCalibration =
             gadrasAPI.spectraFileLoadRecordEnergyCalibration(fullPcfFileName,
             m_pcfRecordIndex);   //Make the call, read first reacord, index 1

            if
             (m_verbose)
              //Review the results
            {
              //Console.WriteLine("\nEnergy calibration constants in\n  {0}:
               \n", fullPcfFileName);
              //Console.WriteLine("Order0:    {0,10:F3}",
               detectorEnergyCalibration.Calibration.Order0);
              //Console.WriteLine("Order1:    {0,10:F3}",
               detectorEnergyCalibration.Calibration.Order1);
              //Console.WriteLine("Order2:    {0,10:F3}",
               detectorEnergyCalibration.Calibration.Order2);
              //Console.WriteLine("Order3:    {0,10:F3}",
               detectorEnergyCalibration.Calibration.Order3);
              //Console.WriteLine("LowEnergy: {0,10:F3}",
               detectorEnergyCalibration.Calibration.LowEnergy);
              //########
              //Console.WriteLine("\nNEUTRON CPS\n  {0}:\n",
               gadrasAPI.spectraFileLoadRecord(fullPcfFileName,
               m_pcfRecordIndex).NeutronsCPS.ToString());
              //Console.WriteLine("\nName\n  {0}:\n",
               gadrasAPI.spectraFileLoadRecord(fullPcfFileName,
               m_pcfRecordIndex).Title);
              //Console.WriteLine("\nChannel Gamma Counts\n  {0}\n",
               gadrasAPI.spectraFileLoadRecord(fullPcfFileName,
               m_pcfRecordIndex).GammaCounts[145].ToString());
              //can just multiply the cps by live time to get total count, but
               the cps is a scaled down version of the total count anyways so
               it's cool
              //Console.WriteLine("\nGamma CPS\n  {0}\n",
               gadrasAPI.spectraFileLoadRecord(fullPcfFileName,
               m_pcfRecordIndex).GammasCPS);

              StreamWriter sw = new StreamWriter(Environment.GetFolderPath
               (Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" +
               "\\dataForGammaAnalysis.txt");

            if (analysisType == "t")
            {
                string name = "";
                string counts = "";
                int numLossRecords = gadrasAPI.spectraFileGetInfo(fullPcfFileName).NumRecords;
                int numControlRecords = gadrasAPI.spectraFileGetInfo(controlPcfFileName).NumRecords;
                for (int recordIndex = 1; recordIndex < numLossRecords + 1; recordIndex++)
                {
                    name = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).Title;
                    counts = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).GammasCPS.ToString();


                    //this is the updated version of passing data to Excel
                    sw.WriteLine(name + "~" + counts.ToString());
                }

                sw.WriteLine("$"); // this divides the loss and control data

                for (int recordIndex = 1; recordIndex < numControlRecords + 1; recordIndex++)
                {
                    name = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).Title;
                    counts = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).GammasCPS.ToString();


                    //this is the updated version of passing data to Excel
                    sw.WriteLine(name + "~" + counts.ToString());
                }


                Process.Start(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\\totalGamma.xlsx");

            }
            else if (analysisType == "c")
            {
                string name = "";
                int numLossRecords = gadrasAPI.spectraFileGetInfo(fullPcfFileName).NumRecords;
                int numControlRecords = gadrasAPI.spectraFileGetInfo(controlPcfFileName).NumRecords;
                for (int recordIndex = 1; recordIndex < numLossRecords + 1; recordIndex++)
                {
```

```
                name = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).Title;

                if (name.Split('~')[0] == time && name.Split('~')[1] == kmp)
                {

                    string cal = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).EnergyCalibration.Order0Text;
                    string cal1 = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).EnergyCalibration.Order1Text;
                    string cal2 = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).EnergyCalibration.Order2Text;
                    string cal0 = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).EnergyCalibration.Order3Text;
                    int fds = gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).ChannelCount;

                    float fdsag = gadrasAPI.detectorGetEnergyForChannel(1000f, 4096, detectorEnergyCalibration);

                    for (int channel=0; channel < 4096; channel++) //need to make this flexible?
                    {
                        sw.WriteLine(gadrasAPI.detectorGetEnergyForChannel(channel + 1, 4096, detectorEnergyCalibration).ToString() + "~" + gadrasAPI.spectraFileLoadRecord(fullPcfFileName, recordIndex).GammaCounts[channel].ToString());
                    }
                }

            }

            sw.WriteLine("$"); //delimits between loss and control scenarios


            for (int recordIndex = 1; recordIndex < numControlRecords + 1; recordIndex++)
            {
                name = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).Title;

                if (name.Split('~')[0] == time && name.Split('~')[1] == kmp)
                {

                    string cal = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).EnergyCalibration.Order0Text;
                    string cal1 = gadrasAPI.spectraFileLoadRecord
```

```
(controlPcfFileName, recordIndex).EnergyCalibration.Order1Text;
                            string cal2 = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).EnergyCalibration.Order2Text;
                            string cal0 = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).EnergyCalibration.Order3Text;
                            int fds = gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).ChannelCount;

                            float fdsag = gadrasAPI.detectorGetEnergyForChannel(1000f, 4096, detectorEnergyCalibration);

                            for (int channel = 0; channel < 4096; channel++) // need to make this flexible?
                            {
                                sw.WriteLine(gadrasAPI.detectorGetEnergyForChannel(channel+1, 4096, detectorEnergyCalibration).ToString() + "~" + gadrasAPI.spectraFileLoadRecord(controlPcfFileName, recordIndex).GammaCounts[channel].ToString());
                            }
                        }

                    }


                    Process.Start(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\Gamma NDA Analysis" + "\\gammaChannels.xlsx");
                }

                sw.Close();



            }



        }
    }



}
```