The Dissertation Committee for Gregory George Thoreson
certifies that this is the approved version of the following dissertation:

A General Nuclear Smuggling Threat Scenario Analysis Platform

Committee:

Erich A. Schneider, Supervisor

Steven R. Biegalski

Dale E. Klein

David P. Morton

Laurie S. Waters

A General Nuclear Smuggling Threat Scenario Analysis Platform

by

Gregory George Thoreson, B.S.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2011

To Kelly

Acknowledgments

I would like to thank my parents for their support and encouragement throughout my entire life. Thanks Mom and Dad.

My advisor deserves my gratitude and respect for the endless hours we have spent over this research. It has been an honor to work with him in my graduate career.

When it seemed that I would never finish, my wife was always there to help organize my thoughts and motivate me, allowing me to see the light at the end of the tunnel. Thank you for your love and support, Kelly.

A General Nuclear Smuggling Threat Scenario Analysis Platform

Publication No. ____________________

Gregory George Thoreson, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Erich A. Schneider

A hypothetical smuggling of material suitable for a nuclear weapon is known as a threat scenario. There is a considerable effort by the U.S. government to reduce this threat by placing radiation detectors at key interdiction points around the world. These detectors provide deterrence and defense against smuggling attempts by scanning vehicles, ships, and pedestrians for threat objects. Formulating deployment strategies for these detectors within the global transportation network requires an understanding of the complex interactions between the attributes of a smuggler and the detection systems. These strategies are rooted in the continued development of novel detection systems and alarm algorithms. Radiation transport simulation provides a means for characterizing detection system response to threat scenarios. However, this task is computationally expensive with existing radiation transport codes. Furthermore, the degrees of freedom in smuggler and threat scenario

attributes create a large, constantly evolving problem space. Previous research has demonstrated that decomposing the scenario into independently simulated components using Green's functions can simulate photon detector signals with coarse energy resolution. This dissertation presents a general form of this approach, applicable to a wide range of threat scenarios through physics enhancements and numerical treatments for high energy resolution photon transport, neutron transport, and time dependent transport. While each Green's function implicitly captures the full transport phase-space within each component, these new methods ensure that this information is preserved between components. As a result, detector signals produced from full forward transport simulations can be replicated within 20% while requiring multiple orders of magnitude less computation time. This capability is presented as a general threat scenario simulation platform which can efficiently model a large problem space while preserving the full radiation transport phase-space.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Non-Proliferation

Non-proliferation is the prevention of the spread of nuclear material and technology, especially that which is necessary to produce nuclear weapons. The physical protection and accounting standards of such material are the primary defense mechanisms. As part of a layered global security strategy, the Department of Energy's National Nuclear Security Agency introduced the Second Line of Defense (SLD) program in 1998 which originally focused on placing radiation detection equipment at key border crossings, seaports, and airports in Russia and other former Soviet Union states. To date, the program has installed detectors at 221 sites in Russia [1]. It has since expanded beyond Russia, installing equipment at 94 sites in countries beyond the former Soviet Union, and with the Megaports Initiative, radiation detectors have been installed at 30 large volume seaports around the world. In 2006, the Department of Homeland Security (DHS) announced the Secure Freight Initiative, which works with SLD to screen cargo destined for the United States either domestically or internationally [2]. Under this combined effort, DHS now operates over 825 radiation detection systems at U.S. ports [3]. This concerted effort among agencies highlights the importance of combatting the smuggling of nuclear weapons or materials.

1.2 Motivation

In this context, a threat scenario is defined as a hypothesized smuggling of a nuclear weapon or the special nuclear material (SNM) required to construct such a weapon. This is further defined by the smuggler's attributes such as transportation methods, movement strategies, and attempts to defeat detection equipment. Modeling threat scenarios requires assumptions regarding the smugglers' knowledge. One one extreme, assuming smugglers are unaware of radiation and detectors is unrealistic. However, it is also highly improbable that smugglers have obtained detailed knowledge regarding how detectors may respond to their smuggling attempts, as even the interdictors may be unaware of such information. Modeling threat scenarios thus requires either a compromise between the uninformed and omniscient smuggler, or more preferably, the ability to design a network based on a range of smuggler attributes.

Smuggler attributes and behavior are constantly evolving in response to non-proliferation strategies. For example, if a smuggler is aware of the presence of an improved detector along his transportation path, he could employ various shielding techniques, based on his expertise and the level of network transparency, to make the SNM less visible to the radiation detectors. A static range of attributes would not capture this phenomenon. Thus, the ability to dynamically alter smuggler attributes based on continually updated defense strategies is also important for a well-rounded network design.

There are three major categories of detection systems currently used: passive, active, and imaging. Passive systems are the simplest of the three, consisting of one or more radiation detectors. They detect the natural radiation constantly emitted from SNM. Active detection utilizes an external source of radiation, such as bremsstrahlung from a linear accelerator, to bombard the SNM and produce secondary radiation which is detectable and indicative of its presence. Imaging systems are similar to active detection in that they require an external radiation source; however, instead of inducing secondary radiation, they rely on radiography or computed tomography to generate an image of the target. Passive systems are the most common due to their relative low cost and portability. However, active and imaging systems can usually detect smaller amounts or highly shielded SNM.

Alarm algorithms seek to differentiate natural background radiation from potential threats based on a complex detector signal. This interpretation produces detection probabilities (DP) and ultimately determines the detector's performance. Model data or observations from deployed detectors are used to develop algorithms by extracting additional information from the signal or by recognizing statistically significant patterns. A major challenge to many algorithms is false alarms resulting from naturally occurring radioactive material (NORM) such as fertilizer and bananas. Because detectors must scan a large volume of traffic, and secondary screening is costly, sensitivity to legitimate alarms is limited by forcing higher alarm thresholds to accommodate NORM. Conditioning algorithms to identify detector signal abnormalities indicative of

NORM reduces the false alarm probability (FAP). Similarly, by conditioning algorithms to search for characteristic SNM signatures based on a spanning set of threat scenarios, the DP is increased.

Detector signals can be a strong function of observable scenario parameters such as the detector stand-off distance, local environment, and vehicle type. If this information is utilized to produce an estimate of the signal, then the sensitivity of the alarm algorithm may be customized to the specific conditions for each interrogation. Applying this approach requires a tool capable of real-time scenario modeling or a large database of precomputed scenarios. Because of the enormous problem space, the latter solution is unattractive. Scanning real-time data such as cargo manifests and vehicle position provides algorithms with a baseline comparison for benign scenarios, increasing sensitivity to abnormalities such as SNM presence.

While developing novel radiation detection systems and alarm algorithms are crucial components to a defense strategy, it is also important to understand how to deploy such devices on a transportation network. Because non-proliferation programs operate on a finite budget, the cost of detection equipment may determine the quantity to deploy on the network. With the deployment of multiple detection systems on a transportation network, a smuggler has the option of multiple paths on this network. In contrast to the DP at a single detector, the chance of interdicting the smuggler somewhere on the network is known as the macroscopic detection probability (MDP). The DP, which is highly dependent on smuggler attributes, drives detector placement and can influence smuggler behavior on the network. Therefore, the DP and MDP are intimately related and modeling this connection is crucial for an accurate MDP estimate. However, determining the DP and MDP independently is computationally intensive with current techniques, making a direct coupling between the two techniques incompatible.

Modeling a spanning set of threat scenarios provides a basis on which robust network deployment strategies may be tested. Doing so in reasonable time requires an approach which is computationally efficient. In addition, a DP determination method which is both computationally efficient and can account for a wide range of scenarios is necessary for conditioning alarm algorithms and for a real-time scenario-customized algorithm.

1.3 Approach

The range of smuggler attributes and the sensitive nature of SNM presents a problem not easily studied on a purely experimental basis. However, with modern computers, studying the fundamental difference between SNM and benign material, radiation, can be accomplished at the computational level. Whether induced or passively emitted, radiation transport through matter has been studied extensively both computationally and experimentally. Multiple radiation transport codes exist to model this behavior. However, even with modern supercomputers this process can be time-consuming, making the study of multiple threat scenarios and detections systems difficult.

There are essentially two parts to the dilemma of modeling radiation transport in threat scenarios. First, the computational effort is considerable. Second, the uncertainty, quantity, and variety of smuggler attributes creates an enormous problem space which is difficult to define. In other words, there is a computational problem, and a combinatoric problem. One solution is to define a subset of the problem space as characteristic scenarios, or a combination of attributes which are representative of all threat scenarios, thus greatly reducing the problem space. Another approach expedites the radiation transport algorithms by simplification, such as reducing the problem to one-dimension, thus solving the computational problem. Each of these two paradigms solve one of the problems, which is sufficient for some studies, but not for network interdiction modeling, alarm algorithm research, or real-time scenario modeling. For these applications, both problems need to be addressed. Preserving as many smuggler attributes as possible is critical in maintaining the complex interactions between threat scenarios and detection systems. It is equally important to preserve the physics of the radiation transport to accurately predict DPs.

To rapidly and accurately model threat scenarios, the approach of decomposition is presented. Decomposition involves separating the scenario into components based on logical or physical boundaries. For example, an SNM source with shielding may be decomposed into the SNM alone, and the shielding separately. This is accomplished through the use of Green's functions, and offers many benefits, the strongest of which is the mitigation of the combinatoric problem. For this argument, assume estimates of the number of threat scenario attributes and their perturbations are available; together these define the problem space. Furthermore, for simplicity, assume that each attribute requires the same number of perturbations. Letting n be the number of attributes and m the number of perturbations, then using a brute-force approach, sampling the entire problem space would require n^m simulations. With decomposition, each attribute is treated independently and therefore only $n \times m$ simulations are required, but the computational effort for each decomposed problem is significantly higher than under the brute-force approach. Therefore, for a small number of attributes and perturbations, direct simulation methods are preferable. However, as the number of attributes and perturbations grow, the decomposition method requires geometrically fewer simulations than the direct method, granting the ability the simulate a spanning set of threat scenarios within reasonable computation time. This ability makes the novel application of Green's functions ideal for dynamic problem spaces where attributes and perturbations may be added or removed as more information becomes available.

With decomposition, components may be added or modified as detector systems or alarm algorithms evolve. Previous work has proven the feasibility of this method for photons passively emitted from SNM using coarse energy resolution Green's functions, which is suitable for many detector systems currently deployed [4]. With newer systems, equipped with higher energy resolution detectors, it is necessary to increase the energy resolution of these functions. This dissertation presents a method to overcome this burden and implement photon energy resolution on the order of 1 keV. In addition to higher energy resolution detectors, neutron detectors are installed to detect fission neutrons from SNM. While the method of decomposition is still valid, neutron transport is fundamentally different from photon transport, thus requiring a new approach to account for albedo effects. Also, neutrons exist within the scenario for a measurable length of time, a phenomenon of which some alarm algorithms take advantage. New methods to use decomposition and Green's functions with time-dependent neutron transport are presented. Methods are also developed to maintain the full phase-space of the radiation transport at the component interfaces, and to parameterize each submodel with respect to details such as geometric dimensions and material composition.

The components of high energy resolution, neutron transport, and time dependence are crucial for a comprehensive threat scenario analysis tool. To this end, this research presents the theory and methods which satisfy these requirements, as well as an implementation in the form a usable software package. Together with previous work, this software introduces the novel ability to rapidly analyze a spanning set of threat scenarios, provides a new platform for designing and testing detection equipment and alarm algorithms, allows real-time modeling of threat scenarios, and presents a tool to the threat reduction community to develop robust detection networks for national security.

Chapter 2

Literature Review

2.1 Detection Systems

Radiation portal monitors (RPM) are the most common radiation detection systems in place today. They are checkpoint gateways for vehicles, cargoes, or pedestrians, and are equipped with radiation detectors, computers to analyze the detector data, and usually staff to respond to alarms. An example of a pedestrian and vehicle RPM is shown in Figure 2.1.

(a) Pedestrian RPM [5]

(b) Vehicle RPM [6]

Figure 2.1: Radiation Portal Monitor Examples

The performance of an RPM is usually defined by some minimal detectable level of activity of nuclear material. Because of the stochastic nature of radioactive decay, there is always some error associated with the detector signal, quantified with counting statistics. From a detector signal, one may apply an alarm algorithm to determine the detection probability (DP). By placing the DP within a confidence interval (e.g. 95% DP), a lower limit of detection may be defined for some benchmark cases. Thus, minimal detectable activity and DP are interchangeable performance tests for RPMs. As previously mentioned, the three major types of detection systems used today are passive, active, and imaging. This research focuses on passive and active systems.

2.2 Passive Detection

Passive systems attempt to detect the constant emission of photons and neutrons from SNM due to gamma decay and spontaneous fission. A common photon detector is polyvinyltoluene (PVT), a plastic scintillating material which is formed into large flat panels. The dimensions of PVT panels vary depending on the application and manufacturer, but are approximately 1.5 m long, 0.5 m wide, and a 3 cm deep [7] [8]. Because of their low cost compared to other photon detectors [9] and large surface area, these panels are ideal for creating detector arrays as each subtends a large solid angle. However, the energy resolution of PVT is very poor, ranging between 15% and 50% at 20 keV [10] [7] [11], leaving the energy spectrum devoid of photopeaks. Therefore, these detectors are used primarily for gross count alarm algorithms which integrate over the entire spectrum.

Gross count algorithms struggle to discriminate NORM cargos from

SNM and are therefore prone to high false alarm rates. To address this issue, the DHS introduced the Advanced Spectroscopic Portal (ASP) requirements in 2004 [12] which requires detectors capable of gamma spectroscopy and NORM discrimination. Gamma spectroscopy can identify gamma decay energies unique to SNM, but requires a higher energy resolution detector than PVT. Although there are many different detectors that meet this need, two commonly addressed detector materials are sodium iodide (NaI) and high purity germanium (HPGe). Their energy resolutions are approximately 7% at 662 keV and 0.2% at 1 MeV, respectively [13]. While HPGe can resolve 1 keV differences in gamma photopeaks, the maximum size available is 9 cm in diameter and requires cooling from a large liquid nitrogen dewar which must be replaced weekly [14]. NaI does not require external cooling and can be grown in a variety of crystal sizes, anywhere from a few centimeters in diameter and length up to 10 cm in diameter and 1 m in length [14] [10] [15], but its energy resolution is poorer than HPGe. While there are a variety of other detector materials under development, such as cadmium zinc telluride (CZT), PVT, NaI, and HPGe detectors are representative of the range of energy resolutions available.

To subtend a solid angle comparable to PVT panels, NaI and HPGe detectors may be placed in an array, usually enclosed in a flat structure resembling a PVT panel. Steel or lead shielding reduces the gamma background from terrestrial radionuclides. In some designs, collimator plates are placed on the sides of the detector to provide additional spatial and temporal resolution

[16]. The orientation, placement, and number of panels varies depending on the manufacturer.

Unlike photon detectors, neutron detectors have limited energy resolution capabilities. Neutrons are detected indirectly through nuclear interactions, such as a capture reaction in a tube filled with a boron-fluoride or helium-3 gas. During a capture reaction, recoiling nuclei ionize the gas in the tube, which produces a count in the detector. Because capture reactions are highly probable at thermal energies, these tubes are usually surrounded by a hydrogenous thermalization medium such as plastic. Helium-3 is the most prevalent gas used today [14] [9], and is the primary neutron detector considered in this research. Tubes are manufactured in a variety of sizes, but can be up to a 7 cm in diameter and 100 cm in length. Like photon detectors, they may be placed in an array to increase the solid angle and absolute detection efficiency.

Solid-state and glass fiber neutron detectors are less common, but are under consideration for use in RPMs. Solid state detectors are constructed by layering semiconductors with neutron absorbers such as boron-10. The semiconductor detects the directly ionizing secondary radiation emitted by the absorber, such as the alpha particle from the $^{10}B(n,\alpha)^{7}Li$ reaction. Traditional planar geometry designs have limited intrinsic efficiency as the thickness of the absorber determines both the alpha penetration depth and neutron absorption probability, which are competing effects. As manufacturing processes improve, three-dimensional fins or pins of semiconductors layered with absorbers can increase this efficiency [17]. Glass scintillating fibers doped with lithium-6 show considerable promise as a replacement for helium-3 tubes [18]. These fibers may be made into a variety of geometries and in large sizes, but struggle to discriminate gamma rays from neutrons as well as helium-3 tubes.

2.3 Active Detection

Active detection utilizes an external source of radiation to induce secondary radiation within SNM. This secondary radiation is characteristic of SNM, and is detected by the passive detection technology outlined in the previous section. Two commonly studied interrogation particles are photons and neutrons.

Interrogation photons are usually produced by accelerating electrons and directing them onto a high-Z target to produce bremsstrahlung or by accelerating protons into nuclei to produce discrete gamma energies [19] [20]. Photon energies range from 6 MeV to 15 MeV [20]. These high energies are required as the photofission reaction energy threshold is about 6 MeV; photofission becomes most probable around 14 MeV [21]. In addition, high energy photons are able to penetrate through shielding material. However, they also produce neutrons in common materials from direct photonuclear interactions, producing significant noise in the detector signal. Direct photoneutrons and prompt neutrons from photofission are emitted within the same time scale. Delayed neutrons from fission are emitted for a measurable length of time after the fission event. Taking advantage of this difference, many active interrogation schemes use a pulsed source, and detect a time-dependent signal following the pulse to search for delayed neutrons. Typical collection times are on the order of hundreds of microseconds to a few milliseconds [22]. Some systems also employ photon detectors to collect fission gammas [19].

Neutron beams are usually produced by accelerating deuterons into tritium (DT), deuterons into deuterons (DD), or protons into lithium-7, producing 14 MeV, 1-8 MeV, and 60 keV neutrons, respectively [19] [20] [23]. The concept is similar to photon beam sources in that the neutron beam is pulsed, inducing fissions in the SNM, and a time-dependent neutron signal is collected after the pulse. Thermalized neutrons from the beam exist for a length of time comparable to the delayed neutrons produced from the SNM, giving rise to an algorithm family known as Differential Die Away Analysis (DDAA). After a neutron pulse, a neutron detector collects a time-dependent signal which has an exponentially decaying or die-away behavior. Without SNM present the detector observes a die-away time as source neutrons are thermalized. If a fissionable source such as SNM is present, additional prompt neutrons are created as well as delayed neutrons, effectively lengthening the die-away time. A comparison of the two cases reveals a differential neutron count rate profile in time.

2.4 Existing Threat Scenario Simulation Tools

The core of threat scenario modeling is radiation transport simulation. Recently, a few software packages designed specifically to model radiation transport in threat scenarios have emerged. This document briefly reviews the most well known radiation transport codes and a few threat reduction codes.

2.5 Radiation Transport Codes

The Boltzmann transport equation governs the transport of neutral particles such as photons and neutrons through matter. Implementations of its numerical solution fall into two broad categories: stochastic and deterministic. Stochastic, also known as Monte Carlo, radiation transport codes may solve the integral Boltzmann equation by direct Monte Carlo integration techniques, or more commonly by analog transport. Analog transport randomly samples probability density functions (PDFs) taken from the transport process, such as cross-sections, to generate random walks of particles. The particles' individual contributions to some measurement, such as a flux or current, are tracked to produce estimates of the mean and variance.

Monte Carlo N-Particle (MCNP) and Monte Carlo N-Particle eXtended (MCNPX) are two stochastic radiation transport packages developed by Los Alamos National Laboratory [24] [25]. They allow the use of general three-dimensional geometries created by the boolean combination of quadric surfaces, general sources specified by PDFs, and the estimation of a variety of quantities such as the flux, energy deposition, and detector pulse height spectra. MCNP transports photons, neutrons, and electrons while MCNPX additionally transports over 40 different particles including protons, muons, and heavy ions. These codes have been applied to a wide range of applications including reactor physics, medical physics and threat reduction. There are many other Monte Carlo based transport codes such as COG and Geant [26] [27]. Most are capable of general geometries, sources, and estimation of a variety of quantities. The most significant differences are available particle types, cross-section data, physics models, and variance reduction techniques. Most Monte Carlo codes are applicable to threat scenarios as studies are usually limited to photons and neutrons. However, experimental detection techniques such as proton interrogation require additional capabilities.

Deterministic transport codes solve the Boltzmann equation by discretizing the phase-space and either approximating derivatives using finite differencing schemes or solving the weak form of the equation by integrating over finite elements. This produces a system of equations which is solvable exactly and approximates the Boltzmann equation. Codes that accomplish this task include Atilla, DANTSYS, PARTISN, and Newt [28] [29] [30] [31]. Deterministic codes are typically faster than stochastic codes, especially for problems with highly attenuating materials. However, generating energy group cross-sections and establishing mesh convergence requires additional computation time. These codes are usually limited to transporting neutral particles, and cannot produce pulse height spectra in detectors. Hence, the particle flux resulting from a deterministic analysis is often coupled to a Monte Carlo code to produce detector spectra.

While either radiation transport paradigm may be utilized to model threat scenarios, the large three-dimensional geometries encountered make Monte Carlo methods more attractive. Furthermore, the extensive capabilities, widespread use in literature, and documentation of MCNP/X make it an ideal software package. However, like any transport code, MCNP/X does not overcome the computational burden associated with the large threat scenario problem space.

2.6 Threat Reduction Codes

Threat reduction software customizes radiation transport analysis for threat scenario modeling. For example, SWORD allows users to graphically construct geometries and utilize pre-built geometries and sources to generate scenarios for MCNPX and Geant [32]. It produces plots of detector spectra and particle tracks for debugging. A similar but more specialized interface is TR-X, an unpublished code developed at Los Alamos National Laboratory. TR-X imports geometry templates in a graphical interface to build threat scenarios. It also manages the simulation of the scenario through MCNPX, and produces detector plots and detection probabilities. These software packages focus on modeling a single threat scenario to a high degree of fidelity, and like the radiation transport codes they execute, do not solve the computational problem.

RADSAT is a collaborative effort between Sandia National Laboratory and Pacific Northwest National Laboratory to utilize Atilla, a deterministic code, for the bulk of the problem geometry and couple the results to a Monte

Carlo code such as MCNP to produce a detector response [33]. Features in Atilla such as unstructured meshing, first-scattered-distributed source, and last-collided flux, greatly reduce the computation time. In the benchmark cases published, RADSAT produced results comparable to a full MCNP calculation within one to two orders less time, however there is some discrepancy in the detector spectra as Atilla uses a multi-group calculation. While RADSAT reduces the computation time compared to a MCNP calculation, the time required remains substantial, and it does not address the large problem space at hand.

A simplified approach to threat scenarios is encapsulated in the LOST software package developed by Pacific Northwest National Laboratory [34]. LOST is designed to simulate a search for radioactive material in a nuisance source environment. Large volumes of NORM such as terrestrial radiation, and NORM cargos are reduced to surface sources in pre-generated calculations. These surface sources are then ray-traced to a searching instrument, such as a detector. Compared to Geant, the ray-tracing in LOST reduces computation time by an order of magnitude or more. However, the benchmark calculation published required approximately an hour of computation time. There has also been some work to generate surface sources based on single radionuclides, allowing a superposition approach to be implemented [35]. Using these surface sources, pre-generated detector response functions, and LOST, a statistical sampling of various NORM cargos is used to characterize false alarms at detectors. This methodology addresses the computational problem by introducing ray-tracing, and the combinatoric problem by pre-generating surface sources and detector responses.

GADRAS is a one-dimensional transport code developed at Sandia National Laboratory. This code utilizes deterministic codes such as ONEDANT (now incorporated in DANTSYS) in combination with ray-tracing to model one-dimensional problems and offers an extensive radiation detector response function library [36]. From a graphical interface users may construct geometries and solve the problem quickly. However, the geometry must be well represented in one-dimension. The computation time for GADRAS is negligible, but because the consequences of simplifying complex three-dimensional threat scenarios to one-dimension is not well understood, its ability to model the entire problem space is limited.

An exhaustive comparison of the mentioned software packages is beyond the scope of this research. However, it is sufficient to note that while all can address threat scenario modeling to some degree, they fall short of addressing both the computational burden and large problem space associated with the scenarios. Decomposition and parameterization of threat scenarios solves both problems in a novel fashion.

2.7 Previous Model Development

The success of decomposition hinges on the ability to treat the radiation transport in each attribute independently. This does not require that the attributes actually be independent, which can be hindered by phenomena such as albedo effects, but just that treatments are available to account for these effects. One method to accomplish decomposition is through the use of Green's functions which are best described within the context of radiation transport theory. Let $\psi(\vec{r}, E, \hat{\Omega})$ be the angular particle flux at $d^3\vec{r}$ about $\vec{r}$ with kinetic energy between E and $E + dE$ traveling in direction $d\hat{\Omega}$ about $\hat{\Omega}$ in units of $[\mathrm{cm}^{-2} \cdot \mathrm{eV}^{-1} \cdot \mathrm{str}^{-1} \cdot \mathrm{s}^{-1}]$. For brevity, let λ represent the phase-space of this flux (i.e. $\psi(\vec{r}, E, \hat{\Omega}) = \psi(\lambda)$, $d\lambda = d^3\vec{r}\ dE\ d\hat{\Omega}$). The Green's function G satisfies the differential equation

$$HG(\lambda; \lambda') = \delta(\lambda - \lambda'), \tag{2.1}$$

where δ is the Dirac delta function, and H is the transport operator to the time-independent Boltzmann equation in a non-multiplying medium,

$$H = \left[\hat{\Omega} \cdot \vec{\nabla} + \sigma_t(\vec{r}, E)\right] - \int dE' \int d\Omega' \sigma_s(\vec{r}, E \leftarrow E', \hat{\Omega}' \cdot \hat{\Omega}) \ , \tag{2.2}$$

where the del operator is $\vec{\nabla}$ (unitless), σ_t is the macroscopic total attenuation cross section in $[\mathrm{cm}^{-1}]$, and σ_s is the macroscopic differential scattering cross section in $[\mathrm{cm}^{-1} \cdot \mathrm{eV}^{-1} \cdot \mathrm{str}^{-1}]$. Given a fixed source q $[\mathrm{cm}^{-3} \cdot \mathrm{eV}^{-1} \cdot \mathrm{str}^{-1} \cdot \mathrm{s}^{-1}]$, the particle flux may be solved for using the Green's function,

$$\psi(\lambda) = \int d\lambda' G(\lambda; \lambda') q(\lambda'). \tag{2.3}$$

Because the flux may be determined for any source using a Green's function, this method may be applied to scenario decomposition. Consider a SNM mass surrounded by shielding. Disregarding the shielding, the flux within the SNM is

$$\psi^{\text{snm}}(\lambda) = \int d\lambda' G^{\text{snm}}(\lambda;\lambda') q(\lambda'), \tag{2.4}$$

where G^{snm} is the Green's function for the SNM in [$\text{cm}^{-2} \cdot \text{eV}^{-1} \cdot \text{str}^{-1} \cdot \text{s}^{-1}$], for some source q [$\text{cm}^{-3} \cdot \text{eV}^{-1} \cdot \text{str}^{-1} \cdot \text{s}^{-1}$]. By generating another Green's function for the shielding, the flux within the shielding is computed by integrating the Green's function and SNM flux over the surface of the SNM sphere,

$$\psi^{\text{shld}}(\lambda) = \int_{\Gamma_{\text{snm}}} d^3\vec{r}' \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \int_0^\infty dt'\ G^{\text{shld}}(\lambda;\lambda')\,(1/v(E'))\,\psi^{\text{snm}}(\lambda') \tag{2.5}$$

where $v(E')$ is the velocity of the particles at energy E', G^{shld} is the Green's function for the shield in [$\text{cm}^{-2} \cdot \text{eV}^{-1} \cdot \text{str}^{-1} \cdot \text{s}^{-1}$], and Γ_{snm} is the physical boundary of the SNM and the interface between the SNM and shielding. Combining Eqs. 2.4 and 2.5 results in

$$\begin{aligned} \psi^{\text{shld}}(\lambda) &= \int_{\Gamma_{\text{snm}}} d^3\vec{r}' \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \int_0^\infty dt'\ G^{\text{shld}}(\lambda;\lambda')\,(1/v(E')) \\ &\quad \int d\lambda_0 G^{\text{snm}}(\lambda';\lambda_0) q(\lambda_0), \end{aligned} \tag{2.6}$$

thus computing the flux within the shielding for any source q within the SNM. This forward approach applies Green's functions sequentially in the order that a particle may experience in its lifetime. This ignores albedo effects, thus assuming the radiation transport in the shielding and SNM are independent. This assumption is relaxed in Chapter 3, which discusses its impact on neutrons and treatments for them. This simple example may be extended to include many different scenario attributes by applying additional Green's functions, each encapsulating the physics of the radiation transport in a reusable form.

The previous example utilized Green's functions which compute the flux at any phase-space within each attribute with respect to any source. If the physical interfaces between attributes are well-defined, and the flux is uniform over the interface, then the fluxes and Green's functions may be integrated over spatial variables. Furthermore, often the angular distribution at the interfaces follows a cosine or cosine-squared distribution, and so the explicit angular dependence may be integrated out of the Green's functions. Finally, if there is no time dependence, by integrating the fluxes and Green's functions over spatial, angular, and temporal variables, they are reduced to spectra and energy transformations, respectively. In addition, using energy groups instead of a continuous energy treatment, the Green's functions may be re-cast as matrices $G \to \mathbf{R} \in \mathbb{R}^{M \times N}$, and Eq. 2.6 transforms into a series of matrix multiplications,

$$\mathbf{q}' = \mathbf{R}^{\text{shld}}\, \mathbf{R}^{\text{snm}}\, \mathbf{q}, \tag{2.7}$$

where $\mathbf{q} \in \mathbb{R}^{N \times 1}$ and $\mathbf{q}' \in \mathbb{R}^{M \times 1}$ are column vectors representing the energy spectrum within the SNM and at the surface of the shield, respectively. Because Eq. 2.7 is just a series of matrix multiplications, the computation time required is negligible compared to the radiation transport simulation.

In many situations, it is desirable to continuously vary a threat scenario attribute. However, even with decomposition, the Green's functions for each attribute is unique to the materials and geometry used in the simulation. As an example, a shielding Green's function is valid for a specific shielding material and geometry, but in modeling many different threat scenarios the shielding thickness may be variable. By applying interpolation or perturbation schemes to an appropriate set of shielding geometries, a continuous sampling of thicknesses is achieved. Techniques such as calculating the solid angle subtended by a detector, for example, may also be applied to these Green's functions. Here, the manipulation of a pre-generated set of Green's functions is referred to as parameterization, and is a valuable tool in reducing the number of Green's functions required to span the problem space.

The method of decomposition and parameterization was applied to the land-based threat scenario in which the smuggler attempts to conceal shielded SNM in a truck-trailer at a RPM. The radiation transport was simulated for passively-emitted photons only. The threat scenario is decomposed into four major attributes: SNM, shielding, surrounding cargo, and detector. Additionally, terrestrial background radiation is considered as a separate attribute. The radiation transport code MCNPX, a Monte Carlo based code, is used to compute the Green's functions [25]. The following sections describe the radiation transport simulations done for each attribute, the assumptions that went into the models, and how the attributes interact with each other.

2.7.1 Special Nuclear Material

The isotopic composition and mass of SNM may vary greatly based on the source of the material and degree of enrichment. Thus, parameterizing with respect to isotopes and mass are important features. Another important feature of SNM is the geometry. For this demonstration, a simple spherical shape is assumed.

Parameterizing with respect to isotopics is accomplished with superposition. Because the photon cross sections of isotopes are identical, the radiation transport through a sphere is unaffected by which isotopes are present. By separately simulating spheres composed of each individual isotope's emission rate and spectrum, and superimposing the results, the SNM source term is pre-computed as

$$\mathbf{q}_i^{\text{snm}} = \mathbf{R}^{\text{snm}}\,\mathbf{q}_i, \tag{2.8}$$

using the notation from Eq. 2.7 and assigning a subscript i representing each isotope. Thus, by using unique isotope signatures in the simulation, there is no explicit Green's function for the SNM sphere, but instead a new source term which represents the gamma emission rate and spectrum leaving the sphere. The total SNM source term is a weighted sum of the individual isotopic results,

$$\mathbf{q}^{\text{snm}} = \sum_i w_i \mathbf{q}_i^{\text{snm}}, \tag{2.9}$$

where w_i is the weight fraction of the i^{th} isotope.

If the gamma spectrum exiting a sphere of weapons-grade plutonium (WGPu), aged 20 years, is computed for masses of 1 g, 10 g, 100 g, 1 kg, and 10 kg, there is an overall mass effect of increasing photon emission rate from the sphere. The 10 kg mass is utilized only to demonstrate a point; the criticality concerns with such a large mass of WGPu would obviously make the existence of such a configuration unlikely. If the exiting spectra are normalized by the respective total photon emission rate, this yields the probability, p that a photon born within the sphere escapes, which is entirely determined by the self-shielding effect. However, if these probabilities are scaled by the respective volume to surface area ratios, the difference is resolved for masses greater than 1 kg. This is due to a saturation layer achieved in masses greater than 1 kg; that is, beyond 1 kg, photons born within an outer spherical shell of constant thickness which dominate the signal.

This scaled probability spectrum may be unscaled for any mass greater than 1 kg by multiplying by the surface area to volume ratio, and also by the total gamma emission rate. If this scaled probability is computed for each individual isotope, by superposition and scaling it becomes possible to compute the exiting spectrum for any combination of isotopes for any mass greater than 1 kg.

2.7.2 Shielding

The shielding configuration is an important decision variable made by the smuggler. It determines the transparency of SNM to detectors. For a demonstration of this method with passive photons, lead is chosen as the sole shielding material. For simplicity it is assumed that the shielding geometry is a spherical shell completely surrounding the SNM.

Interpolation is employed to parameterize the shield with respect to radial thickness. To avoid multiple simulations, a shield of thickness 20 cm is partitioned into 100 layers with the photon current tallied at each layer. Furthermore, a particle accounting method called surface flagging in MCNPX was utilized to flag photons which have reached a certain radial distance through the shield. The current tallies may then subtract particles which have traveled beyond their radial distance, creating the overall effect of replicating a vacuum boundary condition at each layer.

2.7.3 Vehicle

Enumerating all types of cargo in truck-trailers is a daunting task. When considering all combinations and arrangements of these, the task is nearly impossible. Furthermore, the benefit in simulating a large array of highly-detailed cargo arrangements is unclear. Instead, three homogeneous cargoes are chosen as representative of all cargo types, a low-Z, mid-Z, and high-Z, where Z represents the average atomic number of the cargo. An example of a low-Z cargo is something hydrogenous such as a paper, where as a high-Z cargo may be largely composed of iron such as machine parts. As a surrogate for different cargo configurations, a solid angle streaming fraction is introduced, which is discussed later.

Vehicles are not stationary at RPMs. They slowly drive through at approximately five miles-per-hour. Thus, the cargo attribute is slightly different because of the time-dependent relationship between the detector and truck. Furthermore, the smuggler may place the SNM at one or multiple variable locations within the cargo. Instead of directly simulating the time-dependent truck positions and all possible SNM locations, the adjoint method is employed. The adjoint problem is solved via the adjoint transport operator,

$$H^\dagger = \left[-\hat{\Omega}\cdot\vec{\nabla} + \sigma_t(\vec{r},E)\right] - \int dE' \int d\Omega' \sigma_s(\vec{r},E\to E',\hat{\Omega}\cdot\hat{\Omega}'), \tag{2.10}$$

which shares the same notation seen in Eq. 2.2. The adjoint solution $\psi^\dagger$ satisfies

$$H^\dagger\psi^\dagger = q^\dagger, \tag{2.11}$$

where $q^\dagger$ is the adjoint source which in this case is the flux at the detector location. A physical interpretation of the adjoint flux is an importance function, or how likely particles at a certain phase-space in the problem are to contribute to the detector. If the adjoint solution is not available, but a relative importance function is, this may be used to compute the flux as a function of different source positions, without re-solving the transport problem for each source. Given a Green's function for a forward problem with a source fixed at $\vec{r}_0$, $G(E;E_0,\vec{r}_0)$, it may be weighted by the relative importance map to yield a cargo Green's function dependent on source position $\vec{r}$,

$$G^{\text{car}}(E;E_0,\vec{r}) = G(E;E_0,\vec{r}_0)\left(\frac{\psi^\dagger(E_0,\vec{r})}{\psi^\dagger(E_0,\vec{r}_0)}\right), \tag{2.12}$$

where $\psi^\dagger(E_0,\vec{r}_1)/\psi^\dagger(E_0,\vec{r}_0)$ is the relative importance function. The utility in using the relative adjoint arises when a code such as MCNPX does not offer any direct adjoint solution. Instead, weight windows may serve as an estimate of the relative importance. Furthermore, the truck-trailer is made infinite in the direction of the RPM lane. This removes the need for multiple detector locations to simulate different positions in time as the SNM may just be moved incrementally through the semi-infinite truck-trailer.

To simulate streaming pathways, the cargo Green's function is modified by a fractional solid angle. Let Ω be the solid angle subtended by the detector and f_Ω be the fraction of the solid angle which is unimpeded by any material, then the cargo Green's function becomes

$$G^{\mathrm{car}}(E; E_0, \vec{r}, f_\Omega) \leftarrow f_\Omega \frac{\Omega}{4\pi} + (1 - f_\Omega) G^{\mathrm{car}}(E; E_0, \vec{r}). \tag{2.13}$$

2.7.4 Detector

The most prevalent type of detector in current RPMs is PVT; although the geometry of PVT detectors vary between manufacturers. To avoid generating a Green's function for every detector size of interest, multiple one-dimensional Green's functions for the detector material are averaged. The averaging is based on rays drawn from the source point through the detector volume. The rays create Q chord lengths of length t_q, which are used to choose which one-dimensional thicknesses to average. The detector Green's function is given by the average of these one-dimensional chords,

$$G^{\mathrm{det}}(E; E_0) = \frac{1}{Q} \sum_{q \in Q} G(E; E_0, t_q). \tag{2.14}$$

2.7.5 Background Radiation

If the venue is assumed to be a road on which the RPM is installed, then the primary source of photons will be from the soil and the concrete. However, the concentration of these radionuclides can vary greatly based on geographic location and different construction materials. Therefore, having the ability to alter these concentrations is crucial. By calculating the photon flux at the ground surface from the uranium, thorium, and potassium separately in the soil and concrete separately, the results from each may be weighted and superimposed for any location or building material.

A cylinder composed of soil that is 5 m tall with 10 m radius and a 30 cm top-layer of concrete is used as the ground source. The fluence at the ground surface is tallied yielding $\phi_{k,c}/\phi_{k,s}$, $\phi_{u,c}/\phi_{u,s}$ and $\phi_{th,c}/\phi_{th,s}$, the fluence in [cm^{-2}] from each radioactive source in the concrete/soil assuming unit activity in the volume. Let $q_{k,c}/q_{k,s}$, $q_{u,c}/q_{u,s}$, and $q_{th,c}/q_{th,s}$ be the specific activity of the radionuclides in the concrete/soil in [$\gamma{\cdot}\mathrm{s}^{-1}{\cdot}\mathrm{g}^{-1}$]. Given the mass of the concrete and soil as M_c and M_s in [g], the total flux at the surface, ϕ is

$$\phi = M_c(q_{k,c}\phi_{k,c} + q_{u,c}\phi_{u,c} + q_{th,c}\phi_{th,c}) + M_s(q_{k,s}\phi_{k,s} + q_{u,s}\phi_{u,s} + q_{th,s}\phi_{th,s}) \tag{2.15}$$

This flux is used as a disk source; and, in conjunction with the same detector response described in the previous section, the disk source is used to generate the background for the detector.

As vehicles drive through the RPM, they partially shield the detectors from the terrestrial radiation, thus reducing the detector signal. This effect is known as baseline suppression. Its effect on detector performance is discussed further in Section 2.7.7. Instead of directly simulating this phenomenon, it is assumed that all truck-trailers suppress the background by the same profile outlined in reference [37], in which actual vehicle baseline suppression data is averaged.

2.7.6 Integration of Submodels

Although all of the attributes described here have a host of variables with which they are parameterized, after manipulation the Green's functions may be used as transformation matrices. Using the notation for response matrices outlined previously, for a given time interval the signal at a detector is expected to be

$$\mathbf{C} = \mathbf{R}^{\mathrm{det}}\,\mathbf{R}^{\mathrm{car}}\,\mathbf{R}^{\mathrm{shld}}\,\mathbf{q}^{\mathrm{snm}}, \tag{2.16}$$

where $\mathbf{C}$ is a column vector representing the detector spectrum in [counts]. If multiple SNM sources are present, their contributions to the detector signal may be summed. This operation is repeated for each time interval to produce time and energy dependent spectra for each detector present.

2.7.7 Alarm Algorithms

The detector signal is interpreted into a detection probability using alarm algorithms. Considering photons, for which terrestrial radionuclides provide a constant background, algorithms compare the signal at the detector to the expected background. This is the expected and not the actual background as detectors cannot decompose the signal or distinguish the source of the photons during a vehicle scan. Employing multiple detectors, Compton cameras, or energy discrimination can provide partial discrimination.

The simplest test for the presence of SNM is a comparison of the total number of counts collected in a given time interval to the expected background in the same length of time. This is known as a gross count (GC) or K-sigma test. Because there is statistical and systematic fluctuations in the detector signal, the background and source signal distributions always overlap to some degree. The acceptable false alarm probability (FAP) defines a threshold based on this overlap. This threshold, t, can be put in terms of the number, K, of standard deviations, σ_b, from the expected value of the background, b,

$$t = b + K\sigma_b. \tag{2.17}$$

In actual operations, if the counts at the detector during a scan exceed this threshold, an alarm is activated. The probability that it exceeds the threshold is calculated by integrating the normal distributions, taking the form of an error function. Let s be the signal mean and σ_s the standard deviation in the signal, then the detection probability (DP) is given by

$$p = \frac{1}{2} - \frac{1}{2}\text{erf}\left(\frac{t-s}{\sqrt{2}\sigma_s}\right). \tag{2.18}$$

There are variations to this methodology, such as energy windowing, which employs ratios of different energy segments in the detector spectrum instead of using the entire spectrum. These segments or windows may be very coarse, with just a few windows covering the entire spectrum, or very fine which is useful for gamma spectroscopy.

2.7.8 Implementation

The data for the attributes and their parameterizations are implemented in the C++ code XPASS (eXpedited Parametric Analysis of Smuggling Scenarios), which uses a coarse fourteen energy group structure ranging from 1 keV to 3.2 MeV. This program optimizes the use of response functions such that only the time-dependent ones are parameterized multiple times. In addition, a gross-count and energy-window alarm algorithm are applied to the detector spectra to produce detection probabilities. A typical run time for this program is on the order of one second, depending on the number of sources and detectors.

The implementation of XPASS demonstrates the viability of the theory and method outlined here. Despite a coarse energy group structure, the results from XPASS compared well to a high-fidelity benchmark study which used full-forward calculations in MCNP in combination with empirical detector response functions. However, this demonstration focused on photons alone, and better detector technologies are implemented into RPMs, the abilities of this tool is extended to meet these needs.

Chapter 3

Methodology

The implementation of parameterization and decomposition demonstrated the ability of the method to quickly analyze threat scenarios for passive photon detection. However, the fourteen energy group structure for photons limits the application to photon detector with poor energy resolution. With the deployment and testing of more advanced detection systems, a high resolution energy group structure is required to accurately model their capabilities. Neutron detectors are deployed in conjunction with advanced photon detectors to increase sensitivity to SNM. Neutron Green's functions are required to model the transport through each submodel. In addition, the neutron Green's functions must have time dependent capabilities. This chapter describes the methods to provide these capabilities.

Many radiation transport software packages are capable of providing transport data for response functions. Much of this theory is based on the assumption that the Monte Carlo radiation transport package MCNPX [25] is available. However, if another radiation software package is capable of providing the same data required by these methods, it may serve an equivalent role.

3.1 High Energy Resolution for Photons

While PVT detectors are prevalent in deployed RPMs, higher energy resolution detectors are becoming more common. Because PVT resolution is as low as 50% at 20 keV [11], fourteen energy groups over the range 1 keV to 3.2 MeV is sufficient to capture the gradients in PVT detector spectra. High purity germanium (HPGe) detectors have photopeak FWHM values ranging from 800 eV at 122 keV to 2.3 keV at 1.33 MeV [13]. Thus, modeling such detectors requires energy resolution on the order of 1 keV. To allow the Green's functions to be applicable to a wide range of detection technologies, including high energy active interrogation, the range of photon energies is 1 keV to 100 MeV. However, discrete photon energies above a few MeV are rare. For example, 94% of the ten most probable gamma decay energies for all radionuclides in the Evaluated Nuclear Structure Data File (ENSDF) lie below 2 MeV, and 98% below 3 MeV [38]. Although nuclear resonance fluorescense (NRF) active detection technologies produce gamma rays typically in the range 3 MeV to 10 MeV [39], these discrete energies are not as closely spaced as gamma decay energies.

To avoid a 1×10^5 energy group structure (100 MeV $\times$ 1000 bins/MeV), the 1 keV bin width is progressively widened with increasing energy. This dynamic resolution is summarized in Table 3.1. This scheme uses very fine energy bins in the low energy range, which is useful for identifying gamma decay lines. As the energy increases, discrete photon energies become uncommon above 3 MeV and very rare beyond 10 MeV; thus, a progressively coarser resolution is used for higher energies up to 100 MeV. This scheme results in 8831 energy bins over the entire range. The full listing of these energy bins are in Appendix A. While this resolution allows detectors such as HPGe to be accurately modeled, it also introduces computational challenges to response functions or transformation matrices.

Energy Range	**Energy Bins per MeV**
1 keV - 3 MeV	1024
3 MeV - 6 MeV	512
6 MeV - 10 MeV	256
10 MeV - 20 MeV	64
20 MeV - 100 MeV	32

Table 3.1: Energy Resolution for Photons

The dimensions of a transformation matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$ need not be the same. Let $\mathbf{E}_o \in \mathbb{R}^M$ be the vector of energy groups represent the outgoing energies and $\mathbf{E}_i \in \mathbb{R}^N$ represent the incoming or source energies. If $M < N$ (decrease in resolution), information is lost as energy groups are coalesced. If $M > N$ (increase in resolution), no information is lost, spectral transformations within the model may be more accurately modeled, but information from previous submodels is not increased. For example, a discrete gamma line produced in a previous submodel which is captured in a coarse incoming energy bin will retain its coarseness in the outgoing energy structure even if this structure is finer. However, if this gamma is produced within the submodel, the finer outgoing energy structure would be useful. Therefore, unless it is known that a submodel adds or removes significant spectral information, a constant energy structure ($\mathbf{E}_o = \mathbf{E}_i, M = N$) is assumed. However, energy-symmetry requires a relatively large amount of data from simulations. For instance, in a traditional simulation, one may query for an energy dependent flux with N energy bins, resulting in N data points. To construct an energy-symmetric response function, it is necessary to know both the source energy and resultant flux energy, requiring at least N^2 data points. For 14 energy groups this is a feasible 196 data points; for 8831 groups, this results in 7.8×10^8 data points, or approximately 0.5 Gigabytes of double-precision floating point data. The data storage requirement for thousands of these matrices alone is impractical. In addition, computing such a large amount of data would require an enormous computational effort, subverting one of the major benefits of this method.

To circumvent the direct computation of each transformation matrix, it is possible to generate them from energy-asymmetric transformations via interpolation. Energy-asymmetric transformations are those which have a coarser incoming energy resolution than the outgoing energies. By sampling a limited number of source energies, the number of data points is reduced drastically. However, to reconstruct an energy-symmetric transformation, an interpolation scheme on the source energy is required to reconstruct the fine-group structure.

3.1.1 Source Energy Interpolation

Interpolating between source energies requires identification of source-energy-dependent features in the results and methods to estimate those features for an interpolated result. As an example, consider an isotropic point source of photons at the center of a 2 mm sphere of lead. Choosing two source energies at 500 keV and 1 MeV, the energy dependent currents integrated over the surface of the sphere are shown in Figure 3.1. The characteristic features of these curves are a peak corresponding to the uncollided photons, x-ray peaks at lower energies, a bremsstrahlung continuum, a Compton continuum, and a sharp decline in the Compton continuum at the energy corresponding to a backscattered photon of source energy. Aside from the x-ray peaks, the location of these features in the energy dimension are dependent on the source energy.

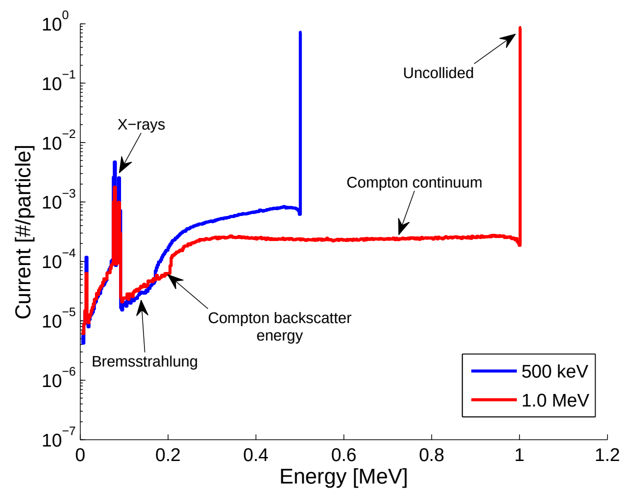


Figure 3.1: Current Integrated over Lead Sphere from 500 keV and 1 MeV Point Source

It is possible to estimate the integrated current exiting the same sphere from a different source energy by interpolating between the known 500 keV and 1 MeV source energies. However, if a simple direct interpolation is used, the results would be inaccurate due to source-energy-dependent features such as the uncollided peak. To preserve these features, the energy dimension must be shifted or transformed to match the predicted features of the new source energy. Transformation requires that the characteristic features of the spectrum be decomposed based on the physical processes which produced them.

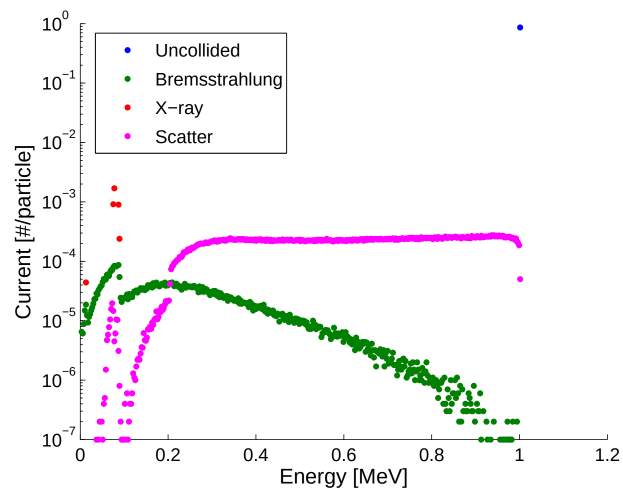


Figure 3.2: Spectral Components of Current Leaving Lead Sphere from 1 MeV Point Source

Let the two simulated source energies be S_A and S_B [MeV], the interpolated source energy be S_C [MeV], the simulated currents integrated over the submodel interface be $\mathbf{T}_A \in \mathbb{R}^M$ and $\mathbf{T}_B \in \mathbb{R}^N$ [$\gamma{\cdot}\mathrm{s}^{-1}$], and the energy bin structure for each be $\mathbf{E}_A \in \mathbb{R}^M$ and $\mathbf{E}_B \in \mathbb{R}^M$ [MeV] which need not be identical. An arbitrary energy bin structure $\mathbf{E}_C \in \mathbb{R}^L$ [MeV] is chosen for the interpolated result $\mathbf{T}_C \in \mathbb{R}^L$ [$\gamma{\cdot}\mathrm{s}^{-1}$]. The spectra may be decomposed by two methods covered here. Using the MCNPX tally tagging feature, the current may be flagged by the origin of the photon such as uncollided, x-ray, and bremsstrahlung as illustrated in Figure 3.2. This is the most direct and accurate method for decomposition. In some submodels, the use of tally tagging is not feasible. For these simulations, the results are decomposed via post-processing. The first step in this process is the identification and removal of any peaks in the spectrum resulting from discrete energy processes. This always includes the energy bin which contains the uncollided component. If the source energy is greater than two electron rest masses (the threshold for pair production), the peak at 511 keV resulting from positron-electron annihilation is included. Lastly, if any x-ray peaks in the material are known a priori, they are included. After identification, the values of the spectrum at these energies are removed from the total and treated separately. After peak identification and removal, the spectrum is split at the source backscatter energy E_b, given by Eq. 3.1 [40].

$$E_b(E) = \frac{E}{1 + 2E/m_e c^2}, \tag{3.1}$$

where E is the initial source energy in [MeV], and $m_e c^2$ is the rest mass of an electron (0.511 MeV). After splitting the spectrum, the lower portion is taken as the bremsstrahlung component, and the upper portion as the Compton continuum.

Peak Interpolation

Once the spectra are decomposed using tally tagging or post-processing, the discrete peaks are interpolated logarithmically using the source energies as the basis. The logarithmic interpolation function Z is given by

$$Z(x, x_1, y_1, x_2, y_2) = \exp\left[\ln(y_1) + \ln\left(\frac{y_2}{y_1}\right)\frac{x - x_1}{x_2 - x_1}\right], \tag{3.2}$$

where (x_1, y_1) and (x_2, y_2) are the known independent and dependent data point pairs and x is the unknown independent variable. A logarithmic interpolation scheme is chosen to model the exponential nature of cross-section data as a function of energy. Higher order interpolation schemes may be employed Using this interpolation scheme, each discrete peak is given by $Z(S_C, S_A, T_{C,i}, S_B, T_{B,j})$, where i, j are the bins containing the discrete peak values for the known source energies S_A and S_B.

Continuum Interpolation

The Compton continuum current and bremsstrahlung current are divided by their bin widths making the units [γ·s^{-1}·MeV^{-1}] and are linearly transformed with,

$$\mathbf{E} \leftarrow E'_1 - \frac{E'_2 - E'_1}{E_2 - E_1}\left(\mathbf{E} + E_1\right), \tag{3.3}$$

where $\mathbf{E}$ is the energy vector of interest, E_1 and E_2 are the original start and end point energies, and E'_1 and E'_2 are the new start and end point energies. The division by bin width is necessary to assure smooth continuums when uneven bin widths are employed. If tally tagging is available, the bremsstrahlung start point energy is unchanged at zero, the endpoint is changed to the new source energy. The Compton continuum start point is changed to the new source backscatter energy, while the endpoint is moved to the new source energy. If the results are instead decomposed via post processing, the bremsstrahlung start point is unchanged, and the end point is moved to the new source backscatter energy. The Compton continuum start point is moved to the new source backscatter energy and the end point moved to the new source energy. For example, for source energy S_A, to transform the continuous components to match the features from source energy S_C, the transformation ranges are summarized in Table 3.2.

	Tally Tagging		**Post-Process Decomp.**	
	Brems.	Compton	Brems.	Compton
Original	$(0, S_A)$	$(0, E_b(S_A))$,$(E_b(S_A), S_A)$	$(0, E_b(S_A))$	$(E_b(S_A), S_A)$
Transformed	$(0, S_C)$	$(0, E_b(S_C))$,$(E_b(S_C), S_C)$	$(0, E_b(S_C))$	$(E_b(S_C), S_C)$

Table 3.2: Example Energy Transformation Range

After transformation, a weighted bin logarithmic interpolation scheme is employed. The interpolated current $\mathbf{T}_C$ is given by

$$\mathbf{T}_{C,l} = Z(S_C, S_A, p_{A,l}, S_B, p_{B,l}) \quad \forall\, l \in L, \tag{3.4}$$

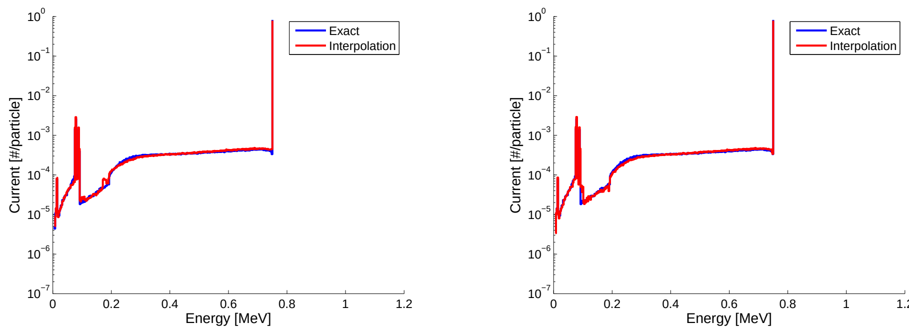
where Z is the interpolation function given by Eq. 3.2, and $p_{A,l}$ and $p_{B,l}$ are weighted sums of the currents given by

$$p_{A,l} = \sum_{i=1}^{I} w_{A,i}\mathbf{T}_{A,i}$$
$$p_{B,l} = \sum_{i=j}^{J} w_{B,j}\mathbf{T}_{B,j}. \tag{3.5}$$

The weights $w_{A,i}$ and $w_{B,j}$ are the widths of the transformed energy bins of $\mathbf{E}_A$ and $\mathbf{E}_B$ which are coincident with $\mathbf{E}_{C,l}$ in [MeV]. The weights are energy widths because the currents are in units of [γ·s^{-1}·MeV^{-1}]. This weighted scheme accounts for gradients in $\mathbf{T}_A$ and $\mathbf{T}_B$ within the energy bin of interest $\mathbf{E}_{C,l}$. As a simple example, if $\mathbf{E}_A = \mathbf{E}_B = \mathbf{E}_C$ and there is no transformation, then $I = J = 1$, $w_{A,1} = w_{B,1} = \mathbf{E}_{A,l}$, and $p_{A,l}$ and $p_{B,l}$ reduce to the original values of the current for that energy bin. By placing the continuums on a per MeV basis and using the weights as energy-widths, this scheme features consistent results even for irregular energy bin spacing.

After the discrete peaks are interpolated and continuous components are transformed and interpolated, they are summed together to produce an estimate for a current from source energy S_C. For example, considering the current leaving a 2 mm lead sphere from 500 keV and 1 MeV point sources in units of [#·particle^{-1}], by following the interpolation algorithm outlined here to estimate the current from a 750 keV source, the resulting spectrum is shown in Figure 3.3(a) and 3.3(b) along with the results from a direct simulation using both tally tagging and the post-processing decomposition methods. For either decomposition method, the error in the estimate is less than 20% for all energy bins except those close to the Compton backscatter energy, where they can be as high as 50% due to the sharp jump seen in the Compton component at those energies. However even with a coarse 500 keV spacing between source points, the overall spectral shape is preserved. Each of the components from the tally tagging simulation and interpolation are summed and compared in

Table 3.3. Even though the highest error is 13% for the X-ray component, it should be noted that the spacing between source points is exaggerated in this example and an implementation would have much smaller spacing.



(a) With Tally Tagging (b) Without Tally Tagging

Figure 3.3: Interpolated Current at 750 keV Compared to Current from Simulation

	Current [#·particle^{-1}]				
	500 keV (MCNPX)	**1 MeV** (MCNPX)	**750 keV** (MCNPX)	**750 keV** (Interp.)	**Error** [%]
Uncollided	0.719	0.861	0.821	0.787	4
Bremsstrahlung	3.6×10^{-3}	6.7×10^{-3}	5.1×10^{-3}	5.2×10^{-3}	0.6
Scatter	7.2×10^{-2}	7.6×10^{-2}	7.8×10^{-2}	7.9×10^{-2}	0.8
X-ray	1.0×10^{-2}	3.8×10^{-3}	5.6×10^{-3}	6.3×10^{-3}	13

Table 3.3: Components of Interpolated Current using Tally Tagging at 750 keV Compared to Direct MCNPX Simulation

This example interpolated between 500 keV and 1 MeV source energies to estimate the spectrum from a 750 keV source. At these energies, the bremsstrahlung component is small compared to the Compton continuum. At higher photon energies, such as the current leaving a 2 mm sphere of lead from a 20 MeV point source as shown in Figure 3.4, the opposite is true. However, the same decomposition and interpolation algorithm used for lower source energies still holds. For example, if 10 MeV and 20 MeV are simulated and interpolated to estimate the current leaving the sphere for a 15 MeV source, the results are compared to a direct 15 MeV source simulation in Figure 3.5. This result demonstrates the interpolation algorithm is valid with higher energy sources as well.

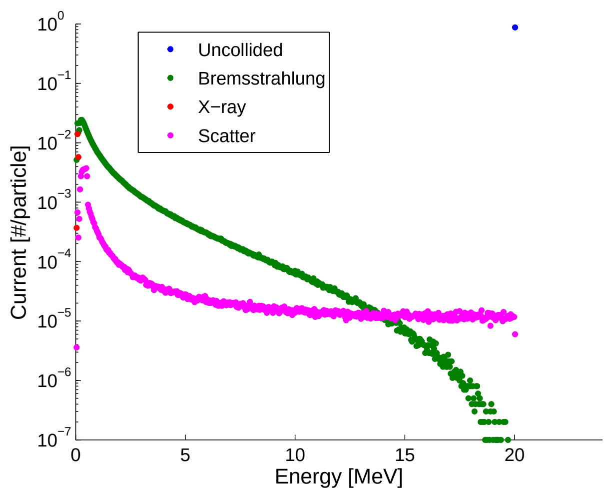


Figure 3.4: Spectral Components of Current Leaving Lead Sphere from 20 MeV Point Source

Thus far, the interpolation of source energies has focused on the current exiting a model, or the energy escaping. For detector response functions, the results estimate the energy captured within the model, which has different source-energy-dependent features. For example, consider a 1 MeV beam inci-

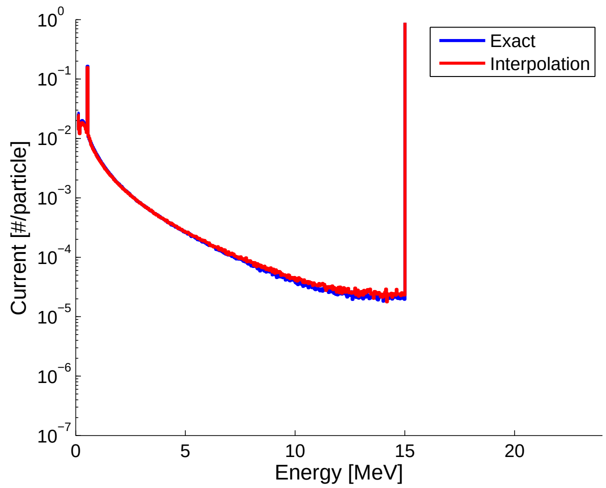


Figure 3.5: Interpolated Current at 15 MeV Compared to Current from Simulation

dent upon a 2 × 2 × 5 cm HPGe crystal in a vacuum. The resulting detector signal is shown in Figure 3.6. The signal shown represents an ideal detector as no Gaussian energy broadening is applied. Some features characteristic of gamma spectroscopy are labeled such as the photopeak and Compton edge. Missing from the spectrum are a backscatter peak and X-ray lines, which are actually phenomena from materials surrounding the detector.

Because tally tagging is not possible with a pulse-height tally in MCNPX, the results must be post processed for decomposition. Similar to before, any discrete peaks are first removed. This includes the photopeak, annihilation peaks, and any escape peaks. The spectrum is split at the Compton edge, and linearly transformed to match the Compton edge of the desired source energy. If this decomposition and transformation is applied to a 500 keV beam and the 1 MeV beam, and the results are interpolated to estimate the signal from a 750 keV beam, the resulting signal is shown in Figure 3.7 in comparison with a direct simulation of a 750 keV beam. Errors are less than 10% for the entire spectrum.

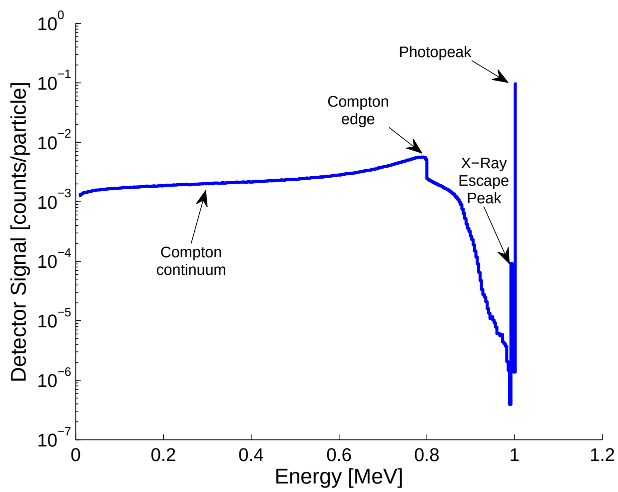


Figure 3.6: HPGe Detector Signal from a 1 MeV Beam Source

This interpolation algorithm allows a limited number of source energies to be simulated and then interpolated to estimate results from source energies not simulated. This is required for constructing energy-symmetric transformation matrices from energy-asymmetric ones. For instance, if $\mathbf{E}_o \in \mathbb{R}^M$ is the energy bins used for a result such as a current leaving a sphere, and $\mathbf{E}_i \in \mathbb{R}^N$ are the limited source points sampled, then these results may be used directly to construct the transformation matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$. As previously discussed, having $M > N$ is undesirable. This is solved by setting $\mathbf{E}_i' = \mathbf{E}_o$, and placing the known source points within $\mathbf{E}_i'$. This leaves the expanded matrix

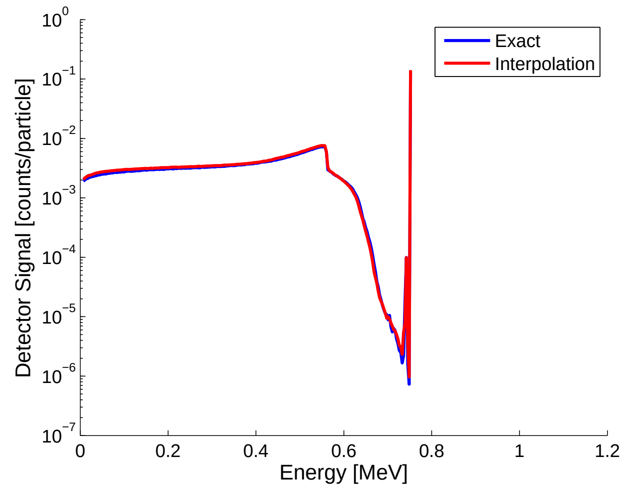


Figure 3.7: Interpolated Detector Signal at 750 keV Compared to Simulation

$\mathbf{A}' \in \mathbb{R}^{M\times M}$ lacking columns of data corresponding to the results from source energies not sampled. By utilizing this interpolation method, the columns of data may be estimated, forming an energy-symmetric transformation matrix. Thus, a high level of energy resolution is achieved for the Green's functions in a computationally tractable manner.

Simulated radiation transport results are complex functions of material cross-section data. Source points should be sampled at intervals which effectively capture the gradients in cross-section data. As shown in Figure 3.8, the total attenuation coefficient for a variety of elements follows the same general trend. At low energies (1 keV to ≈300 keV), photoelectric absorption creates steep gradients and resonances corresponding to electron shell energies. At higher energies above 300 keV, the gradient in the cross section is lessened. Therefore the source energy point should be closely spaced at low energies and progressively widened at higher energies.

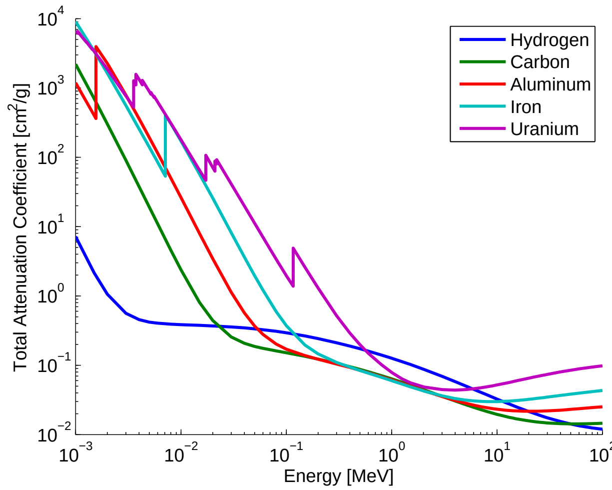


Figure 3.8: Total Attenuation Coefficient for Common Elements

If the source energy structure is in Table 3.4 is used, it results in 67 points. The exception to this regular interval of source energies is around the pair production threshold at 1.022 MeV, where source energies are sampled around this threshold to avoid erroneous annihilation peaks during interpolation. Compared to a direct computation of transformations (7.8×10^8 data points), this limited source energy sampling reduces the number of data points by over two orders of magnitude (5.9×10^5 data points). A full listing of these source energies are in Appendix A.

Because the error incurred with this source energy structure is dependent on material cross sections, an exhaustive estimate of the error for all models is infeasible. Instead, the ability of this interpolation method to estimate results at the logarithmic midpoints of this energy structure, where the error

Energy Range	**Energy Interval Between Points**
1 keV - 100 keV	10 keV
100 keV - 3 MeV	100 keV †
3 MeV - 10 MeV	1 MeV
10 MeV - 100 MeV	5 MeV

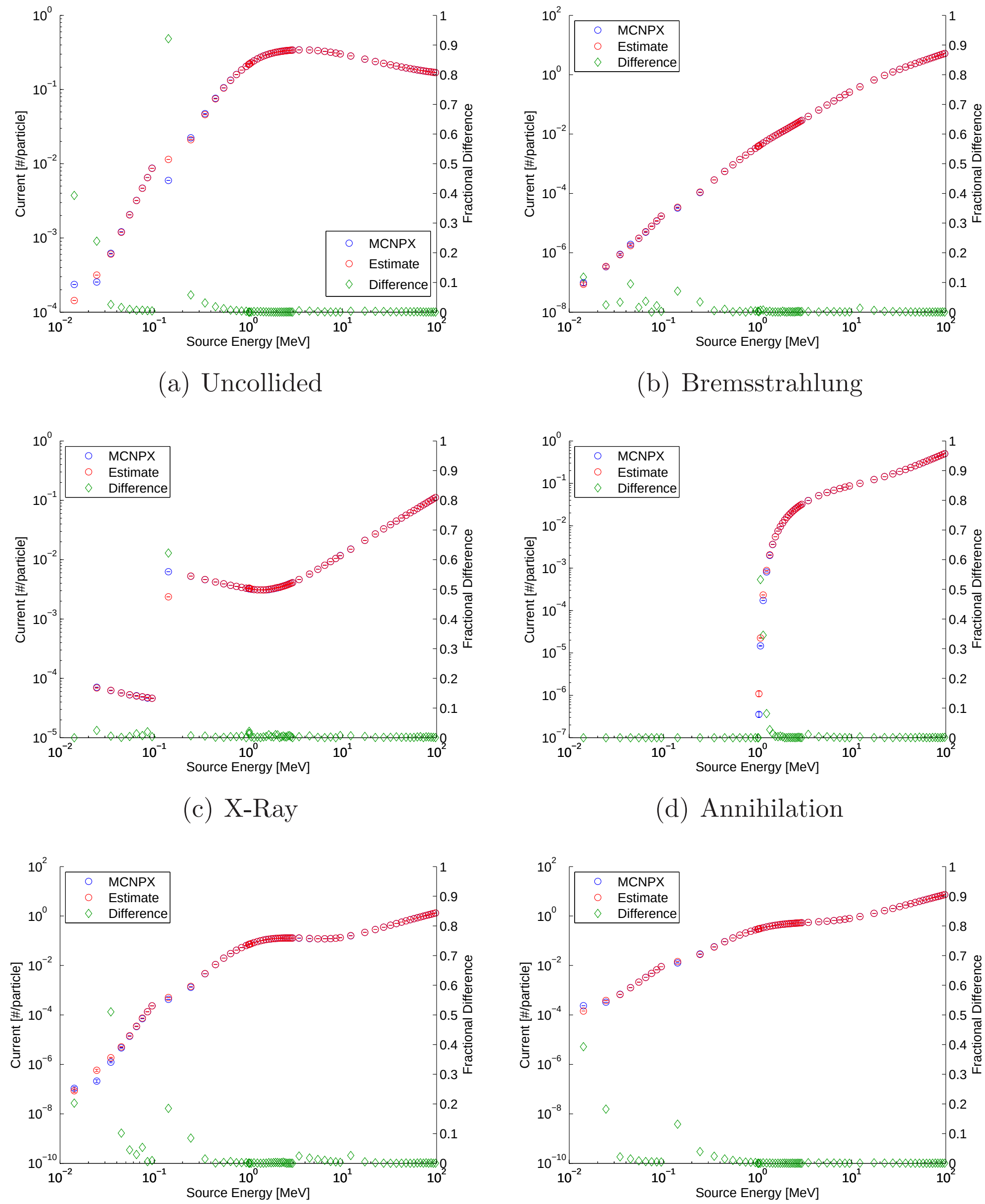
† except around 1.022 MeV

Table 3.4: Source Energy Points for Photons

should be the largest, is examined for two different models. The first model is a 1 kg sphere of uranium. The current integrated over the surface the sphere from a uniform volumetric source is simulated with MCNPX at the source points in Table 3.4, as well as their logarithmic midpoints. The ability of the interpolation method to estimate the results at these midpoints is compared to the simulated results. The simulated and estimated spectra are summed component-wise and the fractional difference between them computed. The summed components and fractional difference as a function of midpoint source energy are plotted in Figure 3.9. The statistical error is plotted for both data sets, although for the majority of the data points they are too small to distinguish. For the majority of interpolated points, the difference is less than 10%. The primary exception to this is the estimated source energy at 141 keV, which is within the uranium K-shell photoelectric absorption resonance, and at lower energies close to 10 keV near the L-shell resonance. Therefore, because the material cross section is non-monotonic in these intervals, the interpolation scheme produces significant error. In Figure 3.9(d), the error at the pair-production threshold 1.022 MeV is also large as the difference in the current between source points spans multiple orders of magnitude. However, as demonstrated by the total in Figure 3.9(f), the annihilation contribution is negligible and its contribution to the overall error as well. Therefore, in general the interpolation method with this source point structure can estimate the integrated current to within 10%, except near cross-section resonances where the error can be significant.

Because the detector signal interpolation is an inherently different process, the error from source energy interpolation of the detector signal is also examined. The second model is a 5×5×5 cm HPGe detector in vacuum. Using the same midpoint source energies as before, the total estimated detector signal as a function of source energy is compared to the detector signal as computed by MCNPX in Figure 3.10 along with the fractional difference between the two. For all source energies, the error is much less than 10%, with the highest error being 3% at 141 keV.

The source energy interpolation scheme is utilized by all submodels to reduce the number of source points sampled from simulations. In addition, each submodel employs additional parameterization models to decrease the amount of data queried from simulations. The following sections describe these parameterizations.



(a) Uncollided (b) Bremsstrahlung

(c) X-Ray (d) Annihilation

(e) Scatter (f) Total

Figure 3.9: Interpolated Current at Midpoint Source Energies Compared to MCNPX Simulations

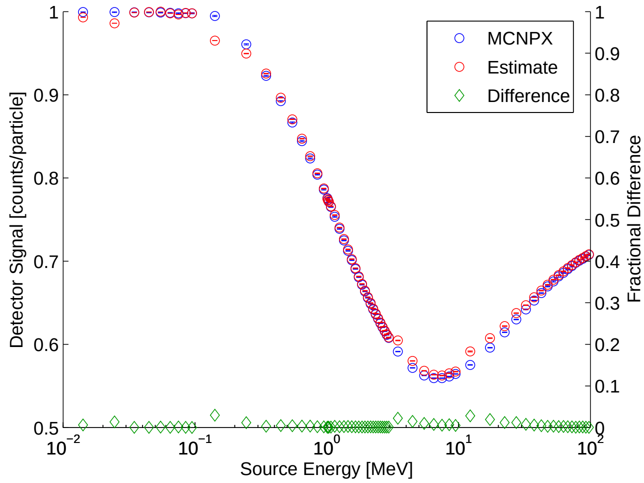


Figure 3.10: Interpolated Detector Signal at Midpoint Source Energies Compared to MCNPX Simulations

3.1.2 Special Nuclear Material

As previously demonstrated, the current leaving an SNM sphere of 1 kg has reached a saturation point in the spectrum and is only dependent on the surface area to volume ratio. However, unlike previous work which generated SNM Green's functions with respect to individual isotopes of uranium and plutonium at different ages, these simulations are done with respect to uniform volumetric sources at each source energy. The source energies are determined by inputing the initial isotopic mixture and age to the RadSrc software package [41] which yields $\mathbf{q}$, the vector of gamma and bremsstrahlung source of photons in $[\gamma{\cdot}\mathrm{s}^{-1}{\cdot}\mathrm{g}^{-1}]$. The transformation data for the SNM, $\mathbf{R}^{\mathrm{snm}}$ (unitless probability) is produced by the source energy interpolation algorithm on current data. Applying this to a source of photons within the sphere $\mathbf{q}$ $[\gamma{\cdot}\mathrm{s}^{-1}{\cdot}\mathrm{g}^{-1}]$

and scaling by surface areas, volumes, and mass, yields the current of photons leaving the SNM $\mathbf{q}^{\text{snm}}$ $[\gamma\cdot\text{s}^{-1}]$. Because the energy structure of any response matrix $\mathbf{R}$ does not necessarily match that of a source vector $\mathbf{q}$, a mapping between energy groups must be employed. Let $\mathbf{E}_o(\mathbf{R})$ and $\mathbf{E}_i(\mathbf{R})$ be the outgoing and incoming energy bins of response $\mathbf{R}$, respectively, and $\mathbf{E}(\mathbf{q})$ be the energy bins of vector $\mathbf{q}$. A mapping matrix $\mathbf{I}(\mathbf{E}_1 \rightarrow \mathbf{E}_2) \in \mathbb{R}^{M\times N}$ (unitless) is created which maps energy structure $\mathbf{E}_1 \in \mathbb{R}^N$ onto $\mathbf{E}_2 \in \mathbb{R}^M$, assuming uniformity within each energy bin. The matrix $\mathbf{I}$ reduces to the identity matrix when $\mathbf{E}_1 = \mathbf{E}_2$. Therefore, the current integrated over the SNM sphere $\mathbf{q}^{\text{snm}}$ $[\gamma\cdot\text{s}^{-1}]$ is

$$\mathbf{q}^{\text{snm}} = m\frac{V}{S}\frac{S_0}{V_0}\mathbf{R}^{\text{snm}}\mathbf{I}\left(\mathbf{E}(\mathbf{q}) \rightarrow \mathbf{E}_i(\mathbf{R}^{\text{snm}})\right)\mathbf{q}, \tag{3.6}$$

where $\mathbf{E}_i(\mathbf{R}^{\text{snm}})$ is the incoming energy bins for response $\mathbf{R}^{\text{snm}}$, $\mathbf{E}(\mathbf{q})$ is the energy bins for source $\mathbf{q}$, S_0/V_0 is the initial surface area to volume ratio for the 1 kg mass used in the simulations, V/S is the volume to surface area ratio for the desired mass, and m is the mass of the SNM in grams.

3.1.3 Shielding

The shielding is a spherical shell completely surrounding the SNM sphere. Two different shielding types are considered: lead and 10 % borated polyethylene (BPE). Multiple thicknesses of each shield are simulated. The thickness intervals and maximum thickness of the shield are functions of the source energy. The intervals are uncollided half-value-layers (HVL), where each additional interval halves the uncollided radiation. The maximum thickness is taken to be 100 HVLs, which is an attenuation factor of $1/2^{100} = 7.9 \times 10^{-31}$. If a thickness beyond the maximum is specified, that source energy's contribution is assumed to be negligible and is set to zero. The spacing of the thickness intervals at the HVL also effectively captures the gradient in attenuation making interpolation between intervals more accurate.

Because the mass and volume of the SNM is variable, the inner and outer radii of the shielding are variable. The inner radius also affects the angular distributions of photons entering the shielding. To avoid simulating the combination of shielding thicknesses with SNM radii and angular distributions, the geometry is modeled as a point source in the center of spherical layers of lead shielding, and the results are modified by two approximations to correct for varying SNM radii and angular distributions.

Angular Distribution Approximation

The SNM and shielding geometry is illustrated in Figure 3.11. Because of symmetry, the angular distribution of the current on the surface of the SNM is only a function of angle $\theta \in [0, \pi/2]$ from the SNM normal $\mathbf{n}$. Let $\mu \in [0, 1]$ be the cosine of this angle ($\mu = \cos\theta$). For many surface sources, the angular distribution follows some power n of a cosine distribution, $p(\mu)d\mu = (n+1)\mu^n d\mu$.

The angular distribution of the photons on the surface of the SNM sphere is a function of source and outgoing energies. For example, consider Figure 3.12 which plots the normalized angular distribution on the surface of a

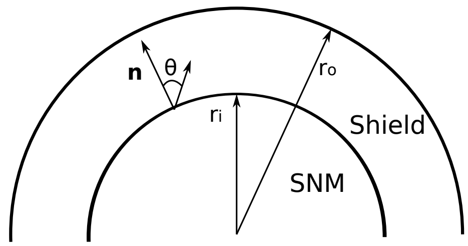


Figure 3.11: SNM and Shielding Angular Distribution

2 kg sphere of metal plutonium for various monoenergetic sources distributed uniformly within the volume. In the same figure a cosine and cosine-squared distribution are plotted for comparison. For a cosine distribution ($n = 1$), this corresponds to an isotropic radiation field, which occurs when the sphere is optically thick and reduces to a surface source. A cosine-squared distribution ($n = 2$) corresponds to a sphere void of material and is more forward directed. Therefore, the cosine and cosine-squared distributions bound all possible angular distributions for a uniform source in a spherical geometry.

From Figure 3.12, the angular distribution becomes more forward directed at higher source energies. This effect is not due to uncollided radiation, but rather the outgoing energies. As an illustration of how the distribution is a function of the outgoing energies, Figure 3.13 plots the scattered and uncollided radiation components from the same scenario. The scattered radiation in Figure 3.13(a) is more forward directed than the uncollided component in Figure 3.13(b), therefore the angular distribution is a function of both source and outgoing energies.

To reduce the dimensionality of the combinations of angles, source en-

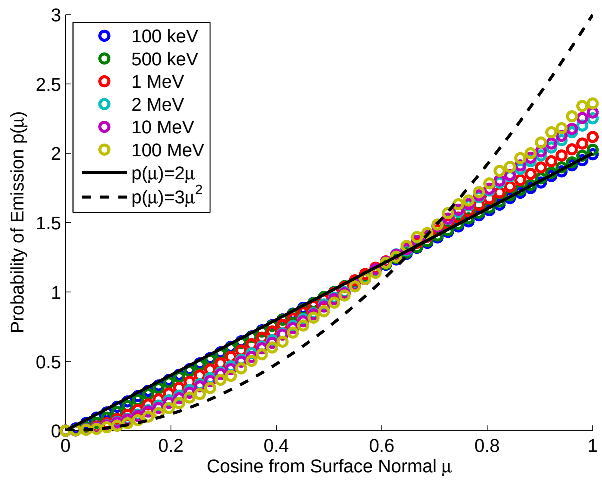
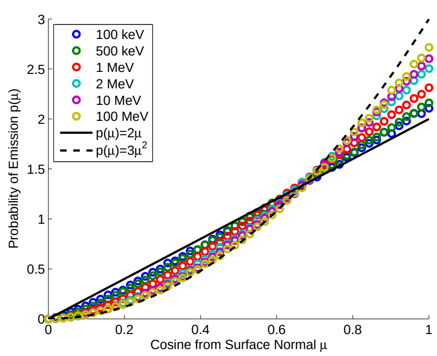
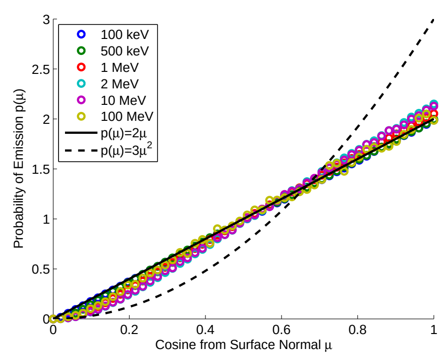


Figure 3.12: Angular Distribution from Plutonium Metal Sphere



(a) Scattered Radiation



(b) Uncollided Radiation

Figure 3.13: Components of Angular Distribution from Plutonium Metal Sphere

ergies, and outgoing energies, a cosine function may be fitted to the data to estimate the power n of the cosine distribution. This is achieved by fitting the data with the function $f(\mu) = (n+1)\mu^n$, where n is the free parameter. Because n is bounded between 1 and 2, this non-linear function can be fit with just a few iterations using the Gauss-Newton algorithm. If this is done for the same plutonium metal sphere and the various components of the angular distribution, Figure 3.14 displays the trend in cosine power as a function of source energy. Electrons are explicitly tracked in this simulation.

As expected, the uncollided component follows the inverse of the total cross-section as shown in Figure 3.8. The X-ray and annihilation components are approximately constant, as x-ray and annihilation photons are isotropically emitted and independent of source energy. The angular distribution of scattered photons are highly anisotropic with increasing energy, becoming more forwarded directed. Because these glancing collisions do not appreciably change the initial direction or energy of the source photon, the scattered component essentially replicates the angular distribution expected from a sphere void of material. Similarly, the average angle of a bremsstrahlung photon is more forward directed with increasing energy; thus, it follows the same trend as the scattering component.

The cosine power is computed based on 50 energy groups from 1 keV to 100 MeV for both source and tally energies for a 1 kg sphere of uranium metal, and is pictured as a surface in Figure 3.15(a). The surface is generally smooth with peaks and valleys near the photoelectric resonance energies. Based on the

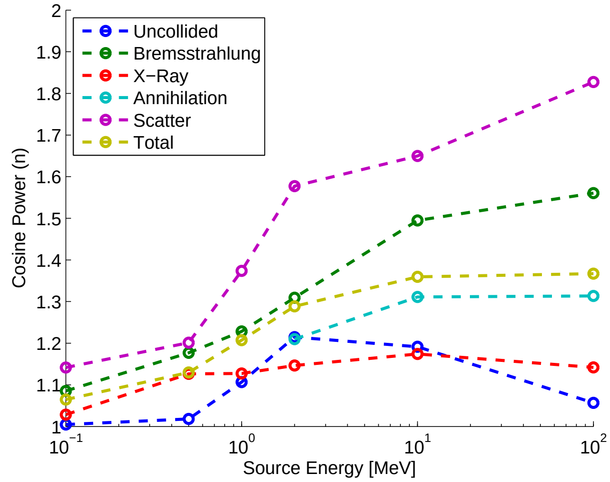
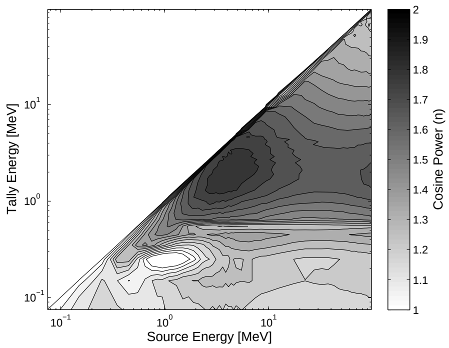


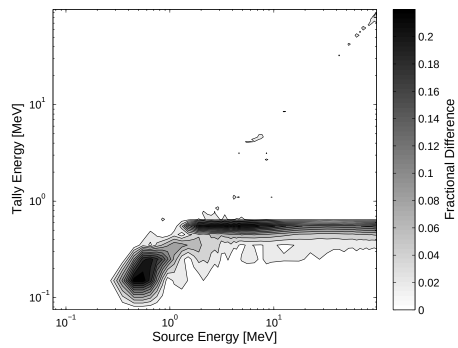
Figure 3.14: Cosine Power of Angular Distribution from Plutonium Metal Sphere

surface contour, the cosine power is a stronger function of tally energy than source energy. To determine convergence the number of energies is doubled, and the fractional difference is shown in Figure 3.15(b). The energy range 100 keV to 1 MeV in both the tally and source energies have errors as high as 20%. To reduce this, the number of energy groups is increased in that range, making 76 groups total. The increased resolution cosine power contour is shown in Figure 3.16(a). The number of energies is again doubled and the logarithm of the fractional difference is plotted in Figure 3.16(b). The difference is dominated by statistical error from MCNPX. The majority of fractional differences are less than 10%, except near 100 MeV source energies where the difference is as high as 50%.

This data may be used to compute an angular distribution for any

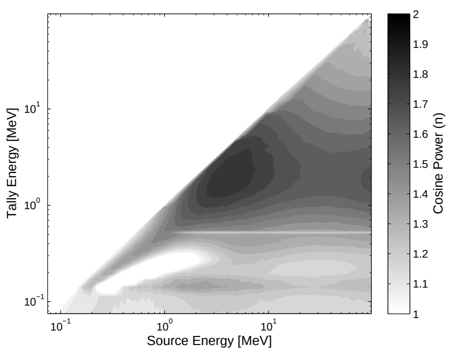


(a) Cosine Power of Angular Distribution

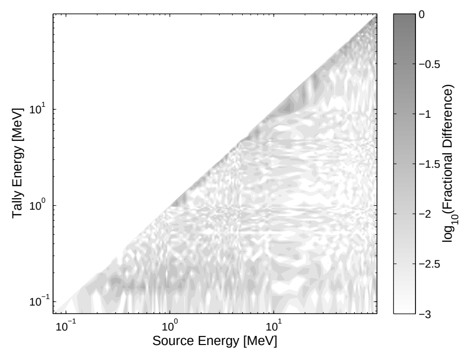


(b) Error of Cosine Power

Figure 3.15: Cosine Power of Angular Distribution from Uranium Metal Sphere



(a) Cosine Power of Angular Distribution



(b) Error of Cosine Power

Figure 3.16: Increased Resolution Cosine Power of Angular Distribution from Uranium Metal Sphere

combination of source energies within the SNM sphere. Let $F(E_{i,m}, E_{o,p}, M_k)$ be the probability that a source photon in energy bin $\mathbf{E}_i \in \mathbb{R}^M$ is transported to energy bin $\mathbf{E}_o \in \mathbb{R}^P$ at the surface of the SNM within cosine angle bin $\mathbf{M} \in \mathbb{R}^K$ to the surface normal. Then the angular distribution for energy bin $E_{o,p}$ is given by

$$\mathbf{C}_k = \sum_{m=1}^{M} F(E_{i,m}, E_{o,p}, M_k) q_m, \tag{3.7}$$

where $\mathbf{C} \in \mathbb{R}^K$. A cosine function is fit to data ($\mathbf{C}$,$\mathbf{M}$) to compute the cosine power n. This process is repeated for all energy bins $E_{o,p}$ until the cosine power as a function of energy at the surface of the SNM is computed.

The smoothness in the MCNPX-computed cosine power function allows a finer energy structure to be binned within the 76 coarse groups shown here. As an example, if the spectrum from HEU aged 20 years is simulated in a 1 kg sphere of uranium metal, the cosine power as a function of energy is shown in Figure 3.17. Also shown in Figure 3.17 is the estimated cosine power using the F function. The error points are placed at the mean of each energy bin. The 76 energy group F function is able to estimate the angular distribution as a function of exiting energy from thousands of discrete gamma energies within 10%.

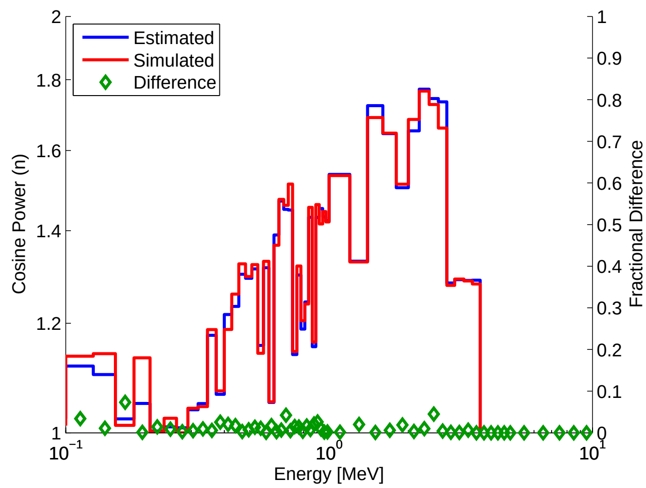


Figure 3.17: Cosine Power of Angular Distribution from HEU Simulation Compared to Estimate

Radial Approximation

The ratio of the SNM radius to shield radius affects the radiation transport within the shield by changing the apparent thickness of the shield. As an example, let r_i be the radius of the SNM in [cm], r_o be the radius of the shield in [cm], t be the thickness of the shielding in [cm] ($t = r_o - r_i$), and μ be the cosine of the angle from the SNM surface normal. If the angular distribution is $p(\mu)d\mu = \delta(\mu)d\mu$ (parallel to surface normal), where δ is the Dirac delta function, then photons emitted on the SNM surface observe a distance t of material to traverse. If photons are emitted at any other angle, they observe a larger distance of material. The chord length through the material as a function of emission angle $c(\mu)$ in [cm] is given by

$$c(\mu) = \sqrt{r_o^2 - r_i^2(1-\mu^2)} - r_i\mu, \tag{3.8}$$

which is derived from the law of cosines. Letting $R = r_i/r_o$, an alternative form is

$$c(\mu) = t\frac{\sqrt{1 - R^2(1-\mu^2)} - R\mu}{1-R} \tag{3.9}$$

The average chord length for a cosine distribution ($p(\mu)d\mu = 2\mu\ d\mu$) is given by the probability moment of Eq. 3.9,

$$\begin{aligned}\bar{c} &= t\int_0^1 p(\mu)\left(\frac{\sqrt{1-R^2(1-\mu^2)} - R\mu}{1-R}\right)d\mu \\ &= t\frac{R}{1-R}\left(R^3\left[1-\left(1-R^2\right)^{3/2}\right]-1\right). \end{aligned}\tag{3.10}$$

For the general cosine distribution of power n, Eq. 3.10 evaluates to

$$\bar{c} = t\frac{R}{1-R}\left(\frac{\sqrt{1-R^2}}{R}\,{}_2F_1\left(-\frac{1}{2},\frac{n+1}{2};\frac{n+3}{2};\frac{R^2}{R^2-1}\right) - \frac{n+1}{n+2}\right), \tag{3.11}$$

where ${}_2F_1$ is the hypergeometric function given by

$$ {}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!}, \tag{3.12} $$

and $(a)_n$ represents the factorial $(a)_n = a(a+1)(a+2)\ldots(a+n-1)$. Thus, the effective thickness of the shielding is a function of only the power n of the cosine distribution and the ratio of the SNM radius to the shielding radius. The cosine power may be determined by methods covered in the previous section. The effect of varying the ratio R is explored further.

The two limiting values of R are 0 and 1, corresponding to a point source in the center of a sphere and a semi-infinite planar source and shield, respectively. As an example of the dependence of the probability of transmission on R, the MCNPX-simulated transmission probability through 1 cm of lead shielding is plotted in Figure 3.18 for various source energies, assuming a cosine source distribution into the shielding. For each source energy, the transmission probability is normalized against its value at $R = 0$ (point source). From Figure 3.18, it is clear that the ratio R affects the transmission probability, and the degree to which it does so is dependent on the source energy.

The shape of the transmission curve is convex for low energies and concave for higher energies. This curve can be estimated by computing the ratio of the uncollided transmission probability at the specified ratio to the

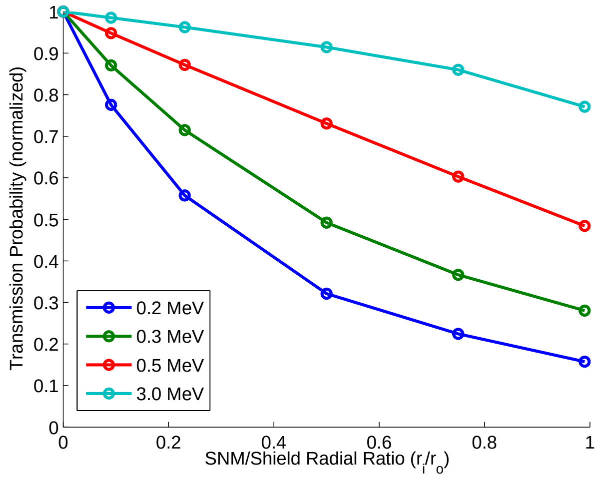


Figure 3.18: Shielding Transmission Probability as a Function of SNM/Shield Radial Ratio (normalized)

transmission probability for a point source, given by

$$S_R(t,R,E) = \int_0^1 p(\mu)\exp\left[N^{\mathrm{mfp}}(E)\left(1 - \frac{\sqrt{1-R^2(1-\mu^2)}-R\mu}{1-R}\right)\right]d\mu, \tag{3.13}$$

where $N^{\mathrm{mfp}}(E)$ is the number of mean-free-paths for a photon of energy E and $S_R(t,R,E)$ is the unitless scaling factor. This integral has no closed-form solution and must be evaluated numerically. This scaling factor may be applied to the simulated point source data to estimate the decrease in transmission probability. Figure 3.19 compares S_R to the simulation results shown in Figure 3.18. The simulation data is matched within 10% using this estimate.

The angular approximation and radial approximation are useful for scaling the shielding point source simulation in which no angular distribution is assumed. The angular distribution (cosine power) as a function of energy

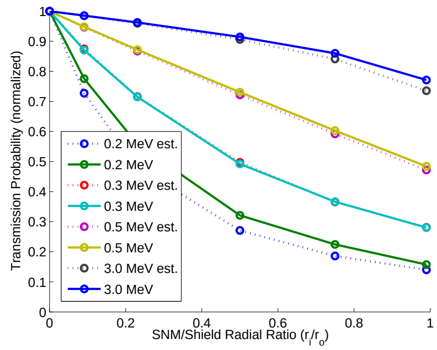


Figure 3.19: Normalized Transmission Probability Estimate Compared to Simulation

at the surface of the SNM is computed with the function $F(E_{i,m}, E_{o,p}, M_k)$. This cosine power is injected into Eq. 3.13 to estimate the relative decrease in transmission probability for each energy entering the shielding. This scaling factor S_R is applied to the current from the simulated monoenergetic sources. Each source energy at the nearest HVL intervals are logarithmically interpolated to the correct shielding thickness and then interpolated between the source energies to construct the response matrix $\mathbf{R}^{\text{shld}}$ (unitless). The current integrated over the shielding surface is found by applying the scaling factor S_R and the shielding response function to the current leaving the SNM,

$$\begin{aligned}\mathbf{q}^{\text{shld}} =& (1 - f_\Omega^{\text{shld}})\mathbf{R}^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{q}^{\text{snm}}) \rightarrow \mathbf{E}_i(\mathbf{R}^{\text{shld}})\right)\mathbf{q}^{\text{snm}} \\ &+ f_\Omega^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{q}^{\text{snm}}) \rightarrow \mathbf{E}(\mathbf{q}^{\text{shld}})\right)\mathbf{q}^{\text{snm}},\end{aligned} \tag{3.14}$$

where $\mathbf{q}^{\text{shld}}$ is the current leaving the shield in $[\gamma{\cdot}\text{s}^{-1}]$, and f_Ω^{shld} is a solid angle streaming fraction. The solid angle streaming fraction is the portion of the solid angle subtended by the shielding, which is all angles for the SNM/shielding interface, which is devoid of shielding material. This allows photons to stream uncollided through the shielding.

3.1.4 Vehicle

In previous work, a semi-infinite cargo container was used with an estimate of the importance function to approximate the time-dependent detector face current per particle emerging from the shielded SNM [4]. However, because this importance function was estimated from weight-windows, the energy resolution was limited to fourteen groups. In this work, a full tractor truck trailer is surrounded by detector tally planes along the length of the truck to account for detector placement and the time dependence of the moving vehicle. Each plane tallies the current [$\#\cdot\text{particle}^{-1}\cdot\text{cm}^{-2}$]. The actual detector position, if present at that location, is taken to be the center of this plane. Multiple point sources are sampled throughout the cargo to account for source placement. The point sources neglect any dependence the vehicle's response function may have on transport back through the shielded SNM and then to the detector face. A disadvantage to enumerating all detector and source positions is the large number of response functions generated from the combination of detector and source positions. A simple depiction of this geometry is shown in Figure 3.20 where the "s" circles represent the three-dimensional matrix of source positions and the "d" circles represent the planes of detector positions on the side and on the top of the vehicle.

The cargo response function is comprised of combinations of discrete

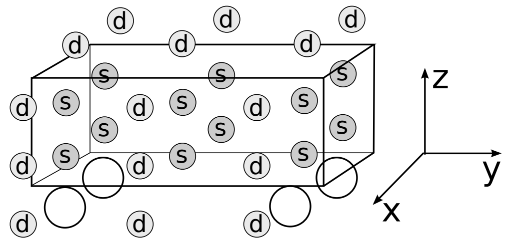


Figure 3.20: Vehicle Source and Detector Geometry

source and detector positions. In general, there are more detector positions available than source positions. In most cases, the actual detector and source position do not lie on these discrete points; thus, an interpolation scheme is required. There are two different interpolations utilized, depending on how close the source is to the detector. For both algorithms, it is assumed that the two dimensions which are not in the direction along the length of the vehicle (in the direction of motion) have been interpolated upon. For example, if the length of the vehicle is in the y-direction as shown in Figure 3.20, the x- and z-components are linearly interpolated out of each detector-source response function prior to these algorithms.

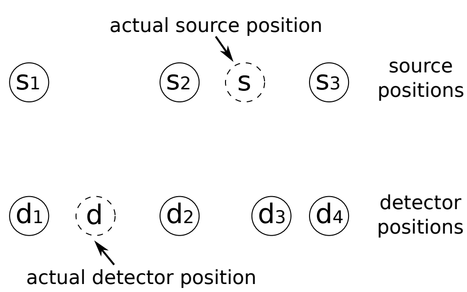


Figure 3.21: Detector and Source Positions for Far-Approach Algorithm

The first algorithm is used when the the source is an appreciable distance from the detector, defined as when the specified source position lies outside the bounds of the two closest sources to the specified detector position. Consider Figure 3.21. Here, d_1, d_2, d_3, d_4 are detector points and d is the actual detector position. Similarly, s_1, s_2, s_3 are the source points with s being the actual source position.

1. Consider the solid-line detector-to-source lines (d_1, s_2) and (d_1, s_3) shown in Figure 3.22. In this figure and all other figures, the solid lines represent known detector/source response pairs, and the dashed line indicates the resulting pair from the interpolation. If a linear interpolation is made between these responses using the distance between the sources as a basis, a response can be estimated for dashed-line (d_1, s).

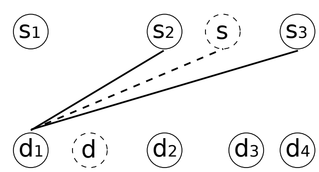


Figure 3.22: Vehicle Interpolation Far-Approach Algorithm: Step 1

2. If this interpolation is repeated for detector position d_2 instead of d_1, the line (d_2, s) is generated as shown in Figure 3.23.

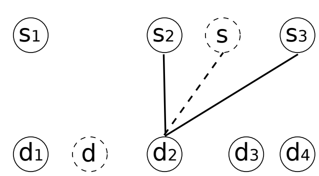


Figure 3.23: Vehicle Interpolation Far-Approach Algorithm: Step 2

3. With the two interpolated lines (d_1, s) and (d_2, s) known, interpolating between the detector positions using the distances between detectors as a basis reveals (d, s), the line for the actual detector source position as shown in Figure 3.24.

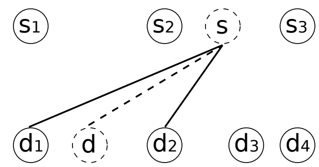


Figure 3.24: Vehicle Interpolation Far-Approach Algorithm: Step 3

The situation in which the closest two sources to the specified detector position bound the specified source position requires an interpolation scheme which preserves the closest-approach between the source and detector, which is expected to be the peak detector signal. The data is generated such that for every source position, there is a detector which captures the closest-approach distance. This algorithm describes an interpolation scheme which preserves this peak. Consider Figure 3.25, in which source positions s_1 and s_2 are the closest to the actual detector position d, and that these two source positions bound the actual source position s. The steps to interpolate this scheme are outlined here.

1. Interpolate between lines (d_1, s_1) and (d_3, s_2) to virtual source position (d, s') using the distance between sources as the basis for interpolation as shown in Figure 3.26. This calculates where the source would be to produce a peak signal in the detector.

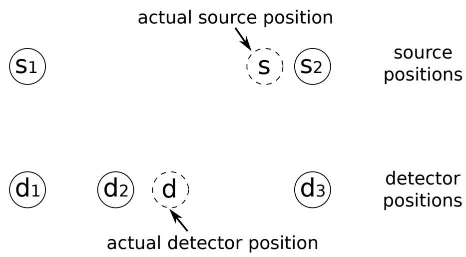


Figure 3.25: Detector and Source Positions for Close Approach Algorithm

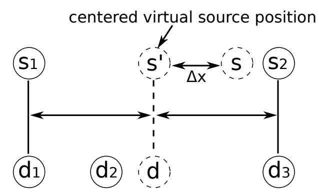


Figure 3.26: Vehicle Interpolation Close-Approach Algorithm: Step 1

2. Calculate the distance between the virtual source position and the actual source position $\Delta x = |s - s'|$. If $\Delta x < \varepsilon$, where ε is 0.1 cm, then this is the actual source position and the correct response to use. If $\Delta x > \varepsilon$, then go to step 3.

3. Interpolate between lines (d_2, s_2) and (d_3, s_2) to (d, s_2) using the distance between detectors as a basis for interpolation. This computes the actual detector response to a source positioned at s_2.

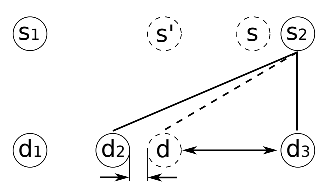


Figure 3.27: Vehicle Interpolation Close-Approach Algorithm: Step 3

4. Interpolate between lines (d, s_2) and (d, s') to (d, s) using the distance

between sources as the basis for interpolation. This computes the correct response based on how far from the peak source position the actual source is.

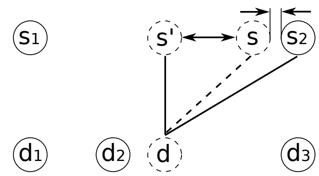


Figure 3.28: Vehicle Interpolation Close-Approach Algorithm: Step 4

The vehicle response function $\mathbf{R}^{\mathrm{car}}$ is produced by first interpolating on the geometric dimensions as was just described and then using the source energy interpolation algorithm to generate a energy-symmetric transformation matrix. This matrix is divided by the area of the detector in the simulation, A_{det} [cm^2]. The current at the detector face $\mathbf{q}_A^{\mathrm{car}}$ [$\gamma\cdot\mathrm{s}^{-1}\cdot\mathrm{cm}^{-2}$] is

$$\mathbf{q}_A^{\mathrm{car}} = \frac{1}{A_{\mathrm{det}}}\mathbf{R}^{\mathrm{car}}\mathbf{I}\left(\mathbf{E}(\mathbf{q}^{\mathrm{shld}}) \rightarrow \mathbf{E}_i(\mathbf{R}^{\mathrm{car}})\right)\mathbf{q}^{\mathrm{shld}}. \tag{3.15}$$

3.1.5 Detector

Photon detectors come in a variety of dimensions and compositions. This work addresses HPGe, NaI, and PVT detectors of variable size. It is assumed that they are rectangular prisms. The response function is generated with respect to a beam perpendicularly incident on a single face of the detector with corrections described below to take into account view factors and apparent thicknesses for the actual incident radiation fields. The detector size is varied in all three dimensions. A response for an actual detector size is computed by interpolating on the three dimensions. An apparent detector thickness is introduced based on the relative position of the source to the detector, illustrated in Figure 3.29.

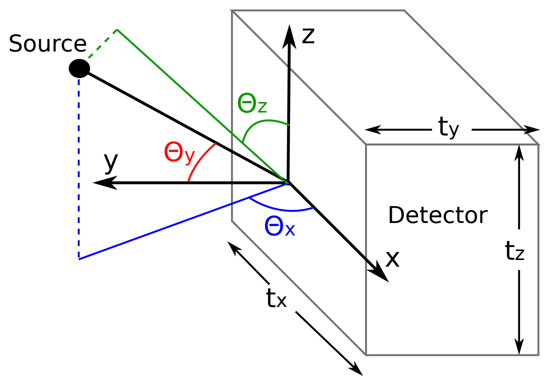


Figure 3.29: Relative Source and Detector Position Angles

Let t_x, t_y, and t_z be the physical dimensions of the detector in [cm]. If $\mathbf{d}$ is the vector from the center of the detector face to the source, then θ_y is the angle between $\mathbf{d}$ and the detector face normal, which is the y-axis in this example. The y-dimension (t_y) of the detector is increased by a factor of $1/\cos\theta_y$ as the chord through the detector from the source is equal to $t_y/\cos\theta_y$. However, as θ_y increases, the leakage out the sides of the detector in the x- and z-directions also increases. This is compensated by reducing the x-dimension by a factor of $\sin\theta_x$ and the z-dimension by a factor of $\sin\theta_z$. These angles are calculated by projecting the detector-to-source vector on the z- and x-planes and computing the angle between the projection and x- and z-axis, respectively. To summarize, the dimensional transformations are

$$
\begin{aligned}
t'_x &= t_x \sin\theta_x \\
t'_y &= \frac{t_y}{\cos\theta_y} \\
t'_z &= t_z \sin\theta_z,
\end{aligned} \tag{3.16}
$$

where t'_x, t'_y, and t'_z are the apparent dimensions of the detector in [cm]. In previous work, multiple rays from the source point through the detector volume are drawn to compute various chord lengths [4]. These chord lengths are used as the thickness for one-dimensional detector responses, which are averaged. This ignores any leakage out the sides of the detector. While this is sufficient for large area detectors such as PVT, the leakage from the sides of small detectors is significant and must be accounted for with the methods presented here.

The coordinate system used in Figure 3.29 is not fixed. The side of the detector which subtends the largest solid angle is taken as the detector face. This method assumes that the flux at the detector face is dominated by radiation moving along the source-detector path, and does not account for the apparent thickness of the detector to radiation which has scattered.

Because the vehicle response function produces the current at the detector face in [$\gamma\cdot s^{-1}\cdot cm^{-2}$]. The area of the detector face must be computed. To account for multiple sides of the detector contributing to the total area, the effective area of the detector face, A_{eff}, is computed with

$$A_{\text{eff}} = \Omega|\mathbf{d}|^2, \tag{3.17}$$

where Ω is the total solid angle subtended by the detector and $\mathbf{d}$ is the source to detector face vector shown in Figure 3.29. This equation computes the equivalent area of the detector if it is projected onto a spherical surface with radius $|\mathbf{d}|$. Depending on the relative source-detector configuration, this effective area can underestimate or overestimate the photon current on the detector face, especially for small source to detector distances.

The error of this method is tested by sampling a limited number of source energies, positions, and detector dimensions for each detector material. The two detector dimensions are a small crystal ($3 \times 10 \times 3$ cm) and a large crystal ($10 \times 10 \times 40$ cm). Three source positions are tested. Let $(0, 0, 0)$ be the center of the face of the detector. The source positions tested are $p_1 = (0, 70, 0)$ cm (minimum detector standoff distance), and $p_2 = (500, 70, 100)$ cm (long distance and oblique angle). The source energies tested are $E_1 =$700 keV and $E_2 =$2 MeV. The results of the apparent detector dimension and effective area approximation are compared to MCNPX simulations in Tables 3.5, 3.6, and 3.7. The percent difference is defined as

$$\% \text{ Diff} = \frac{\text{simulation} - \text{approximation}}{\text{simulation}}. \tag{3.18}$$

As expected, the error is minimal when the source is far from the detector and the detector is small. For small source to detector distances and large detectors, the effective area of the detector is erroneous and the flux over the face of the detector, which is assumed uniform with the approximation, can vary greatly. The difference has a weaker dependence on source energy, which is explained by the effective dimensions not capturing the actual chord lengths traversed by all photons.

	Small Crystal		Large Crystal	
	E_1	E_2	E_1	E_2
p_1	6%	10%	14%	12%
p_2	0.3%	0.4%	13%	12%

Table 3.5: Percent Difference Between Integrated Detector Signal from Approximation and MCNPX Simulation for PVT Detector

	Small Crystal		Large Crystal	
	E_1	E_2	E_1	E_2
p_1	13%	7%	18%	10%
p_2	2%	0.3%	15%	10%

Table 3.6: Percent Difference Between Integrated Detector Signal from Approximation and MCNPX Simulation for NaI Detector

	Small Crystal		Large Crystal	
	E_1	E_2	E_1	E_2
p_1	8%	6%	16%	10%
p_2	-0.04%	0.2%	13%	10%

Table 3.7: Percent Difference Between Integrated Detector Signal from Approximation and MCNPX Simulation for HPGe Detector

After geometric interpolation on the effective dimensions t'_x, t'_y, and t'_z, the source energies are interpolated to generate a detector transformation matrix $\mathbf{R}^{\text{det}}$ [counts$\cdot\gamma^{-1}$]. This is scaled by the effective area and applied to the current at the detector face to produce a detector signal $\mathbf{c}^{\text{det}}_{\text{ideal}}$ [counts$\cdot$s^{-1}],

$$\mathbf{c}^{\text{det}}_{\text{ideal}} = A_{\text{eff}}\mathbf{R}^{\text{det}}\mathbf{I}\left(\mathbf{E}(\mathbf{q}^{\text{car}}_A) \rightarrow \mathbf{E}_i(\mathbf{R}^{\text{det}})\right)\mathbf{q}^{\text{car}}_A. \quad (3.19)$$

The detector response functions are for an ideal detector without Gaussian energy broadening. After the detector signal is computed, the standard deviation of a photopeak in MeV is specified by

$$\sigma(E) = \frac{\text{FWHM}}{2\sqrt{2\ln 2}} = \frac{a + b\sqrt{E + cE^2}}{2\sqrt{2\ln 2}}, \tag{3.20}$$

where FWHM is the full-width-at-half-maximum of the peak in [MeV], E is the energy of the peak in [MeV], and a [MeV] b [MeV$^{1/2}$] and c [MeV^{-1}] are experimentally determined parameters. Let $\mathbf{E} \in \mathbb{R}^M$ be the energy bins of the ideal detector signal $\mathbf{c}^{\text{det}}_{\text{ideal}} \in \mathbb{R}^{M-1}$. The broadening matrix $\mathbf{G} \in \mathbb{R}^{M-1\times M-1}$ is given by

$$\begin{aligned} \mathbf{G}_{m,n} &= F\left(\mathbf{E}_{m+1}, \sqrt{\mathbf{E}_{n+1}\mathbf{E}_n}, \sigma(\sqrt{\mathbf{E}_{n+1}\mathbf{E}_n})\right) \\ &- F\left(\mathbf{E}_{m}, \sqrt{\mathbf{E}_{n+1}\mathbf{E}_n}, \sigma(\sqrt{\mathbf{E}_{n+1}\mathbf{E}_n})\right) \quad \forall\, m,n \in [1, M-1], \end{aligned} \tag{3.21}$$

where $\sqrt{\mathbf{E}_{n+1}\mathbf{E}_n}$ is the geometric mean of energy bin bounded by energies $\mathbf{E}_n \to \mathbf{E}_{n+1}$ and $F(x;\mu,\sigma)$ is the cumulative distribution function with mean μ standard deviation σ at x. The broadened detector signal $\mathbf{c}^{\text{det}}$ [counts$\cdot$s^{-1}] is then

$$\mathbf{c}^{\text{det}} = \mathbf{G}\mathbf{c}^{\text{det}}_{\text{ideal}}. \tag{3.22}$$

This approximates the energy resolution found in realistic detectors.

3.1.6 Background Radiation

The natural background source is generated similarly to previous work [4], except the energy resolution of the ground source tallies is increased to match the high resolution scheme, and these tallies are currents instead of fluxes. The transport from the ground to the detector is identical to the vehicle computation, except the source is a disk source on the ground. This is in contrast to the vehicle response function which assumes the radiation originates from a point. Therefore, the effective dimensions and area of the detector are a function of radial and azimuthal angle on the disk source. These effective dimensions are not utilized, and the nominal dimensions and area of the detector face are assumed. Furthermore, this assumes that the detector facets not facing the vehicle of interest are shielded from terrestrial background radiation such that the contribution is negligible compared to the total signal. The current at the detector face $\mathbf{q}_A^{\mathrm{bg}}$ [$\gamma \cdot \mathrm{s}^{-1} \cdot \mathrm{cm}^{-2}$] is

$$\mathbf{q}_A^{\mathrm{bg}} = \frac{1}{A_{\mathrm{det}}} \mathbf{R}^{\mathrm{bg}} \mathbf{I} \left(\mathbf{E}(\mathbf{q}^{\mathrm{terr}}) \rightarrow \mathbf{E}_i(\mathbf{R}^{\mathrm{bg}}) \right) A_{\mathrm{grnd}} \mathbf{q}_A^{\mathrm{terr}}, \tag{3.23}$$

where A_{det} is the area of the detector face in [cm^2], A_{grnd} is the area of the ground source in [cm^2], $\mathbf{R}^{\mathrm{bg}}$ is the energy transformation from photons on the ground surface to the detector face, $\mathbf{q}_A^{\mathrm{terr}}$ [$\gamma \cdot \mathrm{s}^{-1} \cdot \mathrm{cm}^{-2}$] is the current on the ground given by

$$\begin{aligned} \mathbf{q}_A^{\mathrm{terr}} = M_c (&\mathbf{q}_{A,k,c}^{\mathrm{terr}} \mathbf{q}_{k,c} + \mathbf{q}_{A,u,c}^{\mathrm{terr}} \mathbf{q}_{u,c} + \mathbf{q}_{A,th,c}^{\mathrm{terr}} \mathbf{q}_{th,c}) \\ &+ M_s (\mathbf{q}_{A,k,s}^{\mathrm{terr}} \mathbf{q}_{k,s} + \mathbf{q}_{A,u,s}^{\mathrm{terr}} \mathbf{q}_{u,s} + \mathbf{q}_{A,th,s}^{\mathrm{terr}} \mathbf{q}_{th,s}) \end{aligned} \tag{3.24}$$

and $\mathbf{q}_{A,k,c}^{\mathrm{terr}}$ is the current at the ground surface in [$\gamma \cdot \mathrm{s}^{-1} \cdot \mathrm{cm}^{-2} \cdot \mathrm{g}^{-1}$] from potassium-40 in the concrete, $\mathbf{q}_{A,u,c}^{\mathrm{terr}}$ is the current from the uranium series in the concrete [$\gamma \cdot \mathrm{s}^{-1} \cdot \mathrm{cm}^{-2} \cdot \mathrm{g}^{-1}$], $\mathbf{q}_{A,th,c}^{\mathrm{terr}}$ is the current from the thorium series in the concrete $[\gamma \cdot s^{-1} \cdot cm^{-2} \cdot g^{-1}]$, $\mathbf{q}^{\text{terr}}_{A,k,s}$ is the current from potassium-40 in the soil $[\gamma \cdot s^{-1} \cdot cm^{-2} \cdot g^{-1}]$, $\mathbf{q}^{\text{terr}}_{A,u,s}$ is the current from uranium in the soil $[\gamma \cdot s^{-1} \cdot cm^{-2} \cdot g^{-1}]$, $\mathbf{q}^{\text{terr}}_{A,th,s}$ is the current from thorium in the soil $[\gamma \cdot s^{-1} \cdot cm^{-2} \cdot g^{-1}]$, $\mathbf{q}_{k,c}$ and $\mathbf{q}_{k,s}$ are the specific activity of potassium-40 in the concrete and soil in $[\gamma \cdot s^{-1} \cdot g^{-1}]$, $\mathbf{q}_{u,c}$ and $\mathbf{q}_{u,s}$ are the specific activity of the uranium series in $[\gamma \cdot s^{-1} \cdot g^{-1}]$, $\mathbf{q}_{th,c}$ and $\mathbf{q}_{th,s}$ are the specific activity of the thorium series in $[\gamma \cdot s^{-1} \cdot g^{-1}]$, and M_c and M_s are the mass of the concrete and soil in [g].

3.1.7 Summary

Table 3.8 summarizes the response functions covered in the previous sections by listing the transformation type, the variables on which the response is dependent, and the section where each function is detailed.

Response	**Transform**	**Dependencies**	**Section**
$\mathbf{R}^{\text{snm}}$	$E \rightarrow E'$	m = SNM mass V/S = SNM vol. to surf. area ratio	3.1.2
$\mathbf{R}^{\text{shld}}$	$E \rightarrow E'$	t = shield thickness r_i/r_o = SNM/shield radius ratio $\mathbf{q}^{\text{snm}}$ = SNM spectrum f_Ω^{shld} = streaming fraction	3.1.3
$\mathbf{R}^{\text{car}}$	$E \rightarrow E'$	A_{det} = area of detector f_Ω^{car} = streaming fraction $\vec{d}$ = detector position $\vec{s}$ = source position	3.1.4
$\mathbf{R}^{\text{bg}}$	$E \rightarrow E'$	A_{det} = area of detector A_{grnd} = area of ground source	3.1.6
$\mathbf{R}^{\text{det}}$	$E \rightarrow E'$	t_x, t_y, t_z = dimensions of detector A_{eff} = effective area of detector $\vec{d}$ = detector position $\vec{s}$ = source position	3.1.5
$\mathbf{G}$	$E \rightarrow E'$	a, b, c = GEB parameters	3.1.5

Table 3.8: Summary of Photon Response Functions

3.2 Neutron Transport

For photons, sequentially applying Green's functions in the order that a photon would experience in its lifetime while ignoring albedo effects between submodels is sufficient. In addition, photons transport through the entire scenario almost instantaneously, thus removing any need for time binning. However, neutrons are quite different. They may exist for a measurable length of time and are much more diffusive which leads to significant albedo effects. Furthermore, subcritical multiplication in SNM may produce more neutrons either from an external source or by reflected source neutrons. Neutron cross-sections also contain many resonance peaks and are strongly isotope-dependent unlike photons which have identical cross-sections for any isotopes of an element. Fortunately, unlike photon detectors, neutron detectors can only offer limited energy resolution, removing the need for a large number of energy groups. Based on initial testing of various submodels, an 80 energy group structure logarithmically spaced from 0.1 meV to 20 MeV combined with a 100 group time structure logarithmically spaced from 1 ps (pico-second) to 1 s effectively covers the energy-time phase space. Neutrons binned within time less than 1 ps are considered to be instantaneous as a 20 MeV neutron travels approximately 6 μm in that time period and a thermal neutron travels 22 μm. The effect of this group structure on error is explored in more detail for each submodel. The full listing of these time and energy bins are in Appendix A.

3.2.1 Neutron Green's Functions

The basic methodology of decomposition as derived for photons remains largely unchanged. The treatment of time within each attribute adds another dimension to the transformation of radiation through each submodel. However there is only one dimension to time and it always moves forward. Therefore, by assuming the radiation enters the submodel at time zero, and tracking the time added to each source-tally energy pair, the amount of data grows linearly with the number of time bins unlike the energy bins. As an example of how this extra dimension is utilized within the Green's function paradigm, let $\mathbf{E} \in \mathbb{R}^M$ and $\mathbf{L} \in \mathbb{R}^Q$ be the energy and time bin structure for a source $\mathbf{s} \in \mathbb{R}^{(M-1)\times(Q-1)}$ and time-dependent response $\mathbf{K} \in \mathbb{R}^{(M-1)\times(M-1)\times(Q-1)}$. The transformed source $\mathbf{s}' \in \mathbb{R}^{M\times Q}$ is

$$\mathbf{s}'_{n,z} = \sum_{q=1}^{Q-1}\sum_{w=1}^{Q-1} f\left(\mathbf{L}_z, \mathbf{L}_{z+1}, \mathbf{L}_w + \mathbf{L}_q, \mathbf{L}_{w+1} + \mathbf{L}_{q+1}\right) \sum_{m=1}^{M-1} \mathbf{K}_{n,m,q}\mathbf{s}_{m,w} \quad \forall n \in M, \tag{3.25}$$

where $f(t_A, t_B, t_C, t_D)$ is the fraction of the bin $[t_C, t_D]$ which lies in bin $[t_A, t_B]$. Eq. 3.25 transforms all source energies which contribute to tally energy n in every time step of source $\mathbf{s}$, adding time specified by transformation $\mathbf{K}$. Because the time is additive, this produces a source with irregular time binning which does not necessarily match the desired time binning. The function f assumes a uniform distribution within each time bin, and divides the contribution based on the bin widths into the resultant spectrum $\mathbf{s}'$.

For brevity, the transformation detailed in Eq. 3.25 is shortened to

$$\mathbf{s}' = \mathbf{K}\mathbf{s}, \tag{3.26}$$

to remain consistent with the notation used previously. However, this is no longer a matrix-vector multiplication but a more abstract multi-dimensional transformation in both energy and time. Also carried over from previous notation is the mapping matrix $\mathbf{I}(\mathbf{E}_1 \rightarrow \mathbf{E}_2)$, which maps the energy bins of $\mathbf{E}_1$ to that of $\mathbf{E}_2$. There is no time mapping required.

To account for dependencies between submodels, the concept of an albedo Green's function is introduced. Unlike the previously described Green's functions which transport radiation through an attribute, an albedo Green's function is concerned with the reflection of radiation. In addition, these must be applied iteratively at each interface. For example, consider only the SNM and surrounding shielding. Let $\mathbf{s}$ [n·s^{-1}] be the source of neutrons from spontaneous fission and $\mathbf{K}^{\text{snm}}$ (unitless) be the SNM response function which is the probability, including multiplicity, that a neutron escapes the SNM. If $\mathbf{K}^{\text{snm}-}$ is the probability that neutrons incident on the SNM sphere and reflected back outward and $\mathbf{K}^{\text{shld}+}$ is the probability that neutrons incident on the inner surface of the shield that are reflected back inward, then the spectrum incident on the shield, $\mathbf{s}^{\text{snm}}$, is

$$\mathbf{s}^{\text{snm}} = \left[\mathbf{I} + \sum_{n=1}^{\infty} \left(\mathbf{K}^{\text{snm}+}\mathbf{K}^{\text{shld}-}\right)^n\right] \mathbf{K}^{\text{snm}}\mathbf{s}, \tag{3.27}$$

where $\mathbf{I}$ is the identity response function. The infinite sum represents all possible number of times a neutron may reflect between the SNM and shielding.

For any sub-critical problem, where neutrons have a diminishing probability of reflecting, then the series

$$\sum_{n=1}^{\infty}\left(\mathbf{K}^{\text{snm}+}\mathbf{K}^{\text{shld}-}\right)^{n} \tag{3.28}$$

converges. This summation may be reduced by the geometric series formula to

$$\sum_{n=1}^{\infty}\left(\mathbf{K}^{\text{snm}+}\mathbf{K}^{\text{shld}-}\right)^{n}=\left(\mathbf{I}-\mathbf{K}^{\text{snm}+}\mathbf{K}^{\text{shld}-}\right)^{-1} \tag{3.29}$$

In a computational implementation, the series is truncated when the n^{th} reflection cycle contributes a relatively negligible amount to the total. Here, an arbitrarily small value of $1{\times}10^{-10}$ is chosen as the cutoff point. If N reflection cycles are required to converge, the response functions are time-independent, and the dimensions of their matrix form is $M \times M$, then the left-hand-side of Eq. 3.29 requires $\propto N \times M^{2.8}$ floating point operations. Inverting the response function requires $\propto M^3$ operations. For $M = 80$, the break-even point is approximately two reflection cycles. Including time-dependence does not change the break-even point, as the matrix multiplication or inversion is simply repeated for each time step. However, the inclusion of an additional cutoff where portions of the phase-space less than $1{\times}10^{-20}$ relative to the total are ignored drastically decreases convergence time and increases the efficiency of the reflection cycle approach over the inversion method. This also provides a simple mechanism to check for super-critical systems, as the reflection series will diverge.

3.2.2 Special Nuclear Material

For photons, the cross-section in the SNM is independent of isotopic composition, and the difference between uranium and plutonium cross-sections are negligible. Resonances for each isotope of uranium and plutonium make this treatment from neutrons unattractive. Furthermore, there is no neutron saturation layer. Therefore, a more direct approach is taken to parameterize with respect to mass and isotopic composition.

Nominal concentrations are defined for a set of SNM types, such as HEU. For each SNM type, the energy-time dependent neutron current exiting a metal sphere of varying mass is simulated from a spontaneous fission source uniformly distributed within the sphere. The results are normalized to the number of spontaneous fissions. The nominal concentrations are modeled at each of the masses. In addition, small perturbations are made to the concentration of each isotope independently by adding mass δ_i to each i^{th} isotope. Let $\mathbf{Y} \in \mathbb{R}^I$ be the nominal isotopic mass vector (I total isotopes), $\mathbf{s}$ be the energy-time dependent current exiting the SNM from the nominal concentration [n·fission^{-1}], $\mathbf{s}_i$ be the current from the i^{th} perturbed isotope [n·fission^{-1}]. The current exiting the sphere $\mathbf{s}^{\text{iso}}$ [n·fission^{-1}] from isotopic mass vector $\mathbf{Y}'$ is given by

$$\mathbf{s}^{\text{iso}} = \mathbf{s} + \sum_{i=1}^{I} \frac{\mathbf{Y}'_i - \mathbf{Y}_i}{\delta_i} \left(\mathbf{s}_i - \mathbf{s}\right). \tag{3.30}$$

This equation linearly interpolates on isotopic concentrations to estimate their effect on the exiting current. After isotopic interpolation on the two nearest simulated masses, the current is linearly interpolated again between the total masses to compute $\mathbf{s}^{\text{mass}}$ [n·fission^{-1}]. This is scaled by the spontaneous fission rate in the SNM yielding the rate of neutrons exiting the SNM $\mathbf{s}^{\text{snm}}$ [n·s^{-1}],

$$\mathbf{s}^{\text{snm}} = \mathbf{s}^{\text{mass}} s_{\text{sf}}, \tag{3.31}$$

where s_{sf} is the rate of spontaneous fissions in [fission·s^{-1}].

To model the SNM albedo response function, it is necessary to understand the angular distribution at the surface of the SNM. This distribution is a function of energy and shielding material. As before, $p(\mu)d\mu = (n+1)\mu^n \, d\mu$ is the probability density function of the angular distribution as a function of the cosine from the surface normal μ. Therefore, the power of the cosine distribution n is indicative of the degree to which the radiation is forward directed. A 1 kg HEU sphere is modeled with a spontaneous fission source. Two shielding cases are considered, 10 cm of BPE and 10 cm of lead. The angular distribution is estimated as a function of energy for the current leaving the SNM (forward current) and the current re-entering the SNM (back current). The angular distribution is fit with the cosine power function to determine the cosine power n. These results as a function of energy are shown in Figure 3.30. Because the basis of this method is that each submodel is independent of another, the angular distribution re-entering the SNM must not depend on the shielding type. Therefore, as a simplified approximation, the data from both shielding types are equally weighted for the forward and back distribution and are fit with linear functions which are used to determine the angular distributions in the simulations. The functional form of the cosine power leaving the

SNM is

$$n(E) = 0.1201 \ln(E) + 1.707, \tag{3.32}$$

where E is the energy in [MeV]. The cosine power of neutrons reflected back into the SNM is

$$n(E) = -0.02447 \ln(E) + 0.6814. \tag{3.33}$$

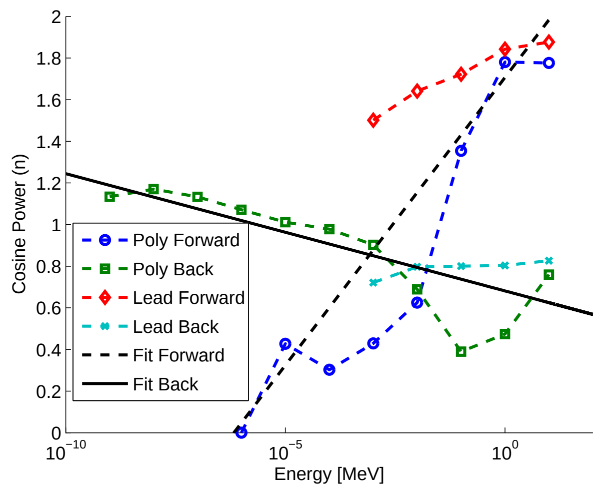


Figure 3.30: Cosine Power of Neutron Angular Distribution at Surface of SNM

To calculate the SNM reflection function, a source is placed on the surface of the SNM directed radially inward for each source energy. Eq. 3.33 is used to determine the angular distribution directed radially inward. Similarly to the spontaneous fission source, this is repeated for nominal isotopic concentrations as well as perturbed concentrations. Let $\mathbf{K}^+$ (unitless) be the reflection transformation for the nominal isotopic concentration and $\mathbf{K}_i^+$ (unitless) be the transformation for the perturbed concentration of the i^{th} isotope.

The interpolated transformation $\mathbf{K}^{\mathrm{iso}+}$ (unitless) is

$$\mathbf{K}^{\mathrm{iso}+} = \mathbf{K}^{+} + \sum_{i=1}^{I} \frac{\mathbf{Y}_i' - \mathbf{Y}_i}{\delta_i} \left(\mathbf{K}_i^{+} - \mathbf{K}^{+}\right). \tag{3.34}$$

As is done for the spontaneous fission source, $\mathbf{K}^{\mathrm{iso}+}$ is then linearly interpolated between masses to calculate $\mathbf{K}^{\mathrm{snm}+}$, the reflection response function for the SNM. The utilization of this function is described in the next section.

3.2.3 Shielding

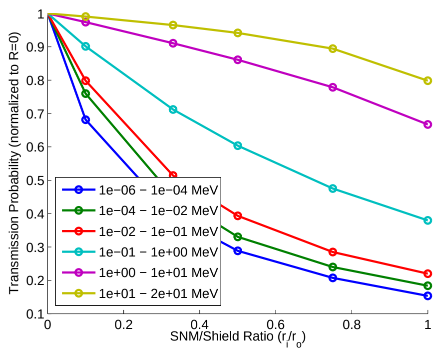
The shielding is a spherical shell of BPE or lead. Unlike the photon shield submodel, the neutron model must account for time gradients as well. Thus, the shielding layers are regular intervals and are not based on HVLs. The angular distribution entering the shielding as a function of energy is determined from Eq. 3.32.

The radius of the SNM affects the transmission probability through the shielding for neutrons in a similar manner to the photon shielding. In addition, albedo response functions require treatments for two additional probabilities. Let p be the total transmission probability through the shielding, p^- be the reflection probability at the inner surface of the shield, and p^+ be the reflection probability at the outer surface of the shield. To illustrate this effect, a 5 cm spherical shell of BPE around a perfectly absorbing sphere of variable radius is simulated. Let R be the ratio of the inner sphere radius r_i to the shield radius r_o. As before, the $R = 0$ condition is an isotropic point source in the center of a solid sphere of shielding. The $R = 1$ case is a planar source with a cosine angular distribution incident on a planar shield.

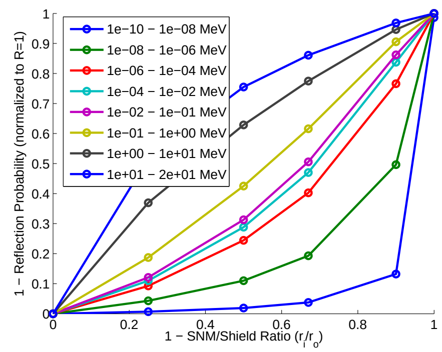
The transmission probability through the shield $p(R)$, normalized to the $R = 0$ result, is shown in Figure 3.31(a) for various source energies. The transmission probability decreases as R increases due to the apparent thickness of the shield increasing. This ratio also has an effect on the reflection probability, $p^{-}(R)$, shown in Figure 3.31(b) normalized to the $R = 1$ (planar source) result. In Figure 3.31(b), the data is illustrated as $1 - p^{-}(R)$ versus $1 - R$ for data fitting purposes. As expected, as the inner radius goes to zero $R = 0$, the probability that a neutron reflects back to that inner surface goes to zero. The normalized reflection probability on the outer surface of the shield is shown in Figure 3.31(c). As the inner radius increases, the probability of escape from the shield decreases as the probability of leakage into the perfectly absorbing inner sphere increases.

The convexity of the curves shown in Figures 3.31(a) 3.31(b) and 3.31(c) are a function of source energy, shielding material, and shielding thickness. To avoid explicitly modeling each ratio R, the convexity is captured in a single parameter as function of energy and shielding thickness for each material.

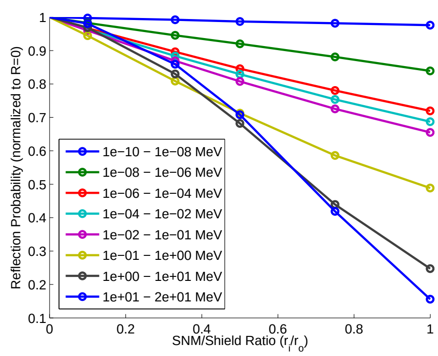
For the transmission data, a two step fitting is performed. An exponential function of the form $f(x) = \exp(bx)$ is fit to $f(x) = p$ and $x = R$ using only the normalized $R = 1$ data point to determine the b coefficient. A convexity coefficient c is then computed by fitting the entire data set to $f(x) = \exp(bx^c)$ using the previously determined b. With the c coefficient tabulated for source energies and shielding thicknesses, the shape of the transmission curve and



(a) Transmission



(b) Reflection (inner)



(c) Reflection (outer)

Figure 3.31: Normalized Transmission and Reflection Probabilities Through BPE

thus the total transmission probability may be determined for any ratio. The transmission geometry scaling factor $S_R(R, E, t)$ for ratio R, source energy E and shielding thickness t is

$$S_R(R, E, t) = \frac{p(R = 0)}{p(R = 1)} \exp\left(\ln\left(\frac{p(R = 1)}{p(R = 0)}\right) R^{c(E,t)}\right). \tag{3.35}$$

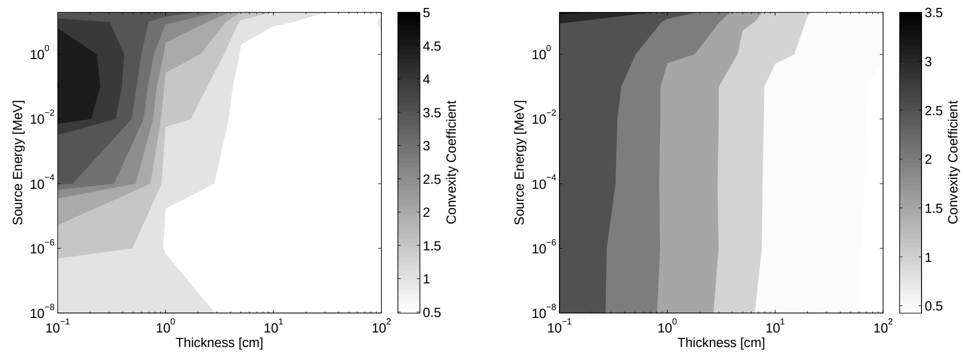
For the normalized reflection probability on the inner surface of the shield, the probability is always zero at $R = 0$ and one at $R = 1$. By fitting a function of the form $f(x) = x^{c^-}$ to $f(x) = 1 - p^-$ and $x = 1 - R$, a convexity coefficient may be determined. This c^- coefficient is used to determine the reflection probability relative to the planar source case. The geometry scaling factor for inner reflection is

$$S_R^-(R, E, t) = 1 - (1 - R)^{c^-(E,t)}. \tag{3.36}$$

The normalized reflection probability on the outer surface of the shield is fit with two-stage exponential fit as is done for the transmission probability to determine the c^+ convexity coefficient. The geometry scaling factor for outer reflection is

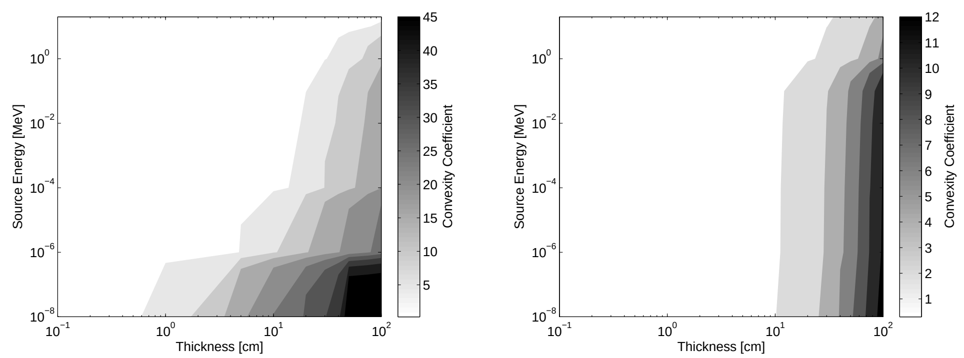
$$S_R^+(R, E, t) = \frac{p(R = 0)}{p(R = 1)} \exp\left(\ln\left(\frac{p^+(R = 1)}{p^+(R = 0)}\right) R^{c^+(E,t)}\right). \tag{3.37}$$

Varying the shielding thickness from 0.1 cm to 100 cm, and source energy from 1×10^{-10} MeV to 20 MeV, the transmission convexity coefficients are shown in Figure 3.32 for BPE and lead. The reflection convexity coefficients are shown in Figure 3.33 and 3.34. If these convexity values are used to estimate the values shown in Figures 3.31, the results are shown as dashed lines in Figure 3.35. The difference between the estimated and simulated probabilities is less than 10%.



(a) BPE (b) Lead

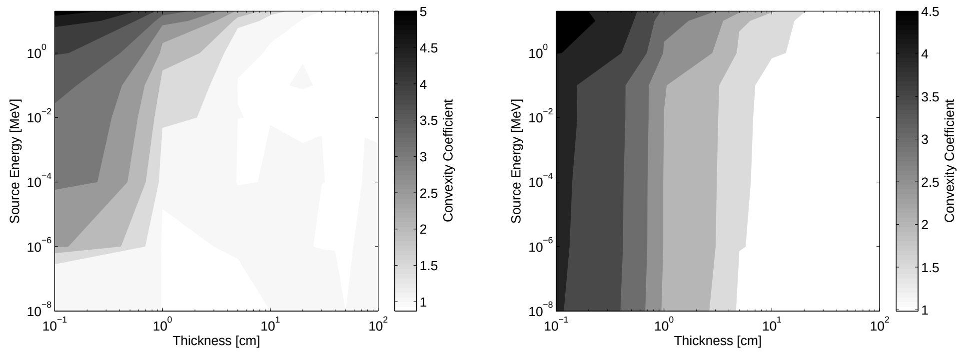
Figure 3.32: Convexity Coefficient for Transmission Probability



(a) BPE (b) Lead

Figure 3.33: Convexity Coefficient for Inner Reflection Probability

For photons, the simulation data from a point source in the center of a sphere of shielding material is sufficient to describe the radiation transport; neutrons require multiple simulations: (a) the planar source (b) reflection from



(a) BPE

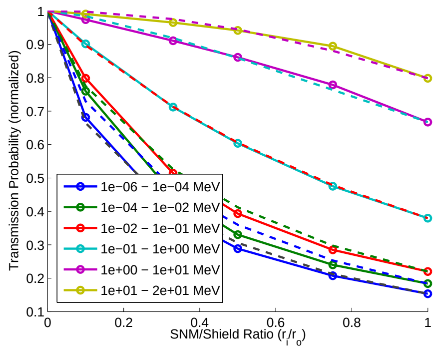
(b) Lead

Figure 3.34: Convexity Coefficient for Outer Reflection Probability

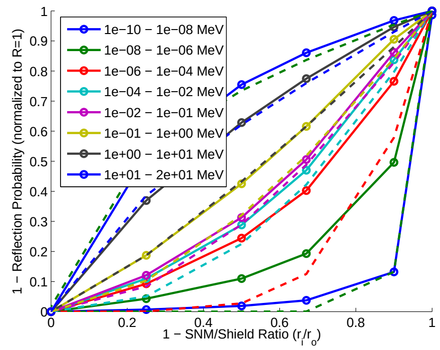
a planar source, (c) a point source in radial geometry to scale the planar source results, and (d) radial geometry reflection to scale the planar reflection. These four simulations are illustrated in Figure 3.36.

The full time and spectral transformation is simulated with the planar sources. The radial simulations only compute the total transmission and reflection as a function of source energy. Therefore, transmission and reflection scaling is performed without regard to energy or time bins. That is, the effect of the ratio R on energy and time transformation is neglected. The response functions $\mathbf{K}^{\text{shld}}$, $\mathbf{K}^{\text{shld}-}$, and $\mathbf{K}^{\text{shld}+}$ are generated at each shielding thickness interval for a planar source, and scaled by $S_R(R, E, t)$, $S_R^-(R, E, t)$, and $S_R^+(R, E, t)$, respectively. After scaling, the current exiting the shield $\mathbf{s}^{\text{shld}}$ [n·s^{-1}], without albedo effects, is

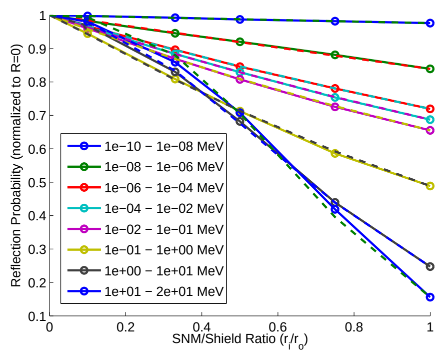
$$\begin{aligned}\mathbf{s}^{\text{shld}} = (1 - f_\Omega^{\text{shld}})\mathbf{K}^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}^{\text{snm}}) \to \mathbf{E}_i(\mathbf{K}^{\text{shld}})\right)\mathbf{s}^{\text{snm}} \\ + f_\Omega^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}^{\text{snm}}) \to \mathbf{E}(\mathbf{s}^{\text{shld}})\right)\mathbf{T}(t)\mathbf{s}^{\text{snm}},\end{aligned} \tag{3.38}$$



(a) Transmission

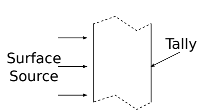


(b) Reflection (inner)

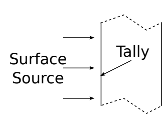


(c) Reflection (outer)

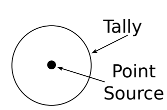
Figure 3.35: Comparison of Estimated Transmission and Reflection Probabilities to Simulated Data



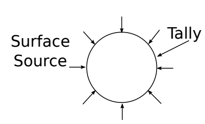
(a) Planar



(b) Planar Reflection



(c) Radial



(d) Radial Reflection

Figure 3.36: Simulated Shielding Geometries

where f_Ω^{shld} is the shielding streaming fraction, and $\mathbf{T}(t)$ adds time for each energy bin equal to $t/\bar{v}(E)$ where $\bar{v}(E)$ is the average velocity in that energy bin in [cm·s^{-1}] and t is the shielding thickness in [cm]. With reflection considered, the current exiting the shielding is

$$\mathbf{s}^{\text{shld}} = \Bigg[(1 - f_\Omega^{\text{shld}})\mathbf{K}^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}^{\text{snm}}) \to \mathbf{E}_i(\mathbf{K}^{\text{shld}})\right) + f_\Omega^{\text{shld}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}^{\text{snm}}) \to \mathbf{E}(\mathbf{s}^{\text{shld}})\right)\mathbf{T}(t)\Bigg]\left[\mathbf{I} + \sum_{n=1}^{\infty}\left(\mathbf{K}^{\text{snm}+}\mathbf{K}^{\text{shld}-}\right)^n\right]\mathbf{s}^{\text{snm}}. \tag{3.39}$$

3.2.4 Vehicle

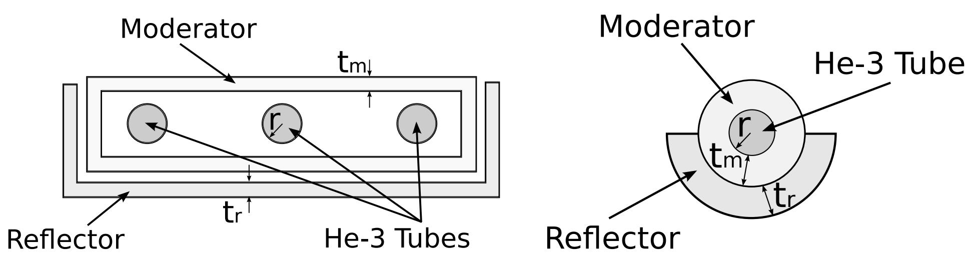
The vehicle formulation is identical to that of the photon models, except that the energy structure is different, time dependence is included, and neutrons are tracked instead of photons. The neutron current at multiple detector panels along the side and top of the truck is simulated for multiple source positions within the cargo. These currents are interpolated based on source and detector position as described in Section 3.1.4 and combined to produce the transformation $\mathbf{K}^{car}$. This transformation is applied to the shielding source to compute the current density at the detector face, $\mathbf{s}_A^{\text{car}}$ [n·s^{-1}·cm^{-2}],

$$\mathbf{s}_A^{\text{car}} = \frac{1}{A_{\text{det}}}\mathbf{K}^{\text{car}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}^{\text{shld}}) \to \mathbf{E}_i(\mathbf{K}^{\text{car}})\right)\mathbf{s}^{\text{shld}}. \tag{3.40}$$

3.2.5 Detector

Two neutron detector types are considered: helium-3 and solid-state. Helium-3 detectors are cylindrical tubes filled with pressurized helium-3 gas.

These are usually placed in an array and surrounded with moderating material to thermalize fast neutrons. A reflector is placed behind the tube to decrease leakage from the back side of the detector and to shield against background radiation [18]. This configuration is shown in Figure 3.37(a). The moderator and reflector have a significant impact on the efficiency of the detector. To avoid simulating this detector setup for multiple source angles, an approximation is made which converts the moderator and reflector into cylindrical shells around the helium-3 tube as shown in Figure 3.37(b). Two source configurations are considered. The first is assumed to be a beam directed on the face of the detector, used to compute detector responses to sources within the vehicle. The second is a surface source with a cosine distributions on each face of the detector to determine background radiation responses. To account for varying configurations, the length of the helium-3 tube is varied along with the thickness of the moderator and the thickness of the reflector.



(a) Realistic Setup (b) Approximation

Figure 3.37: Top-Down View of Helium-3 Detector Geometric Configurations

Helium-3 detectors offer almost no energy resolution. Therefore, the transformation $\mathbf{K}^{\text{det}}$ transforms time-dependent energy spectra to time-dependent counts. The transformations are linearly interpolated between tube length, moderator thickness, and reflector thickness to produce $\mathbf{K}^{\text{det}}$ [counts·n^{-1}·s^{-1}]. The detector signal $\mathbf{d}^{\text{det}}$ [counts·s^{-1}] is

$$\mathbf{d}^{\text{det}} = A\mathbf{K}^{\text{det}}\mathbf{I}\left(\mathbf{E}(\mathbf{s}_A^{\text{car}}) \to \mathbf{E}_i(\mathbf{K}^{\text{det}})\right)\mathbf{s}_A^{\text{car}}, \tag{3.41}$$

where A is the area of the detector face in [cm^2].

A common solid state neutron detector is a semi-conducting silicon wafer in a planar geometry with boron enriched in the boron-10 isotope [17]. Because the neutron conversion volume (boron) is separated from the detection volume (silicon) in this geometry, it is not as efficient as pillars or fins. However, these advanced geometries are limited by manufacturing techniques and are not commercialized. Thus, only the planar configuration is considered here. To account for varying degrees moderation, the face of the detector is surrounded with varying thicknesses of a moderating slab. This geometry is illustrated in Figure 3.38.

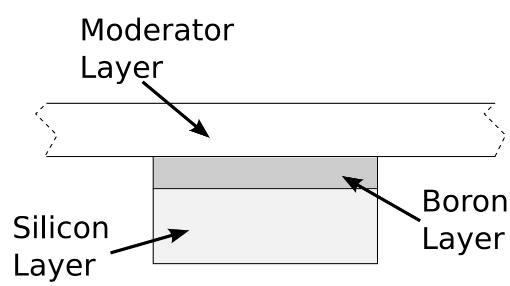


Figure 3.38: Solid State Detector Geometry

Unlike helium-3 tubes, silicon semi-conductors can achieve some level of energy resolution based on the energy of the recoiling nuclei. Therefore, the response function for the solid state detector transforms in both energy and time. Aside from this aspect, Eq. 3.41 still holds in computing the signal from the detector.

3.2.6 Background Radiation

The cosmic-ray induced neutron background source is determined by utilizing an experimental spectrum obtained by Paul Goldhagen et. al. [42] [43], and scaling it by the appropriate factors to get the correct intensity. A functional form of the spectrum and the scaling factor formula is given by the JEDEC standard JESD89A [44]. The spectrum above 1 MeV to within 3.3% of the experimental data can be approximated with

$$\begin{aligned} \phi_0(E) &= 1.006 \times 10^{-6} \exp\left(-0.35 \ln(E)^2 + 2.1451 \ln(E)\right) \\ &+ 1.011 \times 10^{-3} \exp\left(-0.4106 \ln(E)^2 + 0.667 \ln(E)\right), \end{aligned} \quad (3.42)$$

where $\phi_0(E)$ is the energy-dependent flux [n·cm^{-2}·s^{-1}·MeV^{-1}] at energy E [MeV]. The experimental data set with lower energy limit 1×10^{-10} is illustrated in Figure 3.39. The flux for any geographic location is computed with

$$\phi(E) = \phi_0(E) F_A(d) F_B(R_c, I, d), \quad (3.43)$$

where $\phi(E)$ is the differential flux after scaling [n·cm^{-2}·s^{-1}·MeV^{-1}], d is the atmospheric depth [g·cm^{-2}], R_c is the vertical geomagnetic cutoff rigidity [GV] (particle momentum per unit charge), I is the relative neutron count rate from solar modulation, F_A is a function which scales the flux based on altitude dependence, and F_B is a function which scales the flux based on altitude, geomagnetic location, and solar modulation. The details of this formula and how to calculate its values is described in detail in the JESD89A report in Annex A. The scaling factors may be computed based on elevation, a solar modulation factor, and longitude and latitude. The solar modulation factor is a number between 0 and 1 which represents the degree of solar activity, with 0 being the solar minimum and 1 being the maximum. The cutoff rigidity is interpolated between data taken from reference [45], which tabulates rigidity as a function of longitude and latitude. This data is illustrated in Figure 3.40.

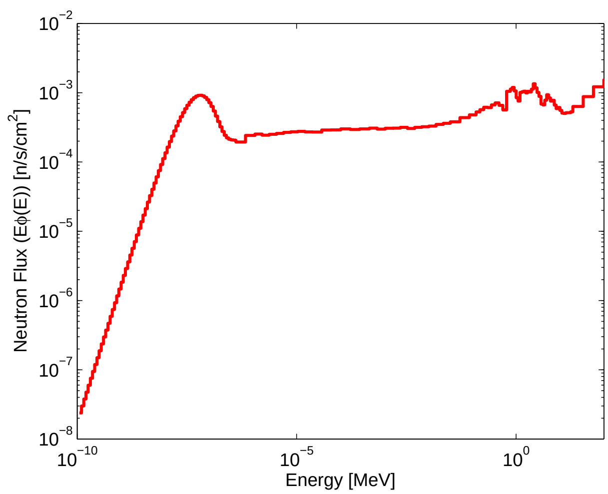


Figure 3.39: Background Neutron Flux

The angular distribution of the cosmic-ray induced neutron flux varies with neutron energy, with higher energy neutrons being directed more downward. Because no angular information is given, a uniform flux is assumed. A background source $\mathbf{b}_0^{\text{bg}} \in \mathbb{R}^{M-1}$ with an arbitrary energy binning $\mathbf{E} \in \mathbb{R}^M$ can

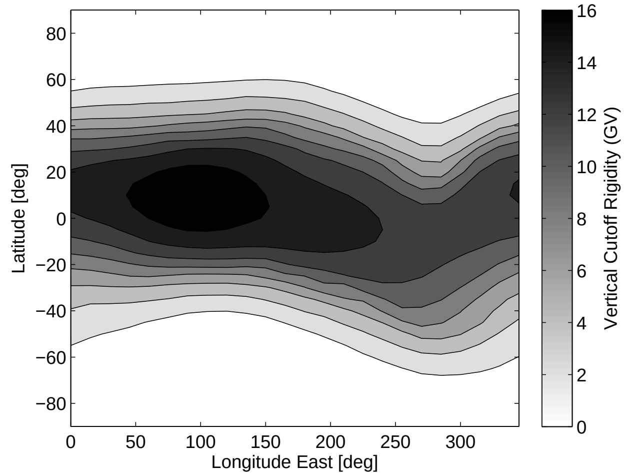


Figure 3.40: Geomagnetic Vertical Cutoff Rigidity Map

be approximated from the differential flux by

$$\begin{aligned} \mathbf{b}_{0,m}^{\mathrm{bg}} &= \int_{\mathbf{E}_m}^{\mathbf{E}_{m+1}} \frac{1}{v(E)} \phi(E) dE \\ &\approx \frac{\Delta \mathbf{E}_m}{2} \left(\frac{\phi(\mathbf{E}_m)}{v(\mathbf{E}_m)} + \frac{\phi(\mathbf{E}_{m+1})}{v(\mathbf{E}_{m+1})} \right), \end{aligned} \tag{3.44}$$

where $\Delta \mathbf{E}_m = \mathbf{E}_{m+1} - \mathbf{E}_m$ is the width of energy bin m in [MeV], and $v(E)$ is the velocity of a neutron in [cm/s] of energy E. The spectrum at the detector interface is computed by inserting this spectrum as a uniform volumetric source around an array of detector locations.

An array of detector boxes similar to the vehicle detector array is constructed. Each of these boxes tallies the energy dependent current entering the box on each of the six faces $\mathbf{b}_{A,x+}$, $\mathbf{b}_{A,x-}$, $\mathbf{b}_{A,y+}$, $\mathbf{b}_{A,y-}$, $\mathbf{b}_{A,z+}$, $\mathbf{b}_{A,z-}$ [n$\cdot$cm^{-2}]. This array of detectors is placed around the vehicle, and in the same positions without the vehicle. These spectra are used in the neutron detector response functions to determine the appropriate detector signal.

3.2.7 Summary

Table 3.9 summarizes the neutron response functions described here. Most responses transform from all energies E to E' as well as tracking time t'. The exceptions to this are the SNM response function, which transforms from the number of spontaneous fissions to a resultant energy E', and the helium-3 detector, which transforms incoming energies E to integral counts.

Response	Transform	Dependencies	Section
$\mathbf{K}^{\text{snm}}$	# spont. fiss$\rightarrow E'$, $0 \rightarrow t'$	m = SNM mass $\mathbf{Y}$ = isotopic vector	3.2.2
$\mathbf{K}^{\text{snm+}}$	$E \rightarrow E'$, $0 \rightarrow t'$	m = SNM mass $\mathbf{Y}$ = isotopic vector	3.2.2
$\mathbf{K}^{\text{shld}}$	$E \rightarrow E'$, $0 \rightarrow t'$	t = shield thickness r_i/r_o = SNM/shield radius ratio c = convexity coefficient $p(R=0)$ = point source probability f_Ω^{shld} = streaming fraction	3.2.3
$\mathbf{K}^{\text{shld-}}$	$E \rightarrow E'$, $0 \rightarrow t'$	t = shield thickness r_i/r_o = SNM/shield radius ratio c^- = convexity coefficient	3.2.3
$\mathbf{K}^{\text{shld+}}$	$E \rightarrow E'$, $0 \rightarrow t'$	t = shield thickness r_i/r_o = SNM/shield radius ratio c^+ = convexity coefficient $p^+(R=0)$ = point source probability	3.2.3
$\mathbf{K}^{\text{car}}$	$E \rightarrow E'$, $0 \rightarrow t'$	A_{det} = area of detector f_Ω^{car} = streaming fraction $\vec{d}$ = detector position $\vec{s}$ = source position	3.2.4
$\mathbf{K}^{\text{det}}$ (He-3)	$E \rightarrow$counts, $0 \rightarrow t'$	A = area of detector h = helium tube height r_m = moderator radius r_f = reflector radius	3.2.5
$\mathbf{K}^{\text{det}}$ (B/Si)	$E \rightarrow E'$, $0 \rightarrow t'$	A = area of detector t_m = moderator thickness	3.2.5

Table 3.9: Summary of Neutron Response Functions

Chapter 4

Implementation

The methods discussed here assume the use of the radiation transport package MCNPX [25]. However, any software package capable of providing the data required for the response functions is applicable. The following sections discuss the model details used in MCNPX to generate the data. Representative MCNPX input files are listed in Appendix A. MCNPX simulation results are considered to be converged when the relative error in the total tally estimate is less than 1% and the variance of the variance is less than 10%. The management and utilization of the data to simulate threat scenarios is performed by the code eXpedited Parametric Analysis of Smuggling Scenarios (XPASS), the capabilities of which are summarized.

4.1 Radiation Transport Models

4.1.1 Photon

Because the photon transport for each submodel is assumed to be independent of all other submodels, each MCNPX model focuses on a single submodel. If tally tagging is possible, it is utilized to increase the accuracy of the spectral decomposition. It is possible to sample all 67 source energies in a single input deck by using a special tally treatment which flags the tally result by source energy. However, it may not be used in conjunction with tally tagging, and variance reduction on such a large range of source energies is difficult. Therefore, all models are generated with monoenergetic sources, with one input deck for each.

Except for the detector models, only photons are explicitly tracked. To account for bremsstrahlung from electrons, MCNPX employs the thick-target bremsstrahlung model. This model assumes electrons ejected from atoms are slowed to rest at the point of creation, and any bremsstrahlung radiation is subsequently transported from that point. This is a valid approximation for low energy electrons as their range in matter is negligible. For example, using the continuous slowing down approximation, the range of a few select electron energies are summarized in Table 4.1. For high-Z materials such as lead, their range is negligible for most energies. For low-Z or hydrogeneous materials, the range becomes appreciable beyond a few MeV. Thus, with passive detection in which the maximum photon energy is less than 4 MeV, using the thick-target bremsstrahlung model is a good approximation. However, with active interrogation where photons and electrons with energies greater than 10 MeV may be produced, this may not be acceptable, especially near submodel interfaces.

4.1.1.1 Special Nuclear Material

The SNM model is a 1 kg sphere placed in a vacuum. The material is uranium at a density of 18.95 g/cm^3, therefore the radius is 2.327 cm. The

Energy	**Range in Water** [cm]	**Range in Lead** [cm]
100 keV	0.014	0.0027
1 MeV	0.44	0.0070
10 MeV	5.0	0.54
100 MeV	33	1.7

Table 4.1: Electron Ranges in Water and Lead

source is sampled uniformly within the sphere with probability proportional to r^2; however, to reduce the variance the radial distribution is biased by sampling from a distribution proportional to r^5, which is chosen arbitrarily. This biasing starts more histories toward the outer surface of the SNM based on the assumption that they are more important to the problem than those started closer to the center. Figure 4.1 summarizes this setup.

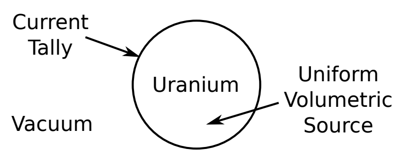


Figure 4.1: SNM Photon MCNPX Model

The simulation for the SNM is done in two stages. The first stage uses the energy-dependent weight window generator to estimate the importance function in space and energy. The windows are generated with respect to a current tally on the surface of the sphere. The second stage utilizes these weight windows in conjunction with an exponential transform. The exponential transform samples the distance to next collision based on an artificially reduced total cross section, and reduces the photon weight appropriately. The degree to which the cross section is reduced is based on the cosine of the angle between the photon flight path and the radial vector of the sphere, thus allowing photons travelling toward the surface of the sphere to traverse greater thicknesses with fewer collisions. This decreases the variance on the estimate of uncollided photons, especially for the lower source energies where the attenuation coefficient in the SNM is large. The energy dependent current on the surface of the SNM is tallied. Because it is surrounded by vacuum, this current only includes radiation leaving the sphere.

A realistic SNM sphere is not composed of pure uranium. Aged uranium contains many daughter isotopes. However, their concentrations are small, and the effect on the macroscopic cross-section is assumed to be negligible. In addition, plutonium is also simulated based on the uranium material. The total cross-section for uranium and plutonium differ by 10% in the l^2-norm and is assumed to be a negligible factor. The density used must only be large enough to achieve the saturation layer for that mass of the SNM, and therefore is somewhat arbitrary. The actual density of the SNM in the threat scenario is still variable as with the mass it determines the volume to surface area ratio used to scale these results.

4.1.1.2 Shielding

The shielding model consists of 100 layered spherical shells, each with a thickness equal to the uncollided half-value-layers (HVL) for the source energy. A monoenergetic point source is placed at the center of a sphere of the shielding material. The energy dependent current is tallied at each HVL. Because beyond each HVL there should be a vacuum boundary condition if the shield physically terminates there, photons are flagged as they cross each HVL, and the flagged contribution is subtracted from the total. The materials simulated are lead and BPE, with density 11.36 g/cm^3 and 1.03 g/cm^3, respectively.

4.1.1.3 Vehicle

The vehicle simulated is a standard US commercial tractor truck trailer. It consists of a cargo hold 102 inches wide, 102 inches high, and 53 feet long. The geometric details of the cargo hold are beyond the scope of this report but are listed in reference [46]. However, the level of detail includes supporting steel I-beams, suspension, wheels, and axles. The tractor is simplified to large homogeneous blocks of steel and air, except for the fuel tanks, engine block, and wheels, all of which matches the total gross weight of a realistic tractor (about 17,000 pounds). Two MCNPX geometry screenshots of the model are labeled with major features in Figure 4.2.

The material inside the cargo hold is comprised of four different materials: a vacuum, and homogenized low-Z, mid-Z, and high-Z materials. The low-Z material is wood, the mid-Z material is tile, and the high-Z material is steel. The composition of these materials are listed in Appendix A. These materials are representative of common materials transported within the US. These materials completely fill the cargo hold and are set at a constant density of 0.2 g/cm^3, making the weight of the cargo 50,000 pounds.

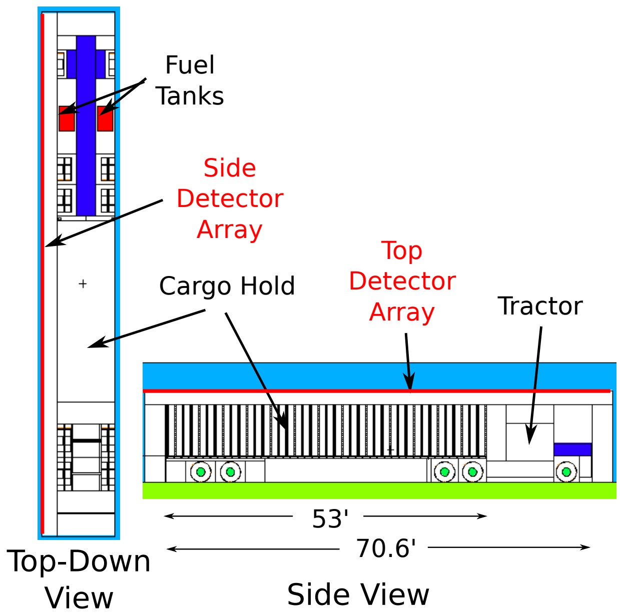


Figure 4.2: Truck Photon MCNPX Model

An array of 20×6 detector panels spans the length and height of the vehicle on the side at a standoff distance of 68.56 cm. An array of 20×4 detectors are placed above half the truck at the same standoff distance. Only half the top is covered by detectors because there is symmetry along the length of the truck. An energy dependent current tally is placed on each detector panel along with two cosine bins to track which photons are coming from the truck and those which reflect back toward the truck. There are 90 monoenergetic point sources evenly distributed throughout the cargo, each sampled with equal probability. Each current tally is flagged based on which point source location is sampled. The resulting currents are scaled by the number of source points to give the correct result for a single point source.

The source points within the cargo are averaged to produce a response function for a distributed source. This allows NORM cargos to be simulated in addition to point sources. While this allows the NORM source to placed within the cargo, the NORM does not affect the radiation transport. However, an appropriate combination of the low-Z, mid-Z, and high-Z cargo types provides an approximation to the actual NORM. A listing of NORM and their radioactive constituents are listed in Appendix A.

4.1.1.4 Detector

The detector models are rectangular prisms of the detector material surrounded by vacuum. The materials used are PVT, NaI, and HPGe. An energy dependent pulse-height tally is placed in the detector volume. A beam of monoenergetic photons is evenly distributed across the detector face. This setup is illustrated in Figure 4.3. The dimensions t_x,t_y,t_z are each set to 1 cm, 5 cm, 20 cm, 50 cm, 100 cm, and 200 cm, resulting in 216 different detector sizes. Each detector size is an independent volume in the same input deck and each face is sampled with uniform probability. Therefore, the results must be scaled by the number of detectors to normalize the data properly. The MCNPX Gaussian energy broadening feature is not used. Any broadening is done by post-processing the simulated detector signal. Because MCNPX broadens the signal based on continuous photon energy transport, post-process broadening introduces error by assuming uniform intensity over each detector channel. However, because the energy bin widths are on the order of 1 keV, and the

FWHM is a smooth function of energy (see Eq. 3.20), this error is expected to be minimal.

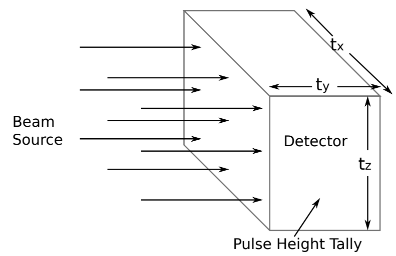


Figure 4.3: Detector Photon MCNPX Model

4.1.1.5 Terrestrial Background

The terrestrial background consists of two response functions. The first transports radiation from the soil and concrete volumes to the surface of the concrete, and the second transports radiation from the concrete surface to the detectors. The first input file is composed of a cylinder of soil with radius 2 meters and thickness 2 meters with a cylinder of concrete 2 meters in radius and 30 cm thick. The radial edge of the geometry is made reflecting to simulate an infinite disk. This geometry is illustrated in Figure 4.4. A uniform volumetric source of photons with spectrum for the uranium series, thorium series, and potassium-40 are placed in the soil and concrete separately. The energy dependent current is tallied at the top surface of the concrete. Six spectra are generated (one for each series in each volume) and are weighted and summed based on the local terrestrial radionuclide concentration.

The second input file which transports radiation from the concrete sur-

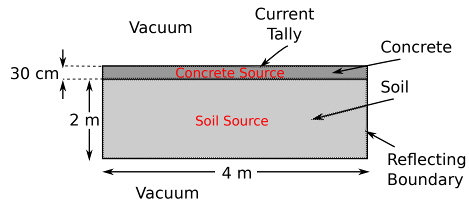


Figure 4.4: Terrestrial Photon MCNPX Model

face to the detector faces consists of two versions. The first version is identical to the input deck for the truck except the source is a surface disk source on the ground. This simulates the detectors' signal to terrestrial radiation in the truck's presence to account for baseline suppression. The truck is absent in the second version to simulate the nominal background signal without any vehicle present.

4.1.2 Neutron

Each neutron MCNPX input deck is simulated with a source evenly distributed in a single energy bin at time zero. The exception to this is the SNM model, which uses a spontaneous fission source. This is in contrast to the photon models which simulated a series of monoenergetic sources. While no source energy interpolation is required, the various simulations must be combined in post-processing to generate the full response function.

4.1.2.1 Special Nuclear Material

Nominal concentrations are defined for HEU, DU, WGPu, and RGPu in Table 4.2 [47]. For the uranium types, the energy-time dependent neutron current exiting a metal sphere with masses 100 g, 1 kg, 5 kg, 10 kg, and 20 kg is simulated from a spontaneous fission source uniformly distributed the sphere. For the plutonium types, 100 g, 1 kg, 4 kg, and 7 kg are modeled. The results are normalized to the number of spontaneous fissions; therefore, to calculate the number of neutrons leaving the SNM, they are scaled by the spontaneous fission rate determined from Table 4.3. These masses are simulated at the nominal concentrations described in Table 4.2, as well as the perturbed concentration for each isotope listed in Tables 4.4, 4.5, 4.6, and 4.7. Perturbations are made by adding additional mass of a single isotope, leaving the individual masses of the remaining isotopes fixed. In addition to the neutron current, the energy-dependent photon current exiting the sphere from prompt fission gammas is also tallied. Alternatively, this formulation may be replicated using the MCNPX perturbation feature instead of explicitly changing the isotopic compositions. This allows first and second order perturbation of SNM isotopics, potentially saving computation time; however, this alternative is not explored in this dissertation.

Because the neutron energy spectrum for all radionuclides is sampled from a Watt fission spectrum [25], the perturbations to each isotope have little effect on the overall shape of the neutron energy spectrum exiting the sphere. The primary effect of these perturbations is on the multiplication factor of the sphere; therefore, low-concentration radionuclides such as ^{232}U and ^{236}Pu do not require perturbations as they have little effect on either the shape or the multiplication factor. However, this does not preclude them from the threat scenario simulation, as only the spontaneous fission rate density is required to scale the simulated current exiting the sphere.

SNM	WGPu	RGPu	HEU	VHEU	DU
Density [g/cm^3]	15.75	15.75	18.95	18.95	18.95
Isotopic Composition [w/o]	^{236}Pu 5×10^{-9}	^{236}Pu 3×10^{-8}	^{232}U 3×10^{-8}		
	^{238}Pu 0.015	^{238}Pu 1.2	^{234}U 0.70	^{234}U 0.70	^{234}U 0.001
	^{239}Pu 93.63	^{239}Pu 59.0	^{235}U 90.3	^{235}U 90.3	^{235}U 0.2
	^{240}Pu 6.0	^{240}Pu 24.0	^{236}U 0.3	^{236}U 0.3	
	^{241}Pu 0.355	^{241}Pu 11.8	^{238}U 8.7	^{238}U 8.7	^{238}U 99.799
		^{242}Pu 4.0			

Table 4.2: SNM Isotopics and Density

	Specific Activity [s^{-1}g^{-1}]	**Spontaneous Fission Yield** [%]	**Spontaneous Fission Activity** [fissions/s/g]
^{232}U	8.28×10^{11}	0	0
^{234}U	2.30×10^{8}	1.7×10^{-9}	0.39
^{235}U	8.00×10^{4}	7.0×10^{-9}	5.6×10^{-4}
^{236}U	2.39×10^{6}	9.6×10^{-9}	0.023
^{238}U	1.24×10^{4}	5.0×10^{-5}	0.62
^{236}Pu	1.96×10^{13}	1.4×10^{-7}	2.7×10^{6}
^{238}Pu	6.34×10^{11}	1.9×10^{-7}	1.2×10^{5}
^{239}Pu	2.30×10^{9}	3.0×10^{-10}	0.69
^{240}Pu	8.40×10^{9}	5.7×10^{-6}	4.8×10^{4}
^{241}Pu	3.83×10^{12}	2.0×10^{-14}	0.077
^{242}Pu	1.47×10^{8}	5.5×10^{-4}	8.1×10^{4}

Table 4.3: SNM Radionuclide Spontaneous Fission Source

4.1.2.2 Shielding

The neutron shielding model considers a planar geometry and point source radial geometry for BPE and lead. The total HVL for BPE and lead can be as high as 10 cm for certain energies. This distance can be unacceptably large for capturing the gradient in time. In addition, unphysically large

	$\delta-$ [%]	**Nominal** [%]	$\delta+$ [%]
^{232}U	n/a	3×10^{-8}	n/a
^{234}U	0.07	0.7	7.0
^{235}U	80	90.3	99
^{236}U	0.03	0.3	3.0
^{238}U	0.87	8.7	20

Table 4.4: HEU Perturbed Isotopics

	$\delta-$ [%]	**Nominal** [%]	$\delta+$ [%]
^{234}U	0.0001	0.001	0.01
^{235}U	0.02	0.2	1.0
^{238}U	90	99.799	99.9999

Table 4.5: DU Perturbed Isotopics

	$\delta-$ [%]	**Nominal** [%]	$\delta+$ [%]
^{236}Pu	n/a	5×10^{-9}	n/a
^{238}Pu	0.001	0.015	0.1
^{239}Pu	80	93.63	99
^{240}Pu	1	6.0	10
^{241}Pu	0.1	0.355	10

Table 4.6: WGPu Perturbed Isotopics

	$\delta-$ [%]	**Nominal** [%]	$\delta+$ [%]
^{236}Pu	n/a	3×10^{-8}	n/a
^{238}Pu	0.1	1.2	10
^{239}Pu	50	59	70
^{240}Pu	15	24	35
^{241}Pu	1	11.8	20
^{242}Pu	0.4	4.0	14

Table 4.7: RGPu Perturbed Isotopics

shielding types can be constructed out of 100 HVL layers. Therefore, based on the physical constraints of the problem and to account for time gradients, the maximum shielding radius is set at 50 cm for all shielding materials, which results in a maximum 1 m diameter threat object. This 50 cm radius is simulated at 100 intervals, or 0.5 cm linearly spaced layers for the planar geometry and 0.5 cm radially spaced layers for the point source geometry. Analogously to the photon model, the current across multiple layers of shielding is simulated, and neutrons are flagged as they traverse each layer. The flagged contribution is subtracted from the total current to calculate the current at a vacuum boundary condition. Reflection shielding simulations tally the current at the source surface, and subtract the flagged contributions at each layer from the total reflection current.

4.1.2.3 Detector

The helium-3 detector model is a 1.96-inch diameter tube filled with helium-3 gas at 300 kPa. The tube height is varied at 10 cm, 50 cm, 100 cm, and 200 cm. To act as a moderator, polyethylene surrounds the helium-3 tube with radial thicknesses 0 cm, 1 cm, 5 cm, and 10 cm. The polyethylene is partially surrounded radially by iron with radial thicknesses 0 cm, 1 cm, 3 cm, and 5 cm which acts as a reflector. The source is a beam uniformly distributed across the cross-sectional area of the detector. A flux tally is placed within the helium-3 volume, and is multiplied by the $^{3}He(n,p)^{3}H$ cross-section to calculate the reaction rate. Each reaction event is assumed to be a count in the detector and any wall effects produced by the recoiling ^{3}H or ^{1}H ions is disregarded.

The solid state neutron detector consists of a detection layer (silicon) placed in contact with a conversion layer (boron) in a planar geometry. The boron material used in the model is enriched to 80% in the boron-10 isotope. A neutron absorbed in the active boron volume produces a 0.84 MeV lithium-7 ion, a 1.47 MeV alpha particle, and a 0.48 MeV gamma ray with 94% probability. With 6% probability it produces a 1.02 MeV lithium-7 ion with a 1.78 MeV alpha particle. Because the range of the 1.47 MeV alpha particle is only approximately 3.3 μm [17], for maximum efficiency the boron layer should be 3 μm [48]. Commercial silicon wafers are much thicker, around 300 μm [48], with varying diameters. Because these materials offer little moderation and the mean free path of the charged particles produced is small compared with the size of the silicon wafer, a 1 cm^2 area wafer with a 3 μm layer of boron on top of a 300 μm layer of silicon is simulated; larger area wafers may be approximated by scaling the result by the area of the actual detector. For the moderating material, a semi-infinite polyethylene layer is placed on the surface of the boron layer with thicknesses 0 cm, 1 cm, 5 cm, and 10 cm. The source is a beam directed perpendicular to the polyethylene face evenly distributed across the 1 cm^2 area. An energy-dependent pulse-height tally for alpha particles, photons, and lithium-7 ions is placed in the silicon volume to produce a detector spectrum.

To compute the detector response to the background, the models are simulated with respect to a cosine distributed source placed on each side of the detector. Any symmetry in the detector is utilized to reduce the number of simulations.

4.1.2.4 Cosmic Ray Induced Background

The neutron background is assumed to be a uniform flux with energy spectrum taken from reference [42]. This flux is cast as a uniformly distributed volumetric source around an array of box detectors with dimensions 30×30×30 cm. These detectors are simulated with and without the truck present. Because the background varies little with position along the length of the truck, the detectors are placed in a coarser mesh than the one employed for the source within the truck. An array of 10×4 detectors is placed along the side of the truck and a 10×2 array on half of the top side. The energy dependent current entering each face of the detector boxes is tallied. Neutrons that enter each detector volume are flagged and subtracted from the total current, to avoid neutrons tallying multiple times on each surface.

4.2 XPASS Software

XPASS is a text-based user interface that allows specification of scenario parameters, invokes the models described in this dissertation, and post processes detector signal results to generate detection probabilities. It parses data files to import post-processed transport data to generate response functions and interpolates on these functions based on user input. An example input file is shown in 4.1. A detailed description of the input file and its options are listed in Appendix B. The source code is also listed in Appendix B.

The statistical uncertainty incurred from utilizing data generated by a Monte Carlo transport code is propagated throughout the response functions in quadrature, assuming no covariance or correlation between the data. However, the data is most likely correlated. For example, gamma decay branches and cascades within the SNM are highly dependent processes. Furthermore, the current leaking from the SNM from a single source energy is correlated across energy groups. Propagating the statistical uncertainty in quadrature disregards these dependencies.

As an example of how these dependencies might be accounted for, let N be the number of energy groups, $\mathbf{q} \in \mathbb{R}^{N\times 1}$ be a source term, $\mathbf{R} \in \mathbb{R}^{N\times N}$ be a response function matrix, and $\mathbf{q}' \in \mathbb{R}^{N\times 1}$ be the transformed source given by

$$\mathbf{q}' = \mathbf{R}\mathbf{q}. \tag{4.1}$$

Because there is uncertainty associated with both the source and response function, the propagation of uncertainty requires additional notation defined in reference [49]. Let $d\mathbf{q}'/d\mathbf{R}$ be the $N^2 \times N^2$ Jacobian matrix of the matrix-vector multiplication with respect to $\mathbf{R}$, $d\mathbf{q}'/d\mathbf{q}$ be the $N^2 \times 1$ Jacobian matrix with respect to $\mathbf{q}$, and $\Sigma(\mathbf{q})$ and $\Sigma(\mathbf{R})$ be $N^2 \times N^2$ block matrices of variance-covariance matrices. The general error propagation formula is

$$\Sigma(\mathbf{q}') = \frac{d\mathbf{q}'}{d\mathbf{R}}\Sigma(\mathbf{R})\left[\frac{d\mathbf{q}'}{d\mathbf{R}}\right]^T + \frac{d\mathbf{q}'}{d\mathbf{q}}\Sigma(\mathbf{q})\left[\frac{d\mathbf{q}'}{d\mathbf{q}}\right]^T + \frac{d\mathbf{q}'}{d\mathbf{R}}\Sigma(\mathbf{R},\mathbf{q})\left[\frac{d\mathbf{q}'}{d\mathbf{q}}\right]^T + \left(\frac{d\mathbf{q}'}{d\mathbf{R}}\Sigma(\mathbf{R},\mathbf{q})\left[\frac{d\mathbf{q}'}{d\mathbf{q}}\right]^T\right)^T. \tag{4.2}$$

There is no correlation between **R** and **q**, therefore Eq. 4.2 reduces to

$$\Sigma(\mathbf{q}') = \frac{d\mathbf{q}'}{d\mathbf{R}}\Sigma(\mathbf{R})\left[\frac{d\mathbf{q}'}{d\mathbf{R}}\right]^T + \frac{d\mathbf{q}'}{d\mathbf{q}}\Sigma(\mathbf{q})\left[\frac{d\mathbf{q}'}{d\mathbf{q}}\right]^T. \tag{4.3}$$

The entries in each row of **R** are uncorrelated, as they correspond to independent source energies. Each column of matrix **R** corresponds to the energy-dependent current or detector signal from a single source energy. Thus, the entries within each column of matrix **R** are potentially correlated. Because **q** is a column vector, these entries are never combined with each other, and the correlation between them is irrelevant to the variance on the result $\mathbf{q}'$. However, the correlation does propagate into the covariance on the members of $\mathbf{q}'$. MCNPX does not directly estimate the covariance on the entries of **R** and they must be determined independently. The covariance in $\Sigma(\mathbf{q})$ is determined from error propagation in the previous submodel.

The covariance terms in $\Sigma(\mathbf{R})$ can be approximated using Pearson's correlation coefficient [50],

$$\text{COV}_{x,y} = \sigma_x \sigma_y \rho_{x,y}, \tag{4.4}$$

where $\rho_{x,y}$ is a correlation coefficient ranging between -1 for perfectly negatively correlated terms to +1 for positively correlated terms. The sign and magnitude of these coefficients are dependent on the details of the radiation transport model. Thus, determining the covariance or the correlation coefficient for each submodel and their variations requires additional research not explored in this dissertation. Furthermore, manipulation and storage of the $N^2 \times N^2$ variance-covariance matrices for $N = 8831$, the number of energy groups for photons, presents additional challenges not solved here. However, the result of propagating covariances is expected to be negligible, as the variance propagated through the submodels, ignoring covariance, results in relative uncertainties being <<1%, which is small compared to the systematic error incurred by parameterizations and other approximations.

Listing 4.1: Example XPASS Input File

```
#     XPASS Threat Scenario Modeling Software

physics
    photon on        # toggle photon transport
    neutron on       # toggle neutron transport
    fissgamma on     # toggle prompt fission gamma source
    background on    # toggle natural background transport
    mactime on       # toggle macroscopic time
    timestep 0.1     # macroscopic time step [s]
    refeps 1e-10     # neutron reflection convergence threshold

source
    snm
        type heu     # choices: du,heu,wgpu,rgpu
        mass 5000    # g
        rho 18.95    # g/cc
        iso
            u232 3e-8
            u234 2.0
            u235 85
            u236 1.0
            u238 12
        age 20  # years
        shield
            layer
                type bopoly
                thick 10.0  # cm
                stream 0.0
            layer
                type lead
```

```
            thick 3.5    # cm
            stream 0.0
        posx 0  # cm (0,0,0) is cargo dead center
        posy 0  # cm
        posz 10 # cm


vehicle
    truck
        cargo void
        stream 0.0
        velocity 8.05    # [km/h]

background
    photon
        usoil 36     # uranium soil conc. [Bq/kg]
        uconc 46     # uranium concrete conc. [Bq/kg]
        thsoil 44    # thorium soil conc. [Bq/kg]
        thconc 21    # thorium concrete conc. [Bq/kg]
        ksoil 85     # k-40 soil conc. [Bq/kg]
        kconc 23.7   # k-40 concrete conc. [Bq/kg]
    neutron
        lat 30.61    # latitude [deg]
        long 96.32   # longitude [deg]
        smod 0.5     # solar modulation factor (0 to 1)
        elev 90.83   # elevation [m]

detection
    nai
        posx 198.1   # [cm]
        posy -10     # relative to detector array [cm]
        posz 212.8   # [cm]
        dimx 10      # dimension in x-direction [cm]
        dimy 10      # [cm]
        dimz 10      # [cm]
        eff 1.0      # light collection efficiency
        gebA 0.0         # GEB parameter A
        gebB 0.05086     # GEB parameter B
        gebC 0.30486     # GEB parameter C
        fanofac 1.0 # fano factor
        alarm
            gc               # gross count
                nint 10      # number of time steps to sum
            ew               # energy window
                nint 10      # number of time steps
                nwindow 10  # number of windows
                spacing lin # type of spacing lin/log
    he3
        posx 198.1   # [cm]
        posy 10      # [cm]
        posz 212.8   # [cm]
        height 20    # [cm]
        modrad 5     # moderator radial thickness [cm]
        refrad 2     # reflector radial thickness [cm]
        eff 1.0      # charge collection efficiency
        fanofac 1.0 # fano factor
```

```
alarm
    gc              # gross count
        nint 10 # number of time steps to sum
```

Chapter 5

Results

5.1 Benchmarking

The results of this method are compared to full forward MCNPX simulations as well as an integral computational benchmark done by an external source. Because experimental benchmarks require access to sensitive material, computational benchmarks are instead chosen to demonstrate that this tool produces comparable results.

5.1.1 MCNPX Benchmarks

5.1.1.1 Photon

The photon benchmark scenario chosen is a 2 kg sphere of WGPu aged 20 years (initial composition of 5×10^{-9} w/o Pu-236 ,0.015 w/o Pu-238 ,93.63 w/o Pu-239,6.0 w/o Pu-240,0.355 w/o Pu-241) at density 15.75 g/cm^3 (delta-phase metal). It shielded with 3.5 cm of lead and placed in the center of an empty truck-trailer. The detector is a $10\times10\times10$ cm NaI crystal 68.56 cm from the side of truck. Only the time interval in which the SNM is directly in front of the detector is studied.

The validity of the response functions for each independent submodel is compared against MCNPX simulations. To avoid the accumulation of error, the XPASS computed spectrum is input as the source for the next submodel. For example, the spectrum leaving the SNM as computed by XPASS is used as the source for both the MCNPX and XPASS shielding submodel. The integral scenario benchmarks presented in Section 5.1.2 demonstrate the consequences of this error accumulation. Also, because the largest source of error between XPASS and MCNPX is the statistical error in the MCNPX estimate, a chi-squared test is performed on the error distribution. If the error is purely statistical, a histogram of the error should match a normal distribution with mean zero.

The current leaving the SNM computed by MCNPX and XPASS are shown in Figure 5.1. Apparent in the spectrum are many discrete gamma energies unique to this WGPu composition and age. If the error is computed as the fractional error of MCNPX compared to XPASS, the frequency of errors is illustrated in Figure 5.2. The data appears to be normally distributed and it passes the chi-squared test, which indicates that the difference between XPASS and MCNPX is purely statistical. However, this distribution has a mean of 12%, revealing a systematic error in XPASS.

The difference is decomposed into four different energy ranges, and the resulting normalized distributions are illustrated in Figure 5.3. It is apparent that the low energy range (<1 MeV) has the highest difference with a mean of 16%. Increasing in energy, the means of the distributions are 11%, 10%, and 7%. The consistent decrease in average difference as a function of energy is explained by the use of uranium simulation data to approximate this plutonium simulation. The difference between uranium and plutonium cross section data is highest at lower energies. There is a competing effect from the use of the saturation spectrum in conjunction with the surface area to volume ratio. The error from this saturation approximation increases as a function of energy, as the saturation layer is not fully realized for the more penetrating photons [4].

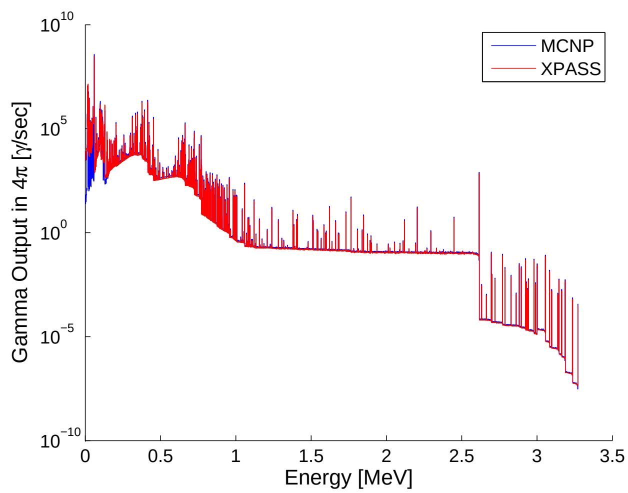


Figure 5.1: WGPu Photon Benchmark: Current Integrated Over SNM Sphere

The current leaving the shielding is compared in Figure 5.4. As expected, the lower end of the spectrum has been greatly reduced by lead's large photoelectric absorption cross-section. The error between MCNPX and XPASS passes the chi-squared test with mean 2%, indicating good agreement between the two simulations.

The current area density at the detector face is plotted in Figure 5.5. The spectrum is comparable to that leaving the shielding, except for an increase in the lower energy portion of the spectrum due to scattering. There

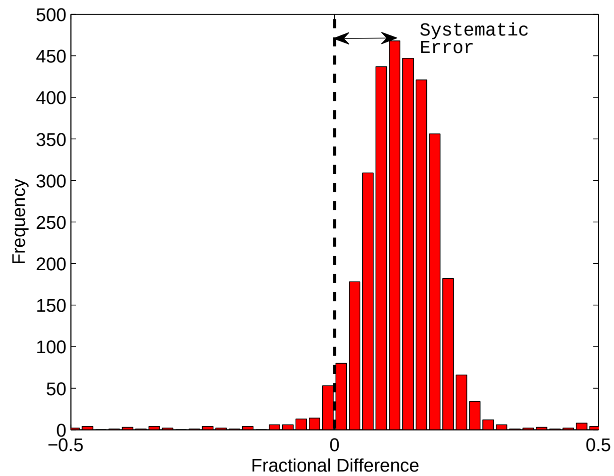


Figure 5.2: WGPu Photon Benchmark: Histogram of Fractional Difference for SNM Sphere Currents

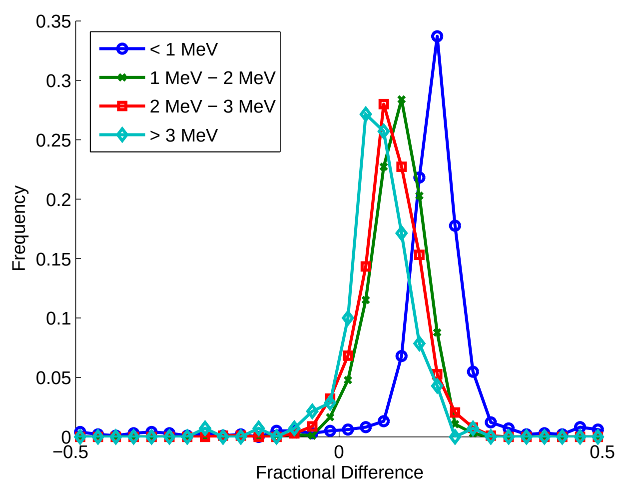


Figure 5.3: WGPu Photon Benchmark: Decomposed Histogram

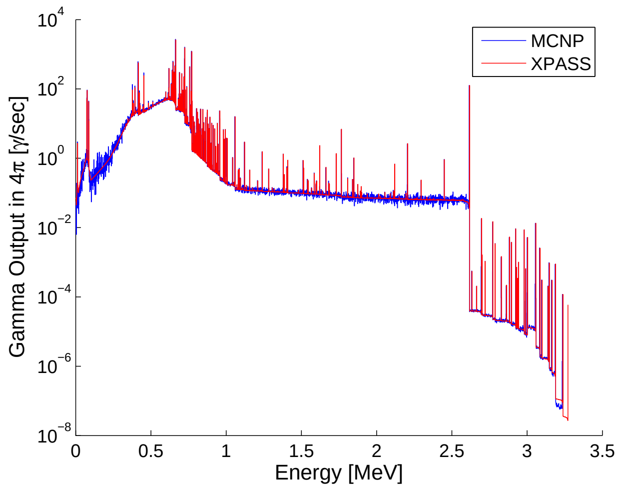


Figure 5.4: WGPu Photon Benchmark: Current Integrated Over Shield Surface

is also a backscatter peak at approximately 250 keV resulting from photons initially directed away from the detector that have backscattered toward the detector. The error between XPASS and MCNPX passes the chi-squared test with a mean of 3%.

The detector signal from the NaI detector is shown in Figures 5.6(a) and 5.6(b). Figure 5.6(a) is the signal without any Gaussian energy broadening (GEB). Figure 5.6(b) includes the XPASS post-processing GEB and the built-in GEB capabilities from MCNPX. The broadened data presents a spectrum which is typical of the energy resolution capabilities of NaI (approximately 7% at 662 keV). The error between the data sets passes the chi-squared test and the means are 9% and 2%, respectively. This indicates that the detector response function and the post-processing GEB is able to estimate the results

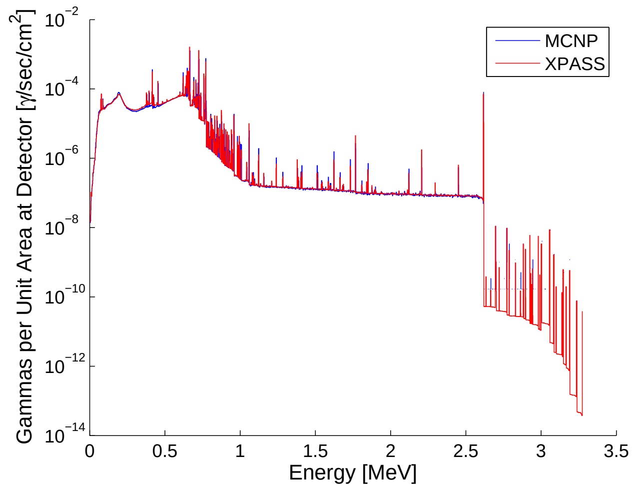


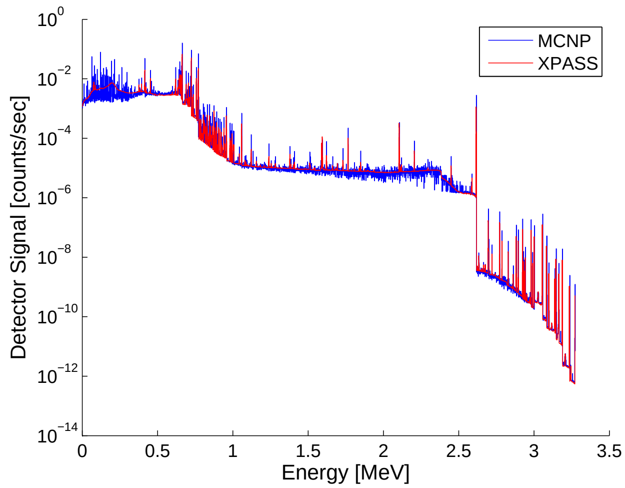
Figure 5.5: WGPu Photon Benchmark: Current at Detector Face

from MCNPX.

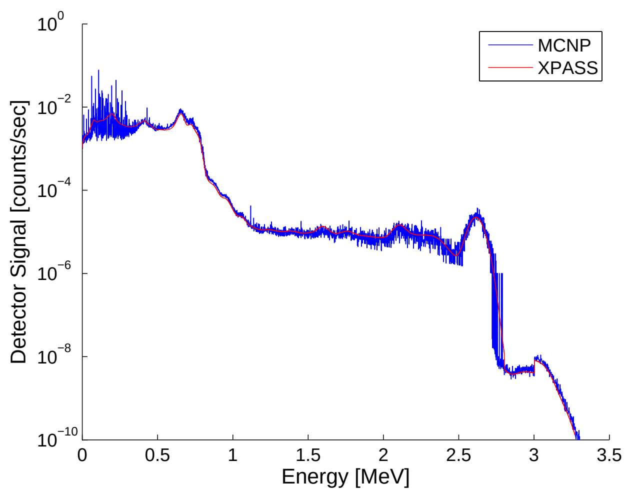
5.1.1.2 Neutron

The scenario chosen for the neutron benchmark is a 2 kg metal sphere of HEU at density 18.95 g/cm^3 aged 20 years (initial composition U-234 2%, U-235 85%, U-236 1%, U-238 12%). It is surrounded by 10 cm of BPE and placed in the center of an empty truck-trailer. A single helium-3 tube 20 cm in height surrounded by 5 cm of polyethylene with a 2 cm iron reflector is placed 68.56 cm from the side of the truck.

The integrated current exiting the bare SNM sphere without any shielding is compared to an MCNPX simulation shown in Figure 5.7. The time scale on the y-axis is given in shakes (1 shake = 1×10^{-8} s). There is a significant number of neutrons exiting the SNM in time less than 1×10^{-4} shakes because



(a) Without Gaussian Broadening



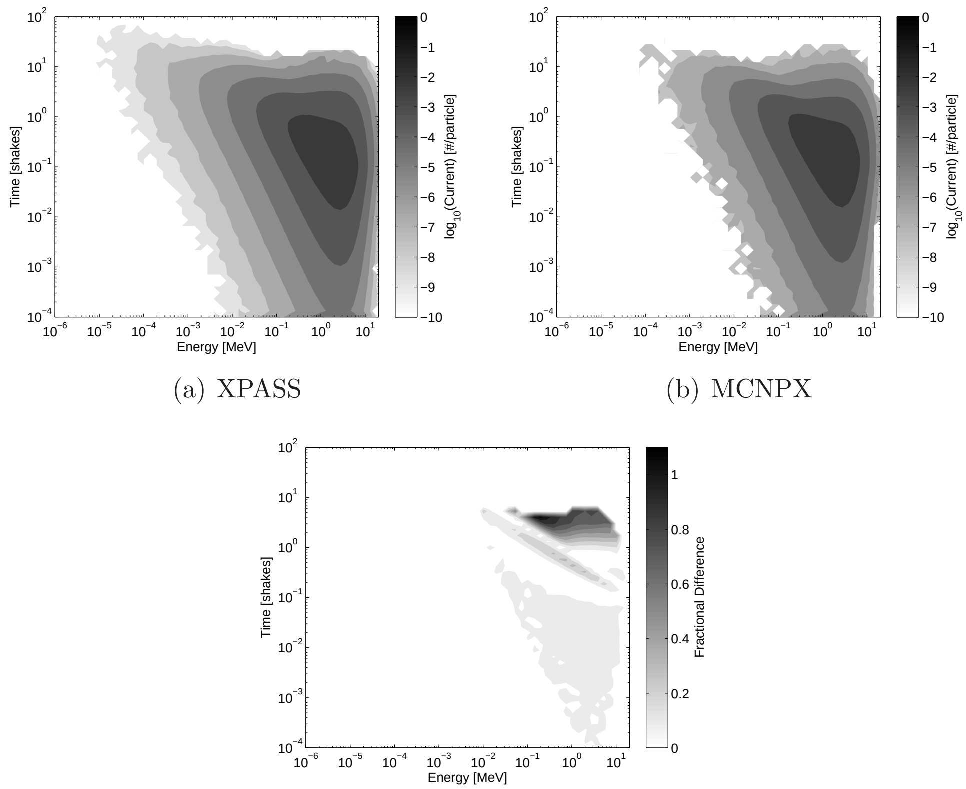
(b) With Gaussian Broadening

Figure 5.6: WGPu Photon Benchmark: Detector Signal

some are emitted at or very near the surface of the SNM. However, the elapsed time of neutrons escaping the SNM, especially for those which are born near the surface, is negligible compared to the time required to transport through the rest of the geometry. The error between XPASS and MCNPX can be as high as 100% in some regions due to the mass interpolation. The 2 kg sphere is interpolated between 1 kg and 5 kg masses. A 5 kg sphere of SNM allows neutrons to travel greater lengths and diffuse more than a 2 kg sphere. By including a 5 kg sphere in the mass interpolation, neutrons have a higher probability of existing at greater elapsed time than within a 2 kg sphere. Despite this interpolation artifact, the interpolated and simulated results integrated over all energy and time bins differ by less than 0.04%.

The integrated current exiting the shielding is compared to MCNPX in Figure 5.8. These simulations do not include any reflection between the SNM and shielding. The difference is generally less than 10% except in phase-space regions where the tally estimate is low and thus the statistical error is high. The largest error occurs in the high energy region with longer elapsed times as it did with the SNM. The integrated difference between the interpolation and the MCNPX simulation is 5%.

If reflection between the SNM and shielding is accounted for, the result from XPASS as compared to an MCNPX simulation is shown in Figure 5.9. From Figure 5.9(a), the addition of reflection cycles allows fission neutrons to be produced at greater elapsed times. The difference for the majority of data points is much less than 10%. There are some regions of the time-energy

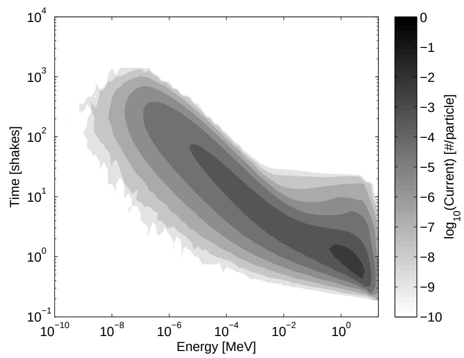


(a) XPASS

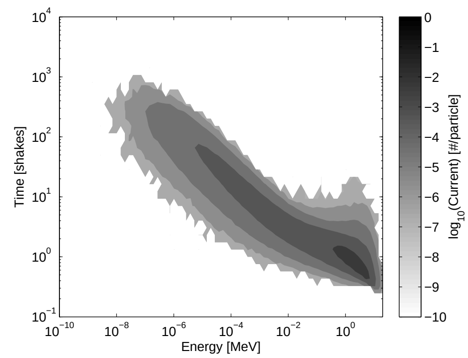
(b) MCNPX

(c) Difference

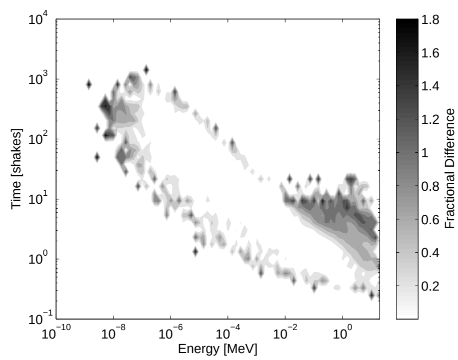
Figure 5.7: HEU Neutron Benchmark: Current Integrated Over SNM Sphere



(a) XPASS



(b) MCNPX

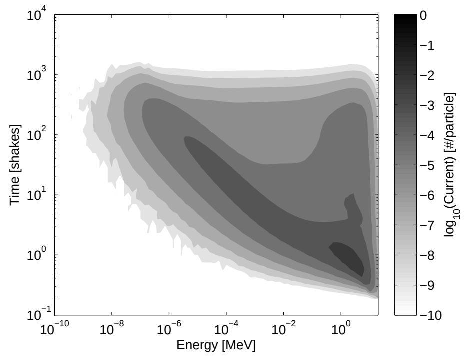


(c) Difference

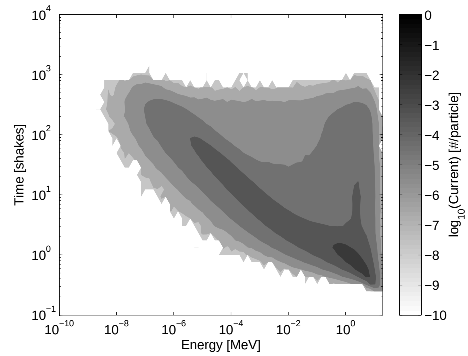
Figure 5.8: HEU Neutron Benchmark: Current Integrated Over Shielding Surface

phase space where the difference is as high as 80%. This error in the estimated data stems from the transmission scaling factors. Although these scaling factors preserve the total transmission and reflection probability, they are applied uniformly to resultant energy and time bins. This assumes that the contour of the time-energy phase space does not change significantly with these factors. However, the time and energy dependent current exiting the shield is a function of the SNM/shielding radial ratio, which introduces error. Because each reflection cycle utilizes these scaling factors, and each cycle is based on the results from the previous cycle, this error becomes progressively worse with each cycle as the contour is distorted further. Despite this error being considerable in localized phase-space regions, the difference between the integrated interpolation and simulation data is 3%, an indication that the total number of neutrons is being preserved by the scaling factors.

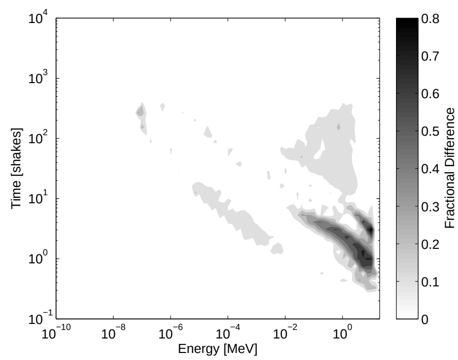
For this example, 14 reflection cycles were required to reach the convergence criterion that the reflection cycle contributes relatively less than 1×10^{-10} to the total integrated current at the SNM/Shielding interface. This relative contribution decreases exponentially with reflection cycle as illustrated in Figure 5.10. This figure also shows the relative contribution to the current for a 20 kg sphere of HEU with the same composition and age. Because the multiplication factor for the more massive sphere is larger, the slope of this line is greater and requires more reflection cycles to converge. The slope of this line is directly related to the criticality state of the system. Any subcritical system will have a negative slope, as each sequential cycle adds less



(a) XPASS



(b) MCNPX



(c) Difference

Figure 5.9: HEU Neutron Benchmark: Current Integrated Over Shielding Surface with Reflection

neutrons to the system. A critical system will have a slope of exactly zero, and a super-critical system will have a positive slope.

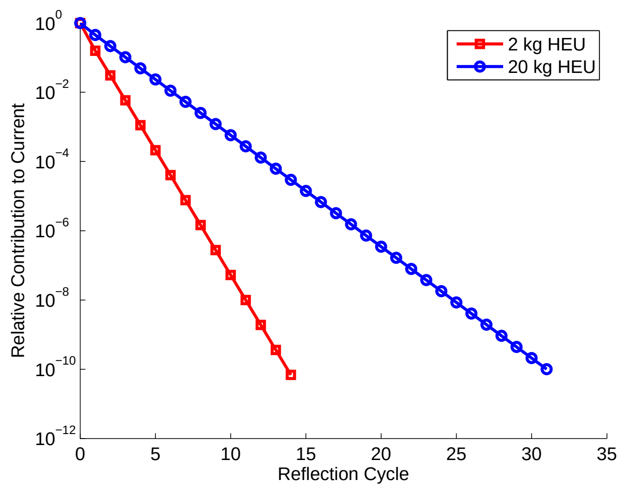
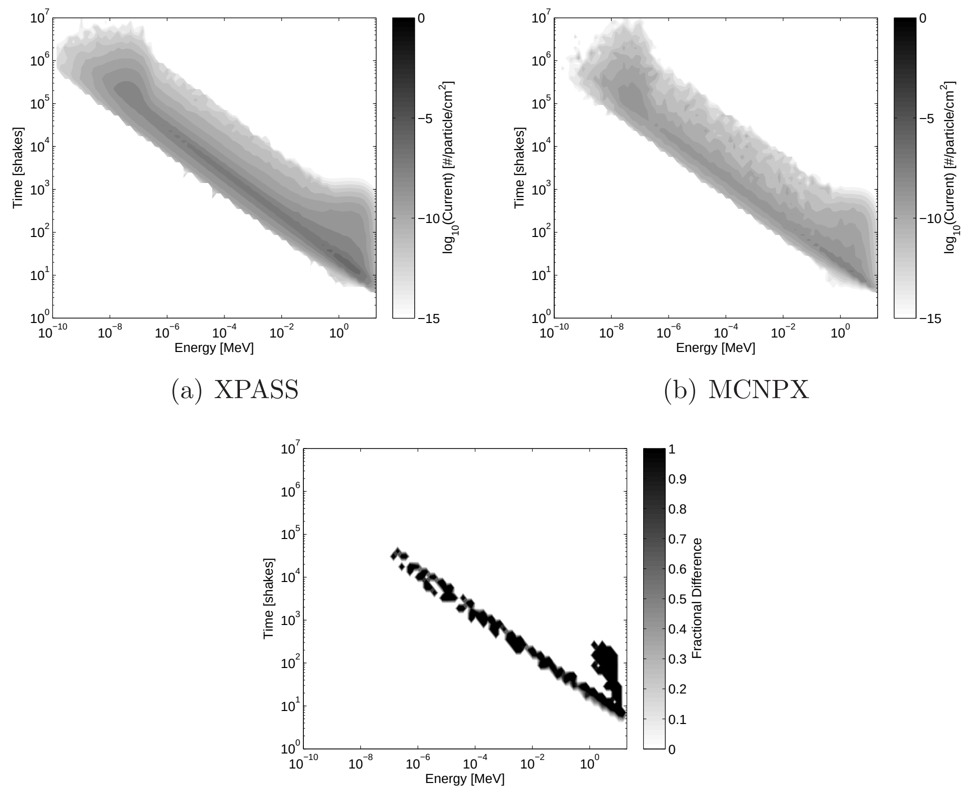


Figure 5.10: HEU Neutron Benchmark: Relative Current at SNM/Shield Interface for Different Reflection Cycles

The current at the detector face is compared in Figure 5.11. Uncollided radiation dominates the detector signal as the current is highest in the phase-space corresponding to the minimum time required for a neutron of a given source energy to travel from the shielding to the detector face. The relative difference between MCNP and XPASS is less than 10% for all phase space regions except within these uncollided regions. This is due to XPASS interpolating between detector points which are a greater straight-line distance from the source than the actual detector location. Uncollided neutrons require a greater length of time to traverse this distance than they would the shorter distance to the actual detector. This artificially shifts the uncollided vector forward in time. However, this shift is small and inconsequential when compared to the total neutron lifetime as will be shown in the detector signal. The integrated difference between XPASS and MCNPX for the current at the detector face is 14%.



(a) XPASS

(b) MCNPX

(c) Difference

Figure 5.11: HEU Neutron Benchmark: Current at Detector Face

The time-dependent detector count profile as calculated by XPASS and MCNPX are compared in Figure 5.12. There is a pulse in the detector signal at $1{\times}10^4$ shakes as the initial wave of fast neutrons is incident on the detector face. Neutrons which undergo multiple scattering events in the shielding and vehicle manifest as a smaller secondary pulse in the signal at 5×10^5 shakes. The relative difference between the two signals is considerable where the MCNPX statistical error is high. However, the integrated difference between them is 7%, indicating good agreement between the two. Furthermore, because these signals inherit any error from the response functions which compute the current at the detector face, the errors in time-of-flight stemming from interpolating on data representing different straight-line distances are seen to be negligible.

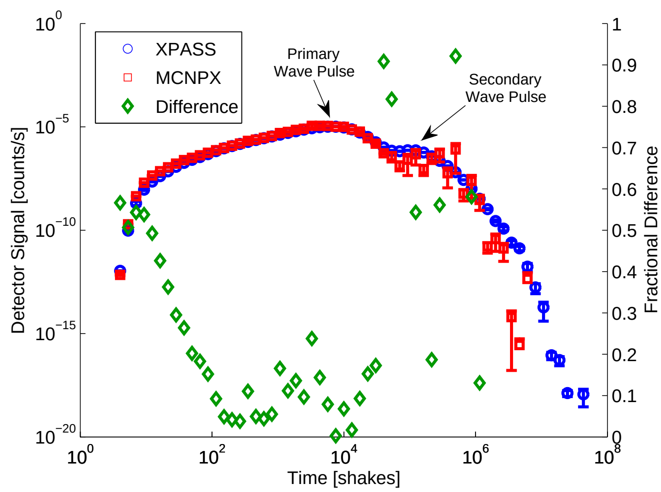


Figure 5.12: HEU Neutron Benchmark: Detector Signal

5.1.2 External Benchmarks

A study from Lawrence Livermore National Laboratory (LLNL) examined the combination of various SNM, shielding, and detector configurations to simulate a series of benchmark sources and detector responses [47]. The study employed MCNP and GADRAS together and included a variety of shielding materials not modeled here. The cases which XPASS is capable of modeling are compared in the following sections.

5.1.2.1 Photon

The photon scenario is a spherical mass of SNM surrounded by a spherical shell of shielding. The shielded SNM is placed in a vacuum and a Ludlum 4500 PVT panel with dimensions 3.8×52×173 cm and a NaI detector with dimensions 2×4×16 inches are placed 2.5 meters and 5 meters from the source, respectively. No information is given regarding elevation from the ground, detector housing, or external equipment. Thus, the detectors are assumed to be their active detection volumes suspended in space with no ground underneath. Therefore, the problem only consists of the shielded source and detector volume. It is also assumed that the distance reported in the LLNL study is the distance from the center of the SNM to the center of the detector face. Because the precise details and assumptions of the benchmark study are unavailable, the results are also compared to full forward simulations using MCNPX.

The current integrated over the surface of the SNM and shielding, PVT detector response, and NaI detector response from XPASS, LLNL, and MCNPX are compared in Table 5.1. For MCNPX results in which the statistical error is greater than 1%, the standard deviation is reported. The detector signals are integrated over the energy range 40 keV to 3.2 MeV.

The gamma output from the SNM between the three studies compare well. The PVT detector signals between XPASS and MCNPX compare to

SNM / Shielding	**Gamma Output in** 4π $[\gamma \cdot s^{-1}]$			**PVT Response @ 2.5 m** $[\text{counts} \cdot s^{-1}]$			**NaI Response @ 5 m** $[\text{counts} \cdot s^{-1}]$		
	XPASS	LLNL	MCNPX	XPASS	LLNL	MCNPX	XPASS	LLNL	MCNPX
2 kg WGPu Bare	4.70×10^8	4.67×10^8	5.47×10^8	3.60×10^5	4.39×10^5	$(3.70 \pm 2.14) \times 10^5$	2440	3840	1920±269
2 kg WGPu 1 cm Pb	9.91×10^5	1.19×10^6	1.05×10^6	2670	n/a	2930±88.0	55.2	138	71.3±14.3
2 kg RGPu Bare	1.49×10^{10}	1.47×10^{10}	1.65×10^{10}	6.87×10^5	1.03×10^7	$(5.58 \pm 0.37) \times 10^5$	7590	5.02×10^4	6610±1780
2 kg RGPu 2.9 cm Pb	8.64×10^5	1.13×10^6	8.65×10^5	2280	n/a	2170±173	40.1	123	34.7±12.8
5.54 kg HEU Bare	3.97×10^6	3.66×10^6	4.03×10^6	8760	7040	9030±902	254	292	210±83.4
5.54 kg HEU 3.0 cm Pb	4.40×10^4	4.56×10^4	4.39×10^4	88.0	90	88.2±9.71	2.06	n/a	3.04±1.3
5.54 kg VHEU Bare	3.99×10^6	3.44×10^6	3.82×10^6	8760	6580	7350±2660	244	271	264±39.4
5.54 kg VHEU 0.5 cm Pb	2.86×10^4	2.58×10^4	2.57×10^4	92.5	58	64±1	1.99	n/a	1.6±0.2
5.54 kg DU Bare	5.70×10^5	5.77×10^5	5.88×10^5	1380	1110	1415	31.4	57	48.2±15.1
5.54 kg DU 2.5 cm Pb	4.84×10^4	4.62×10^4	4.60×10^4	114	100	106±3	2.35	n/a	1.8±0.4

Table 5.1: Comparison of XPASS, LLNL, and MCNPX Integral SNM/Shielding Photon Benchmarks

within 20%. The LLNL PVT and NaI detector signals for many studies differs greatly. This difference stems from the LLNL study utilizing GADRAS, which contains empirical detector response functions. In general, XPASS is able to reliably reproduce full forward calculations from MCNPX and provide order-of-magnitude estimates to benchmarked empirical response functions. Because the data from XPASS is based on MCNPX simulations, this difference is a limitation of the ability of MCNPX to reproduce empirical detector signals which account for varying light and charge collection efficiencies as well as detector electronics.

5.1.2.2 Neutron

The LLNL neutron study is a single helium-3 tube 173 cm tall within a Ludlum 4500 monitor. However, because details regarding the placement of this tube, its diameter, and any moderating or reflecting material are not listed in the study, the neutron detector signal is only reported for the XPASS and MCNPX simulations. As before, the source is a solid sphere of SNM surrounded by a spherical shell of shielding material suspended in a vacuum. The helium-3 detector is placed 2.5 m away from the center of the sphere. The moderator thickness is set to 3.8 cm, and no iron reflector is employed.

The current integrated over the surface of the SNM and shielding and the response from a helium-3 detector is compared in Table 5.2. The neutron output and detector signal from the bare sources compares very well. For the shielded simulation, the LLNL study used alternating layers of lead and BPE.

Simulations from XPASS and MCNPX only considered the total thicknesses of these layers. This introduces a 26% error between the XPASS and LLNL results for the source term. The shielded XPASS source term differs from the MCNPX result by only 12% which is propagated through to the detector signal which also differs by 12%.

SNM / Shielding	**Neutron Output in** 4π [n·s^{-1}]			**He-3 Response @ 2.5 m** [counts·s^{-1}]	
	XPASS	LLNL	MCNPX	XPASS	MCNPX
2 kg WGPu Bare	2.10×10^5	2.08×10^5	2.10×10^5	44.1	43.4
2 kg WGPu 20 cm BPE 3.5 cm Pb	8.89×10^3	7.03×10^3	7.90×10^3	2.00	1.78
2 kg RGPu Bare	1.10×10^6	1.10×10^6	1.11×10^6	231	225

Table 5.2: Comparison of XPASS, LLNL, and MCNPX Integral Neutron Benchmarks

5.1.3 Applications

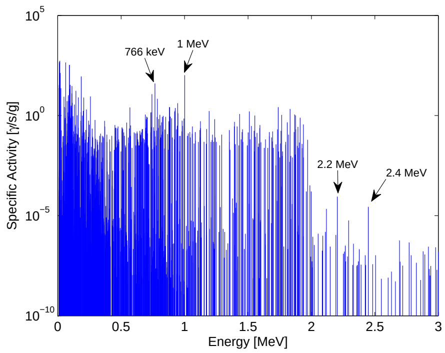
The previous sections demonstrate results from the implementation of the methods developed here into a threat scenario simulation program. This program is able to reliably compute detector signals that compare to full forward simulations using MCNPX, and are reasonable estimates of detector signals produced from empirical responses based on experimental data. The ability to rapidly simulate threat scenarios allows one to estimate the detector signal in real-time, providing a baseline against which the actual detector signal may be compared.

The advantage of energy-window (EW) algorithms is that they are relatively insensitive to baseline suppression compared to gross-count (GC) algorithms. In addition, they can potentially discriminate against NORM cargo based on spectral information. However, the degree to which the vehicle changes the shape of the spectrum is dependent on the cargo type. This rapid scenario simulation tool provides the ability to account for baseline suppression, and further enhance the capabilities of EW algorithms by accounting for spectral shifts from various cargo manifests. In addition, if the cargo manifest indicates NORM presence, this tool can predict the increase in counts and spectral shifts, further increasing sensitivity to abnormalities and reducing false alarms. Essentially, this tool allows one to change, in real-time, the baseline case from the natural unsuppressed background to the expected signal, which, as demonstrated here, can drastically improve the detection probability (DP) and reduce false alarms.

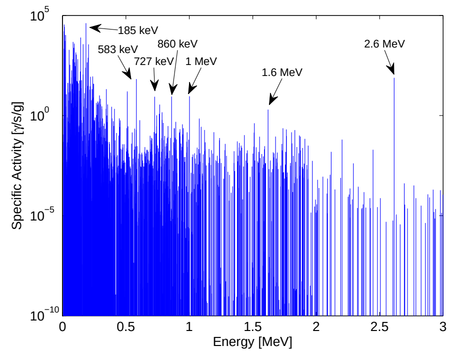
5.1.3.1 Baseline Suppression Estimation

To illustrate how this tool might be used to increase alarm sensitivity by predicting baseline suppression, various threat objects are placed in the center of a commercial truck cargo container width-wise and length-wise, elevated 10 cm from the cargo container floor. A single Ludlum 4500 PVT panel and a 2×4×16 inch NaI crystal are modeled and placed at a standoff distance of 68.56 cm from the side of the truck with the center of the detector face 2.32 m elevated from the ground for each. The PVT detector utilizes a GC algorithm. A modified EW algorithm is applied to the NaI detector which limits the windows to the peaks labeled in Figure 5.13. The energy-width of each window is set to the FWHM for that photopeak, calculated for NaI based on Gaussian energy broadening parameters $a = 0.0$ MeV $b = 0.05086$ MeV$^{1/2}$ and $c = 0.30486$ MeV^{-1}, which equates to approximately 7% FWHM at 662 keV. The windowed photopeaks for each SNM type are summed together.

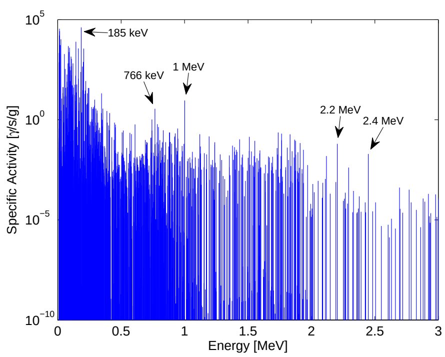
A further layer of abstraction to detector performance is the minimal detectable mass (MDM) for a given SNM type and shielding configuration. It is defined here as the mass of SNM which produces a DP greater than 95% at 1% false alarm probability. The MDM is computed based on a traditional GC or EW algorithm using the natural background as the baseline and also based on the expected signal using this tool. The expected signal is the benign signal based on the declared cargo manifest in a homogeneous configuration. Because realistic cargo configurations are heterogeneous, and details regarding their geometric configuration and composition may be difficult to ascertain



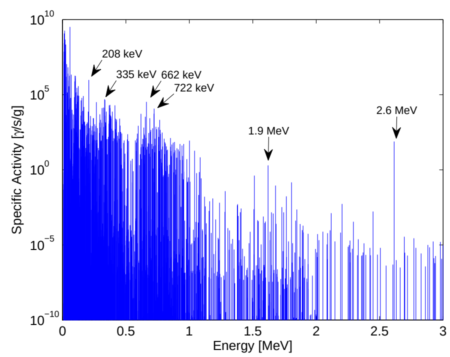
(a) DU



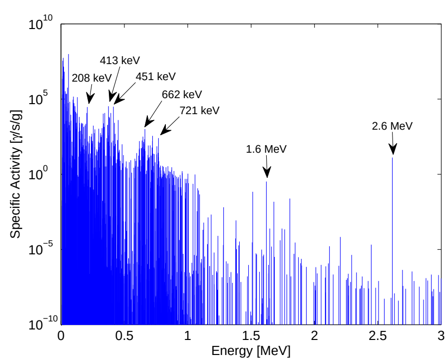
(b) HEU



(c) VHEU



(d) RGPu



(e) WGPu

Figure 5.13: Energy Signatures of SNM (Aged 20 Years)

during interrogation, the cargo response functions are superimposed from multiple types. The superposition is a weighted sum of multiple vehicle response functions, which transport radiation from the source to detector face. The weights are the percentages of each cargo type present, which can adopt multiple interpretations. One interpretation is that the weights are probabilities of that cargo type being present. The alternative interpretation considered here is that these weights approximate a heterogeneous cargo configuration. Because the baseline signal is based on an assumed homogeneous cargo configuration, making the actual cargo composition heterogeneous, in which the SNM is present, introduces uncertainty into the baseline estimate. The assumed and actual cargo compositions are summarized in Table 5.3. This study could be extended by increasing the number of assumed and actual cargo templates over a range to quantify the sensitivity of the result to the error in the cargo estimate.

	Low-Z		**Mid-Z**		**High-Z**	
	Assumed	Actual	Assumed	Actual	Assumed	Actual
Void		10%		10%		10%
Low-Z	100%	85%		10%		10%
Mid-Z		5%	100%	75%		10%
High-Z				5%	100%	70%

Table 5.3: Assumed and Actual Cargo Compositions

To avoid presenting MDM's for realistic cases, the data is presented as the relative change in MDM when the alarm algorithm is based on real-time detector signal estimates,

$$\% \text{ Change in MDM} = \frac{\text{Real Time MDM} - \text{Normal MDM}}{\text{Normal MDM}}. \tag{5.1}$$

The percent reduction for the three cargo types is presented in Table 5.4, 5.5, and 5.6. In all cases, the detector performance is improved when the alarm algorithm is provided with a baseline signal which is more representative of the expected background. This improvement is especially large with the PVT detector utilizing the GC algorithm. Because the relative fluctuation in the PVT signal is small compared to the NaI detector, accounting for baseline suppression drastically increases its sensitivity to abnormalities. The NaI signal also benefits from the baseline suppression estimate, although to a lesser degree.

SNM / Shielding	**Change in MDM for PVT**	**Change in MDM for NaI**
WGPu 1.0 cm Pb	-97%	-68%
RGPu 2.9 cm Pb	-97%	-70%
HEU 0.1 cm Pb	-96%	-56%
VHEU 0.1 cm Pb	-96%	-36%
DU 0.3 cm Pb	-96%	-70%

Table 5.4: Relative Improvement in Detector Performance for Assumed Low-Z Cargo Type

SNM / Shielding	**Change in MDM for PVT**	**Change in MDM for NaI**
WGPu 1.0 cm Pb	-98%	-67%
RGPu 2.9 cm Pb	-98%	-72%
HEU 0.1 cm Pb	-98%	-57%
VHEU 0.1 cm Pb	-98%	-40%
DU 0.3 cm Pb	-99%	-67%

Table 5.5: Relative Improvement in Detector Performance for Assumed Mid-Z Cargo Type

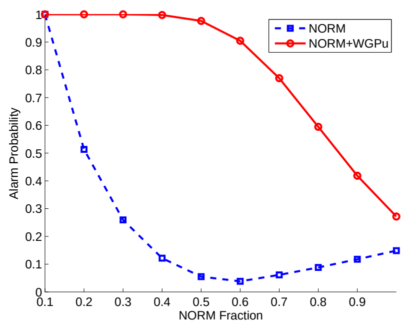
SNM / Shielding	**Change in MDM for PVT**	**Change in MDM for NaI**
WGPu 1.0 cm Pb	-99%	-69%
RGPu 2.9 cm Pb	-99%	-70%
HEU 0.1 cm Pb	-99%	-65%
VHEU 0.1 cm Pb	-99%	-46%
DU 0.3 cm Pb	-99%	-59%

Table 5.6: Relative Improvement in Detector Performance for Assumed Mid-Z Cargo Type

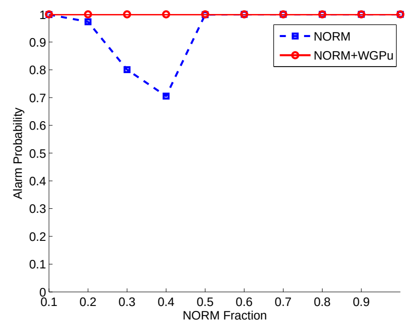
5.1.3.2 NORM Discrimination

The data presented thus far considered non-radioactive cargo types. At detector installations, NORM cargos can limit the performance of detectors by causing false alarms, forcing operators to raise the alarm threshold or use expensive and time consuming secondary screening methods [51]. Even EW algorithms have difficulty discriminating against all types of NORM cargos. As an example, consider three common NORM cargos: cat litter, phosphates, and bananas. The cat litter and phosphates are assumed to be of cargo composition 10% void, 20% low-Z, and 70% mid-Z. The banana cargo is set to 10% void, 70% low-Z, and 20% mid-Z. The mass of the NORM is defined by the mass fraction of the total cargo it comprises and is assumed to be uniformly mixed throughout the cargo. Using a five-window EW algorithm on a Ludlum 4500 PVT detector, the alarm probability is computed for the NORM for various mass fractions, fixing the false alarm probability at 1%. In addition a 2 kg WGPu shielded with 1 cm is simulated with the NORM to examine the detectability of a threat object masked by NORM. The alarm probability is plotted for various mass fractions in Figure 5.14.

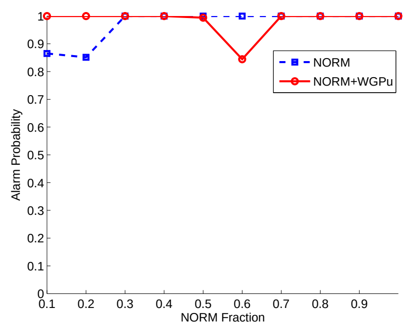
The alarm probabilities shown in Figure 5.14 are not monotonic because the EW algorithm examines each window individually to determine an alarm probability. The window with the highest probability is taken to be the alarm probability. The ability of the EW algorithm to discriminate against NORM and detect abnormalities from the WGPu is inconsistent and highly dependent on the NORM type, amount of NORM, and cargo type. This makes any



(a) Phosphate



(b) Banana



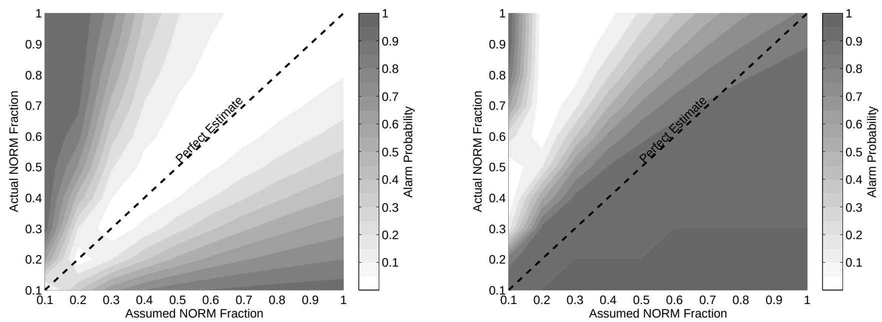
(c) Cat Litter

Figure 5.14: Alarm Probability for NORMs and WGPu

conclusions of the algorithm's effectiveness against a spanning set of threat scenarios difficult.

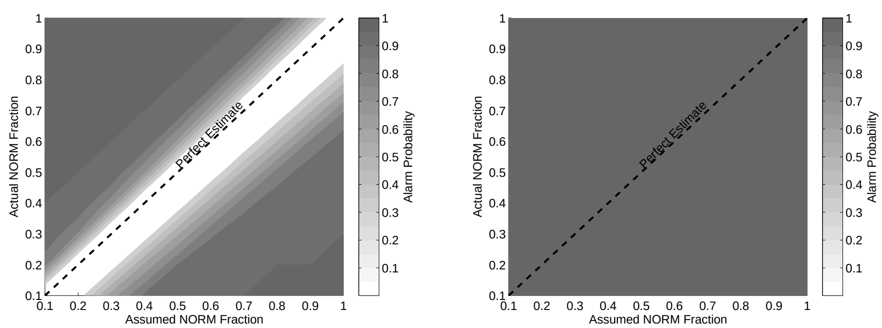
The tool developed here can reduce the false alarms due to NORM while maintaining the sensitivity to SNM hidden within NORM cargo by estimating the signal in real-time for the declared NORM-bearing cargo and using that as a baseline comparison for the EW algorithm. The baseline is computed for various fractions of NORM. To understand how cargo manifest uncertainty affects the alarm probability, the actual fraction of NORM is also varied. The alarm probability as a function of assumed and actual NORM mass fraction is illustrated in Figures 5.15, 5.16, and 5.17 again assuming 1% false alarm probability. In each figure, the left panel illustrates the alarm probability for NORM alone. The right panel shows the alarm probability for NORM with WGPu present. Each plot contains a dashed line indicating the alarm probability for a perfect estimate of the NORM mass fraction. An ideal alarm algorithm would have a white band along this diagonal in the left panel, indicating low false alarm probability for a perfect estimate of the NORM fraction. This band would also be wide as it would be insensitive to the difference between the assumed and actual NORM fraction. This ideal algorithm would also have a thick dark band along the diagonal in the right panel, indicating high detection probability to SNM despite any difference between the assumed and actual NORM fraction.

If the manifest is precise, the NORM alarm probability is small for all cases, outperforming the EW algorithm based on natural background. As the actual mass deviates from the assumed, the performance worsens, in some situations to a degree worse than the original EW algorithm. However, the performance degrades in a predictable manner, based on the error between the actual and assumed NORM mass fraction.



(a) Phosphate (b) Phosphate and WGPu

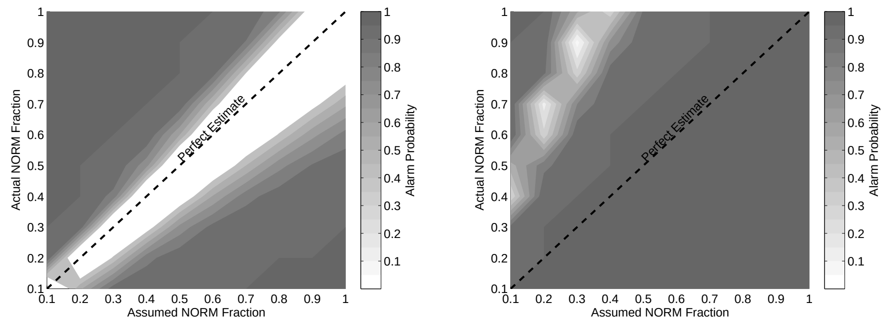
Figure 5.15: Alarm Probability for Phosphate and WGPu Using Real-Time Algorithm



(a) Bananas (b) Bananas and WGPu

Figure 5.16: Alarm Probability for Bananas and WGPu Using Real-Time Algorithm

Figures 5.15, 5.16 and 5.17 also illustrate how the difference between the assumed and actual cargo affect the detectability of WGPu hidden within the



(a) Cat Litter

(b) Cat Litter and WGPu

Figure 5.17: Alarm Probability for Cat Litter and WGPu Using Real-Time Algorithm

NORM. As before, when the NORM mass is known to within approximately 20%, the hidden WGPu is highly detectable while the alarm probability for benign NORM is maintained at a low level. As the actual NORM mass deviates greatly from the assumed, in some cases the DP is lower than that based on the traditional EW. However, this tool also predicts the gross counts the detector can expect given the assumed NORM cargo, which are a strong function of the total NORM mass, making any attempt to defeat the system by falsifying the actual mass of NORM difficult. As an example, the gross counts at the detector for the phosphate NORM are shown in Figure 5.18. To enter the area in which the sensitivity to threat objects is actually reduced, a smuggler would need to fabricate a cargo manifest which contains two to four times less NORM than actually present with a corresponding two to four times disagreement in gross counts. Although estimating the exact fraction of NORM is difficult based on uncertain cargo configurations, because the gross counts are almost a linear function of the NORM mass, this tool provides a simple check against grossly misrepresented NORM cargo manifests.

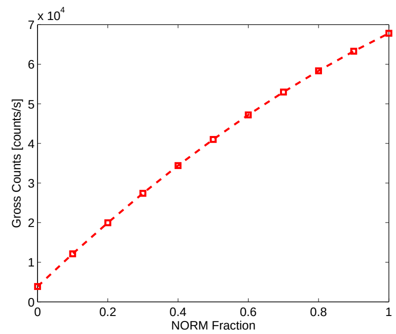
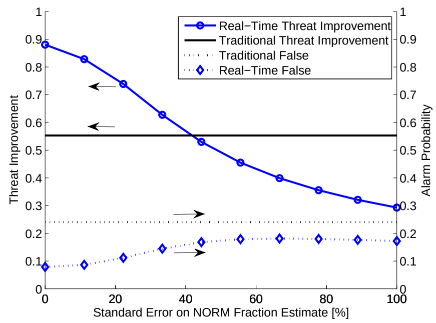


Figure 5.18: Gross Counts at Detector as a Function of Phosphate NORM Fraction

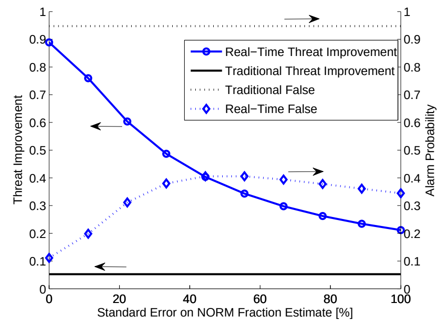
For the scenarios in which both NORM and WGPu are present, it is impossible to determine which source is actually causing the alarm, as it is their combined signal which defines the detector spectrum. Furthermore, the improvement of real-time NORM discrimination is dependent on the accuracy of the cargo manifest and the actual NORM mass fraction. To reduce the dimensionality of the data, it is assumed that the difference between the actual and assumed NORM fraction follows a normal distribution with variable standard error to be investigated in a sensitivity study. This provides a more physical interpretation of cargo manifest accuracy. In addition, it is also assumed that all NORM mass fractions are equally probable. These assumptions allow the computation of an alarm probability averaged over all actual mass fractions as a function of the certainty in the assumed mass fraction. This average alarm probability is computed for NORM only scenarios and NORM with WGPu present.

As metric of performance, the threat improvement is defined as the change in alarm probability with both NORM and WGPu present as compared to the NORM only case. An ideal alarm algorithm has a threat improvement equal to one, where the NORM alarm probability is zero and the WGPu is always detectable. An alarm which has no discernible benefit in detectability of the WGPu will have a threat improvement of zero, and a poor alarm algorithm where the addition of WGPu to the NORM decreases the alarm probability will have a negative threat improvement. The threat improvements for the real-time baseline estimation are compared for each NORM type as a function of cargo manifest error to the threat improvement achieved with the traditional alarm algorithm based on natural background radiation in Figure 5.19. Because the traditional algorithm requires no estimation of the cargo contents, it is a constant value averaged over all NORM mass fractions. While the threat improvement measure is important, it is also crucial that false alarms due to benign NORM cargos be reduced with this method. Therefore, the false alarm probability from NORM is also compared in Figure 5.19 for both types of algorithms.

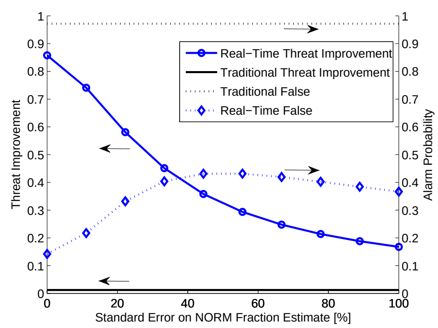
For phosphate NORM cargo, the false alarms from benign cargo are, on average, reduced despite the error in the cargo manifest assumption. The threat improvement is 35% higher for precisely known cargo compared to the traditional algorithm; however, the threat improvement declines as the estimate error increases, matching the traditional algorithm at 42% estimate error



(a) Phosphate



(b) Bananas



(c) Cat Litter

Figure 5.19: Threat Improvement and NORM Alarm Probability for Various Degrees of NORM Cargo Estimate Confidence

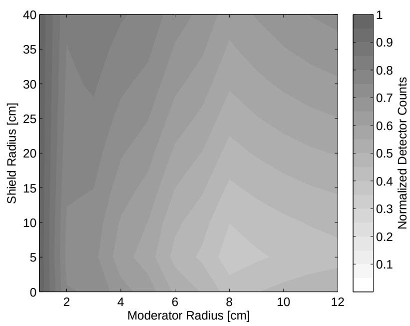
and dropping to 25% less than the traditional algorithm with 100% estimate error. The banana NORM cargo demonstrates a 85% reduction in false alarms for well known cargo and at least a 50% reduction for cargos with higher estimate error. In addition, the threat improvement using the real-time algorithm is higher for all estimate errors compared to the traditional algorithm, as much as 85%. The improvement for cat litter NORM is similar to the banana NORM.

For all the NORM scenarios simulated, a GC algorithm applied to the detector signal using the natural background as the baseline alarms 100% of the time due to the high activity NORM source. A GC algorithm applied using the real-time detector signal is useful only if the exact cargo composition is known, otherwise small perturbations to the cargo type result in false alarms and reduced sensitivity to threat objects. An EW algorithm is capable of discriminating against some NORM configurations, but its effectiveness is inconsistent. With a reasonable estimate of the NORM mass, combining the EW algorithm with an estimate of the detector signal both decreases false alarms from NORM and improves sensitivity to threat objects.

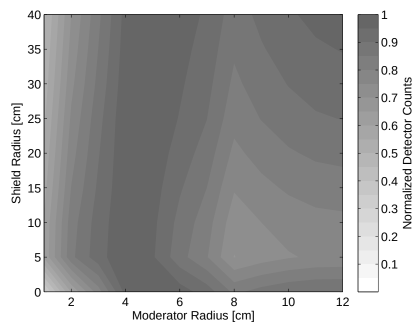
5.1.3.3 Neutron Detector Design

Although some solid state neutron detectors can achieve crude energy resolution, helium-3 neutron detectors have none. The peak efficiency of helium-3 detectors to different neutron energies is determined by the moderating layer thickness. Thus, by employing varying thicknesses of moderating material, the count rate in the detector changes based on the incident neutron spectrum. The emitted neutron flux energy spectrum is relatively insensitive to the SNM type, as it follows a Watt's fission spectrum. However, any surrounding shielding and cargo can affect this spectrum and change the sensitivity of the detector. For example, a 1 kg sphere of RGPu (nominal isotopic concentration) is surrounded by variable thickness of BPE and placed at the center of a commercial truck 50 cm above the floor of the cargo container. A 173 cm tall helium-3 tube with variable moderator thickness is placed at a 68.56 cm from the side of the truck. The normalized count rate at the detectors, without any background present, as a function of detector moderator thickness and BPE thickness is illustrated for a homogeneous low-Z, mid-Z, and high-Z cargo in Figure 5.20. Because the count rate as a function moderator thickness acts a crude form of neutron spectroscopy, the count rates for each BPE thickness are normalized to the maximum count rate of all moderations in that group.

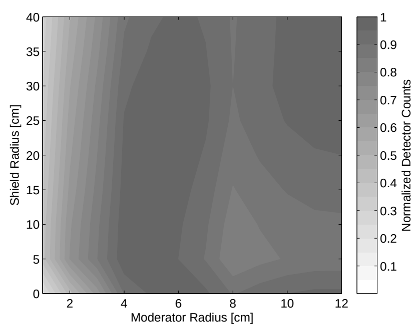
From Figure 5.19, there is no apparent BPE thickness that maximizes the response for all cargo types. The BPE shielding greatly reduces the thermal component of the neutron flux, leaving mostly fast neutrons as the source. The low-Z cargo is hydrogenous, downscattering neutrons and creating a thermal peak which is most easily detected by thin moderators as shown in Figure 5.20(a). The mid-Z and high-Z cargos do not have this moderating potential, allowing many fast neutrons to escape the truck, more easily detected by moderator thickness within the range of 4-6 cm. However the unique spectral transformations of the cargo have little effect on the detectability of the SNM.



(a) Low-Z



(b) Mid-Z



(c) High-Z

Figure 5.20: Normalized Count Rate at Helium-3 Detectors for Various Cargo Types

To illustrate this point, Figure 5.21 compares the normalized count rates for 15 cm of BPE shielding in various cargos to the natural background signal. The natural neutron background has a large thermal peak and no shielding such as BPE to reduce this peak. Therefore, with increasing moderator thickness, the background count rate consistently decreases. Despite the maxima shown in Figure 5.20, the DP is maximized for very thick moderators where the signal to noise ratio is maximized.

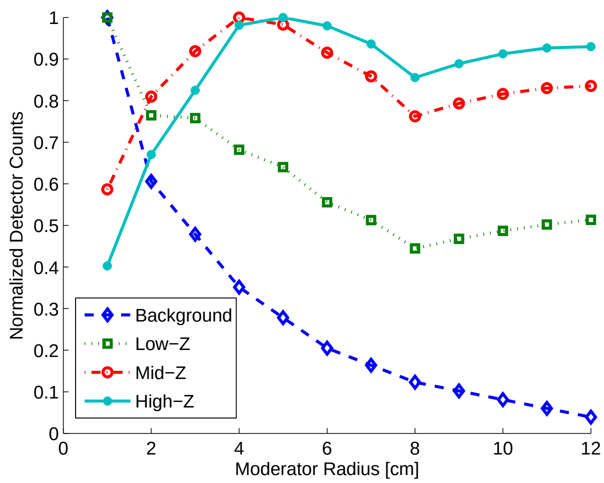


Figure 5.21: Normalized Count Rate at Helium-3 Detectors Compared to Background

Based on this data, the large thermal component of the background spectrum hinders detection. Focusing the detector on fast neutrons may increase sensitivity to SNM. As an example, if a neutron absorber with a large, low-energy radiative capture resonance surrounds the detector, such as cadmium-113, then neutrons with energy less than this resonance have a low probability of entering the detector. To approximate this behavior, a perfect neutron filter is placed around the detector which absorbs all neutrons with energy less than 10 eV. Its effect on the spectra shown in Figure 5.20 is illustrated in Figure 5.22. Because the thermal component of the neutron spectrum is removed, and the neutron background is mostly thermal neutrons, the addition of a neutron absorber improves the signal to noise ratio for all detectors.

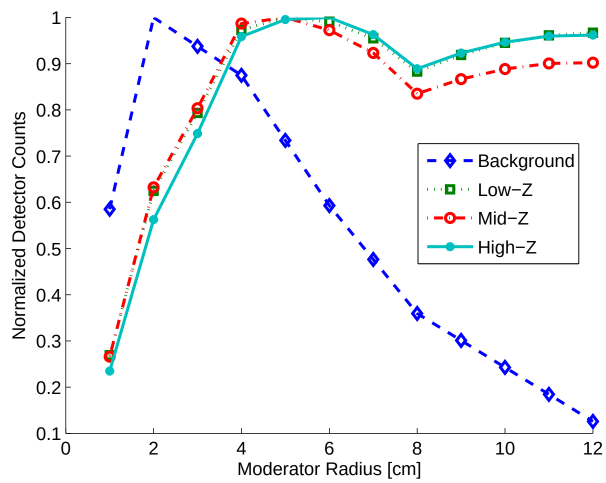
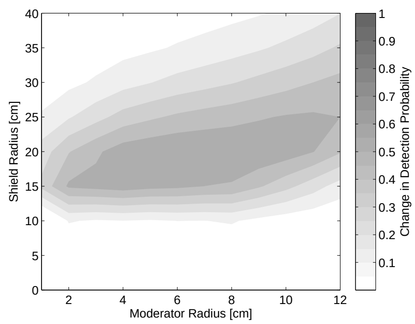


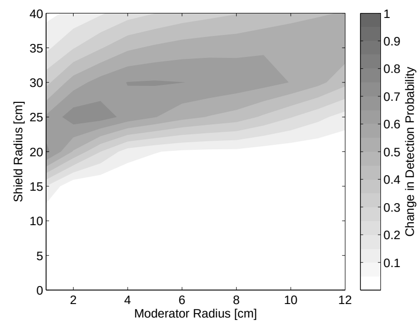
Figure 5.22: Normalized Count Rate at Helium-3 Detectors Wrapped in Thermal Neutron Absorber

To quantify the result of the improved signal to noise ratio, it is necessary to scale the neutron background signal appropriately. The scenario is modeled at 2300 m above sea level, where the neutron count rate is considerably higher than at sea level. The change in DP from adding the perfect neutron absorber as a function of moderator thickness and BPE thickness is shown in Figure 5.23. For thin BPE shields, the SNM is easily detected and the addition of a thermal neutron filter around the detector does not improve the DP. However, thick BPE shields greatly reduce the count rate in the detector, and the signal to noise ratio becomes increasingly important. There is a clear improvement in the DP for heavily shielded SNM from the addition of the neutron filter, even if the improvement is exaggerated by the use of a perfect neutron absorber. As expected, the DP is highest for thick moderators, where the sensitivity to fast neutrons is maximized. It should also be noted that these detection probabilities, although illustrative of the argument, are based on a fictional location in which the neutron background is high enough to mask the SNM; however, the neutron spectrum does not change with elevation and therefore the signal-to-noise ratio at any location will follow the same trend shown here.

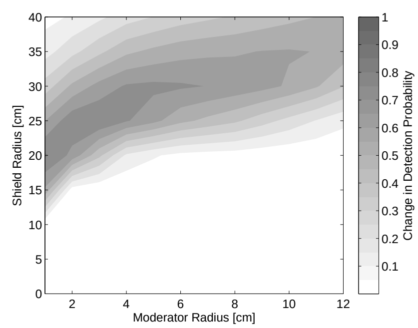
Previous studies have optimized moderating thickness to maximize threat source signals isolated from background [18]. Because the background spectrum is different than threat sources, maximizing the signal to noise ratio results in a different optimal value. This research concludes that helium-3 detector sensitivity to SNM may be improved by increasing sensitivity to fast neutrons by increasing moderator thickness and adding a thermal neutron filter around the detector.



(a) Low-Z



(b) Mid-Z



(c) High-Z

Figure 5.23: Change in Detection Probability for Various Cargo Types

Chapter 6

Conclusion

Threat scenarios may be modeled through radiation transport simulations. However, simulating the full phase-space of radiation within these models is computationally expensive. Given the large problem space of smuggler attributes and detector designs, this limits the utility of simulation as a tool to model and understand a spanning set of scenarios. Decomposing the scenario into submodels alleviates the computational burden by reducing the number of simulations at a geometric rate as the problem space grows. Central to the method of decomposition is the assumption that radiation transport within each submodel can be simulated independently, with dependencies captured through physically meaningful interface conditions. This is accomplished through the use of Green's functions, which encapsulate the radiation transport for each submodel. Convolving these functions combines the separate submodels to transport radiation through the entire scenario. This combination requires negligible computation time compared to the full forward radiation transport simulation. Previous work demonstrated that this method is applicable to a coarse photon energy group structure. However, this level of resolution limited the problem space to poor energy resolution photon detectors such as PVT.

Modeling the full space of available photon detector technologies requires a high degree of energy resolution, on the order of 1 keV, resulting in over 8000 energy bins. Because the computation time required for Green's function generation increases with the number of energy bins squared, this high resolution a prohibitive to treat by brute force, even if other components of the phase-space are handled implicitly through the interface conditions. As a solution, a new method was developed to interpolate photon currents and detector signals between sparsely spaced source energies, significantly reducing computational requirements and making this fine group structure feasible within the decomposition method.

The error incurred from source energy interpolation is less than 10% except where the interpolation spans a region in which the photon interaction cross-section is not monotonic. Combining multiple interpolations together to generate submodel response functions reproduces results from equivalent MCNPX simulations to within 10% except for the SNM submodel which inherits systematic error from a simplifying transport assumption. From an integral scenario transport perspective, this method compares to detector signals from full-forward MCNPX simulations to within 20%, and provides order-of-magnitude estimates to a MCNP/GADRAS computational benchmark.

Capturing neutron transport within the decomposition paradigm requires that the submodels have treatments available to account for inter-submodel interactions such as reflection. New albedo response function methods were developed to allow submodels to interact. This effect is especially important between the SNM and surrounding shielding, which acts as a reflector and produces additional pulses of fission neutrons. In addition to introducing dependencies between submodels, neutrons can exist within the scenario for a measurable length of time. Thus, neutron response function methods were developed with respect to time and energy, adding an additional dimension to the explicit phase-space. The integrated differences between generated response functions and MCNPX simulations are less than 15% for each submodel. However, localized errors in the time-energy phase space can be larger than 15% due to artificial shifts in elapsed time created by interpolating between response functions. Integral scenario benchmarks are within 12% of MCNPX simulations.

The methods developed for high resolution photon transport, neutron transport, and time dependence were combined into the software package XPASS. This implementation provides a threat scenario analysis platform capable of analyzing a wide variety of smuggler attributes and detection systems for land-based detection systems. Computational benchmarking demonstrated the ability of XPASS to quickly reproduce full-forward MCNPX simulations. This allows scenarios to be modeled in real-time during vehicle scanning, which can be used to increase the sensitivity of detection hardware by providing an estimate of the detector signal. For example, XPASS accounts for baseline suppression, a systematic weakness of many gross counting detectors, and significantly increases sensitivity by providing an estimate of the suppressed detector signal to the alarm algorithm. For a gross counting PVT detector scanning commercial trucks, the minimal detectable mass (MDM) for a variety of shielded SNM is reduced by 90% or more using this tool to predict the suppressed signal compared to the same alarm algorithm based on the natural background. Furthermore, baseline suppression also introduces spectral shifts into the background spectrum. When this effect is accounted for, the MDM is reduced by at least 36% for a NaI detector utilizing templates to search for SNM signatures.

The predictive ability of this tool is also useful for discriminating against nuisance alarms, particularly those arising from NORM. Energy window (EW) alarm algorithms are not consistently effective against all types of NORM and cargo configuration. However, if these alarms are provided with a detector signal estimate, they have the ability to consistently reduce false alarms while simultaneously increasing sensitivity to SNM hidden within NORM. The degree to which they improve EW performance is dependent on the type of NORM and the ability to correctly estimate the materials and geometry of the vehicle being scanned. For commercial trucks, if the mass of NORM is known to within 20% of the actual mass, false alarms from NORM are reduced by 10% for phosphates, 70% for bananas, and 67% for cat litter when compared to the same EW alarm based on natural background. Furthermore, the increase in alarm probability with SNM present is improved by 20% for phosphates, 60% for bananas, and 60% for cat litter.

In addition to being a valuable asset for informing alarm algorithms in real-time, this tool may be used to test detector designs. For example, with helium-3 detectors, it is desirable to design the moderating material surrounding the detector such that the detection probability of threat objects is maximized. The integrated simulation environment developed here was able to estimate how perturbations to the detector design affect the count rate from natural background sources as well as a variety of threat configurations. From this data it is evident that the optimal moderator thickness for isolated threat sources is not necessarily the best design when the natural background source is taken into account and the goal is maximize the signal to noise ratio. Furthermore, increasing detector sensitivity to fast neutrons by increasing the moderator thickness and adding a thermal neutron filter can improve the detection probability to heavily shielded SNM.

Combined with decomposition, the theory, methods, and implementation developed in this dissertation construct the basis for a general threat scenario simulation tool. The combination of computational efficiency and combinatoric mitigation makes this novel tool a virtual testbed for new applications and studies of threat scenarios not previously possible. New detectors and alarm algorithms may be developed and optimized based on a spanning set of threat scenarios. At the architectural level, deployment and operation of systems of detectors can be improved with respect to a variety of smuggler attributes, creating a feedback mechanism between macroscopic and localized detection probabilities. In addition, alarm algorithms may be trained in a virtual environment to match detector signals to libraries of statistically generated benign and threat cases. Detector signals can be computed in real-time, providing vehicle-specific cases against which to condition alarm algorithms.

6.1 Future Work

Building this platform into a comprehensive scenario analysis tool is application dependent and a constant process as the problem space changes. The reusability of response functions reduces this computational burden. For example, active interrogation may produce additional fission and delayed neutrons within the SNM. While response functions are required to capture the radiation transport of the interrogating source and the initiation of fission within the SNM, post-fission event these neutrons simply become another source term in the current set of response functions.

The flexibility of decomposition does not limit the submodel granularity to that outlined here. For instance, the commercial truck and cargo could become separate submodels, or the cargo itself could be decomposed into separate pallets of cargo. The caveat with finer granularity is the radiation transport information required, such as spatial and angular distributions, and the additional number of Green's function convolutions required to reconstruct the scenario.

Improving alarm algorithms by providing real-time detector signal estimates hinges on a pre-generated library of response functions. While the prototype modeling tool and response function library was developed in this dissertation, a full library of response functions, such as a standard set of vehicles, which together can model most scenarios is required before this tool can be implemented into detection systems. However, with the current set of response functions this tool may be used for sensitivity analysis validation for a basic set of commercial truck cargos.

As mentioned before, this software can be a virtual training tool for alarm algorithms. For example, a large number of threat scenarios may be sampled to produce a set of benign detector signals for a principal components analysis. Principal components are orthogonal vectors in the energy window space, ranked by their ability to explain the variance in the benign data set. When the principal components are used as a basis set for an observed detector signal, the coefficients of these basis vectors quantify the signal's departure from the benign set. The definition and richness of this benign set of data is critical to produce robust principal components. This platform is capable of efficiently sampling from a large set of benign cases to fully populate this data set.

The method of decomposition mitigates the combinatoric problem of a large scenario space. The wide application of this platform stems from the ability to replicate high fidelity radiation transport simulations. This is possible because Green's functions capture the full transport phase-space within each submodel, but more importantly the methods developed in this dissertation provide the critical coupling between submodels, preserving the necessary components of the phase-space. This allows high fidelity threat scenario simulation in negligible computation time compared to a full forward transport simulation. Furthermore, within the decomposition framework response functions may be continually added and updated, making the problem space customizable for different applications. The implementation developed here provides an extensible platform onto which dynamic problem spaces may be built and studied.

Appendices

Appendix A

Appendix A

A.1 Photon Energy Structure

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
1.0×10^{-2}	2.0×10^{-2}	3.0×10^{-2}	4.0×10^{-2}	5.0×10^{-2}
6.0×10^{-2}	7.0×10^{-2}	8.0×10^{-2}	9.0×10^{-2}	1.0×10^{-1}
2.0×10^{-1}	3.0×10^{-1}	4.0×10^{-1}	5.0×10^{-1}	6.0×10^{-1}
7.0×10^{-1}	8.0×10^{-1}	9.0×10^{-1}	1.0	1.02199
1.022	1.05	1.1	1.2	1.3
1.4	1.5	1.6	1.7	1.8
1.9	2.0	2.1	2.2	2.3
2.4	2.5	2.6	2.7	2.8
2.9	3.0	4.0	5.0	6.0
7.0	8.0	9.0	1.0×10^{1}	1.5×10^{1}
2.0×10^{1}	2.5×10^{1}	3.0×10^{1}	3.5×10^{1}	4.0×10^{1}
4.5×10^{1}	5.0×10^{1}	5.5×10^{1}	6.0×10^{1}	6.5×10^{1}
7.0×10^{1}	7.5×10^{1}	8.0×10^{1}	8.5×10^{1}	9.0×10^{1}
9.5×10^{1}	1.0×10^{2}			

Table A.1: Source Energy Points for Photons

Table A.2: Photon Tally Energy Bins

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
1×10^{-3}	1.977×10^{-3}	2.953×10^{-3}	3.93×10^{-3}	4.906×10^{-3}
5.883×10^{-3}	6.859×10^{-3}	7.836×10^{-3}	8.812×10^{-3}	9.789×10^{-3}
1.077×10^{-2}	1.174×10^{-2}	1.272×10^{-2}	1.37×10^{-2}	1.467×10^{-2}
1.565×10^{-2}	1.662×10^{-2}	1.76×10^{-2}	1.858×10^{-2}	1.955×10^{-2}
2.053×10^{-2}	2.151×10^{-2}	2.248×10^{-2}	2.346×10^{-2}	2.444×10^{-2}
2.541×10^{-2}	2.639×10^{-2}	2.737×10^{-2}	2.834×10^{-2}	2.932×10^{-2}
3.03×10^{-2}	3.127×10^{-2}	3.225×10^{-2}	3.323×10^{-2}	3.42×10^{-2}
3.518×10^{-2}	3.616×10^{-2}	3.713×10^{-2}	3.811×10^{-2}	3.909×10^{-2}
4.006×10^{-2}	4.104×10^{-2}	4.202×10^{-2}	4.299×10^{-2}	4.397×10^{-2}
4.494×10^{-2}	4.592×10^{-2}	4.69×10^{-2}	4.787×10^{-2}	4.885×10^{-2}
4.983×10^{-2}	5.08×10^{-2}	5.178×10^{-2}	5.276×10^{-2}	5.373×10^{-2}
5.471×10^{-2}	5.569×10^{-2}	5.666×10^{-2}	5.764×10^{-2}	5.862×10^{-2}
5.959×10^{-2}	6.057×10^{-2}	6.155×10^{-2}	6.252×10^{-2}	6.35×10^{-2}
6.448×10^{-2}	6.545×10^{-2}	6.643×10^{-2}	6.741×10^{-2}	6.838×10^{-2}
6.936×10^{-2}	7.034×10^{-2}	7.131×10^{-2}	7.229×10^{-2}	7.327×10^{-2}
7.424×10^{-2}	7.522×10^{-2}	7.619×10^{-2}	7.717×10^{-2}	7.815×10^{-2}
7.912×10^{-2}	8.01×10^{-2}	8.108×10^{-2}	8.205×10^{-2}	8.303×10^{-2}
8.401×10^{-2}	8.498×10^{-2}	8.596×10^{-2}	8.694×10^{-2}	8.791×10^{-2}
8.889×10^{-2}	8.987×10^{-2}	9.084×10^{-2}	9.182×10^{-2}	9.28×10^{-2}
9.377×10^{-2}	9.475×10^{-2}	9.573×10^{-2}	9.67×10^{-2}	9.768×10^{-2}
9.866×10^{-2}	9.963×10^{-2}	1.006×10^{-1}	1.016×10^{-1}	1.026×10^{-1}
1.035×10^{-1}	1.045×10^{-1}	1.055×10^{-1}	1.065×10^{-1}	1.074×10^{-1}
1.084×10^{-1}	1.094×10^{-1}	1.104×10^{-1}	1.114×10^{-1}	1.123×10^{-1}
1.133×10^{-1}	1.143×10^{-1}	1.153×10^{-1}	1.162×10^{-1}	1.172×10^{-1}
1.182×10^{-1}	1.192×10^{-1}	1.201×10^{-1}	1.211×10^{-1}	1.221×10^{-1}
1.231×10^{-1}	1.24×10^{-1}	1.25×10^{-1}	1.26×10^{-1}	1.27×10^{-1}
1.28×10^{-1}	1.289×10^{-1}	1.299×10^{-1}	1.309×10^{-1}	1.319×10^{-1}
1.328×10^{-1}	1.338×10^{-1}	1.348×10^{-1}	1.358×10^{-1}	1.367×10^{-1}
1.377×10^{-1}	1.387×10^{-1}	1.397×10^{-1}	1.406×10^{-1}	1.416×10^{-1}
1.426×10^{-1}	1.436×10^{-1}	1.446×10^{-1}	1.455×10^{-1}	1.465×10^{-1}
1.475×10^{-1}	1.485×10^{-1}	1.494×10^{-1}	1.504×10^{-1}	1.514×10^{-1}
1.524×10^{-1}	1.533×10^{-1}	1.543×10^{-1}	1.553×10^{-1}	1.563×10^{-1}
1.572×10^{-1}	1.582×10^{-1}	1.592×10^{-1}	1.602×10^{-1}	1.612×10^{-1}
1.621×10^{-1}	1.631×10^{-1}	1.641×10^{-1}	1.651×10^{-1}	1.66×10^{-1}
1.67×10^{-1}	1.68×10^{-1}	1.69×10^{-1}	1.699×10^{-1}	1.709×10^{-1}
1.719×10^{-1}	1.729×10^{-1}	1.739×10^{-1}	1.748×10^{-1}	1.758×10^{-1}
1.768×10^{-1}	1.778×10^{-1}	1.787×10^{-1}	1.797×10^{-1}	1.807×10^{-1}
1.817×10^{-1}	1.826×10^{-1}	1.836×10^{-1}	1.846×10^{-1}	1.856×10^{-1}
1.865×10^{-1}	1.875×10^{-1}	1.885×10^{-1}	1.895×10^{-1}	1.905×10^{-1}
1.914×10^{-1}	1.924×10^{-1}	1.934×10^{-1}	1.944×10^{-1}	1.953×10^{-1}
1.963×10^{-1}	1.973×10^{-1}	1.983×10^{-1}	1.992×10^{-1}	2.002×10^{-1}
2.012×10^{-1}	2.022×10^{-1}	2.031×10^{-1}	2.041×10^{-1}	2.051×10^{-1}
2.061×10^{-1}	2.071×10^{-1}	2.08×10^{-1}	2.09×10^{-1}	2.1×10^{-1}
2.11×10^{-1}	2.119×10^{-1}	2.129×10^{-1}	2.139×10^{-1}	2.149×10^{-1}
2.158×10^{-1}	2.168×10^{-1}	2.178×10^{-1}	2.188×10^{-1}	2.197×10^{-1}
2.207×10^{-1}	2.217×10^{-1}	2.227×10^{-1}	2.237×10^{-1}	2.246×10^{-1}
2.256×10^{-1}	2.266×10^{-1}	2.276×10^{-1}	2.285×10^{-1}	2.295×10^{-1}
2.305×10^{-1}	2.315×10^{-1}	2.324×10^{-1}	2.334×10^{-1}	2.344×10^{-1}
2.354×10^{-1}	2.363×10^{-1}	2.373×10^{-1}	2.383×10^{-1}	2.393×10^{-1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
2.403×10^{-1}	2.412×10^{-1}	2.422×10^{-1}	2.432×10^{-1}	2.442×10^{-1}
2.451×10^{-1}	2.461×10^{-1}	2.471×10^{-1}	2.481×10^{-1}	2.49×10^{-1}
2.5×10^{-1}	2.51×10^{-1}	2.52×10^{-1}	2.53×10^{-1}	2.539×10^{-1}
2.549×10^{-1}	2.559×10^{-1}	2.569×10^{-1}	2.578×10^{-1}	2.588×10^{-1}
2.598×10^{-1}	2.608×10^{-1}	2.617×10^{-1}	2.627×10^{-1}	2.637×10^{-1}
2.647×10^{-1}	2.656×10^{-1}	2.666×10^{-1}	2.676×10^{-1}	2.686×10^{-1}
2.696×10^{-1}	2.705×10^{-1}	2.715×10^{-1}	2.725×10^{-1}	2.735×10^{-1}
2.744×10^{-1}	2.754×10^{-1}	2.764×10^{-1}	2.774×10^{-1}	2.783×10^{-1}
2.793×10^{-1}	2.803×10^{-1}	2.813×10^{-1}	2.822×10^{-1}	2.832×10^{-1}
2.842×10^{-1}	2.852×10^{-1}	2.862×10^{-1}	2.871×10^{-1}	2.881×10^{-1}
2.891×10^{-1}	2.901×10^{-1}	2.91×10^{-1}	2.92×10^{-1}	2.93×10^{-1}
2.94×10^{-1}	2.949×10^{-1}	2.959×10^{-1}	2.969×10^{-1}	2.979×10^{-1}
2.988×10^{-1}	2.998×10^{-1}	3.008×10^{-1}	3.018×10^{-1}	3.028×10^{-1}
3.037×10^{-1}	3.047×10^{-1}	3.057×10^{-1}	3.067×10^{-1}	3.076×10^{-1}
3.086×10^{-1}	3.096×10^{-1}	3.106×10^{-1}	3.115×10^{-1}	3.125×10^{-1}
3.135×10^{-1}	3.145×10^{-1}	3.155×10^{-1}	3.164×10^{-1}	3.174×10^{-1}
3.184×10^{-1}	3.194×10^{-1}	3.203×10^{-1}	3.213×10^{-1}	3.223×10^{-1}
3.233×10^{-1}	3.242×10^{-1}	3.252×10^{-1}	3.262×10^{-1}	3.272×10^{-1}
3.281×10^{-1}	3.291×10^{-1}	3.301×10^{-1}	3.311×10^{-1}	3.321×10^{-1}
3.33×10^{-1}	3.34×10^{-1}	3.35×10^{-1}	3.36×10^{-1}	3.369×10^{-1}
3.379×10^{-1}	3.389×10^{-1}	3.399×10^{-1}	3.408×10^{-1}	3.418×10^{-1}
3.428×10^{-1}	3.438×10^{-1}	3.447×10^{-1}	3.457×10^{-1}	3.467×10^{-1}
3.477×10^{-1}	3.487×10^{-1}	3.496×10^{-1}	3.506×10^{-1}	3.516×10^{-1}
3.526×10^{-1}	3.535×10^{-1}	3.545×10^{-1}	3.555×10^{-1}	3.565×10^{-1}
3.574×10^{-1}	3.584×10^{-1}	3.594×10^{-1}	3.604×10^{-1}	3.613×10^{-1}
3.623×10^{-1}	3.633×10^{-1}	3.643×10^{-1}	3.653×10^{-1}	3.662×10^{-1}
3.672×10^{-1}	3.682×10^{-1}	3.692×10^{-1}	3.701×10^{-1}	3.711×10^{-1}
3.721×10^{-1}	3.731×10^{-1}	3.74×10^{-1}	3.75×10^{-1}	3.76×10^{-1}
3.77×10^{-1}	3.78×10^{-1}	3.789×10^{-1}	3.799×10^{-1}	3.809×10^{-1}
3.819×10^{-1}	3.828×10^{-1}	3.838×10^{-1}	3.848×10^{-1}	3.858×10^{-1}
3.867×10^{-1}	3.877×10^{-1}	3.887×10^{-1}	3.897×10^{-1}	3.906×10^{-1}
3.916×10^{-1}	3.926×10^{-1}	3.936×10^{-1}	3.946×10^{-1}	3.955×10^{-1}
3.965×10^{-1}	3.975×10^{-1}	3.985×10^{-1}	3.994×10^{-1}	4.004×10^{-1}
4.014×10^{-1}	4.024×10^{-1}	4.033×10^{-1}	4.043×10^{-1}	4.053×10^{-1}
4.063×10^{-1}	4.072×10^{-1}	4.082×10^{-1}	4.092×10^{-1}	4.102×10^{-1}
4.112×10^{-1}	4.121×10^{-1}	4.131×10^{-1}	4.141×10^{-1}	4.151×10^{-1}
4.16×10^{-1}	4.17×10^{-1}	4.18×10^{-1}	4.19×10^{-1}	4.199×10^{-1}
4.209×10^{-1}	4.219×10^{-1}	4.229×10^{-1}	4.238×10^{-1}	4.248×10^{-1}
4.258×10^{-1}	4.268×10^{-1}	4.278×10^{-1}	4.287×10^{-1}	4.297×10^{-1}
4.307×10^{-1}	4.317×10^{-1}	4.326×10^{-1}	4.336×10^{-1}	4.346×10^{-1}
4.356×10^{-1}	4.365×10^{-1}	4.375×10^{-1}	4.385×10^{-1}	4.395×10^{-1}
4.404×10^{-1}	4.414×10^{-1}	4.424×10^{-1}	4.434×10^{-1}	4.444×10^{-1}
4.453×10^{-1}	4.463×10^{-1}	4.473×10^{-1}	4.483×10^{-1}	4.492×10^{-1}
4.502×10^{-1}	4.512×10^{-1}	4.522×10^{-1}	4.531×10^{-1}	4.541×10^{-1}
4.551×10^{-1}	4.561×10^{-1}	4.571×10^{-1}	4.58×10^{-1}	4.59×10^{-1}
4.6×10^{-1}	4.61×10^{-1}	4.619×10^{-1}	4.629×10^{-1}	4.639×10^{-1}
4.649×10^{-1}	4.658×10^{-1}	4.668×10^{-1}	4.678×10^{-1}	4.688×10^{-1}
4.697×10^{-1}	4.707×10^{-1}	4.717×10^{-1}	4.727×10^{-1}	4.737×10^{-1}
4.746×10^{-1}	4.756×10^{-1}	4.766×10^{-1}	4.776×10^{-1}	4.785×10^{-1}
4.795×10^{-1}	4.805×10^{-1}	4.815×10^{-1}	4.824×10^{-1}	4.834×10^{-1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
4.844×10^{-1}	4.854×10^{-1}	4.863×10^{-1}	4.873×10^{-1}	4.883×10^{-1}
4.893×10^{-1}	4.903×10^{-1}	4.912×10^{-1}	4.922×10^{-1}	4.932×10^{-1}
4.942×10^{-1}	4.951×10^{-1}	4.961×10^{-1}	4.971×10^{-1}	4.981×10^{-1}
4.99×10^{-1}	5×10^{-1}	5.01×10^{-1}	5.02×10^{-1}	5.029×10^{-1}
5.039×10^{-1}	5.049×10^{-1}	5.059×10^{-1}	5.069×10^{-1}	5.078×10^{-1}
5.088×10^{-1}	5.098×10^{-1}	5.108×10^{-1}	5.117×10^{-1}	5.127×10^{-1}
5.137×10^{-1}	5.147×10^{-1}	5.156×10^{-1}	5.166×10^{-1}	5.176×10^{-1}
5.186×10^{-1}	5.196×10^{-1}	5.205×10^{-1}	5.215×10^{-1}	5.225×10^{-1}
5.235×10^{-1}	5.244×10^{-1}	5.254×10^{-1}	5.264×10^{-1}	5.274×10^{-1}
5.283×10^{-1}	5.293×10^{-1}	5.303×10^{-1}	5.313×10^{-1}	5.322×10^{-1}
5.332×10^{-1}	5.342×10^{-1}	5.352×10^{-1}	5.362×10^{-1}	5.371×10^{-1}
5.381×10^{-1}	5.391×10^{-1}	5.401×10^{-1}	5.41×10^{-1}	5.42×10^{-1}
5.43×10^{-1}	5.44×10^{-1}	5.449×10^{-1}	5.459×10^{-1}	5.469×10^{-1}
5.479×10^{-1}	5.488×10^{-1}	5.498×10^{-1}	5.508×10^{-1}	5.518×10^{-1}
5.528×10^{-1}	5.537×10^{-1}	5.547×10^{-1}	5.557×10^{-1}	5.567×10^{-1}
5.576×10^{-1}	5.586×10^{-1}	5.596×10^{-1}	5.606×10^{-1}	5.615×10^{-1}
5.625×10^{-1}	5.635×10^{-1}	5.645×10^{-1}	5.654×10^{-1}	5.664×10^{-1}
5.674×10^{-1}	5.684×10^{-1}	5.694×10^{-1}	5.703×10^{-1}	5.713×10^{-1}
5.723×10^{-1}	5.733×10^{-1}	5.742×10^{-1}	5.752×10^{-1}	5.762×10^{-1}
5.772×10^{-1}	5.781×10^{-1}	5.791×10^{-1}	5.801×10^{-1}	5.811×10^{-1}
5.821×10^{-1}	5.83×10^{-1}	5.84×10^{-1}	5.85×10^{-1}	5.86×10^{-1}
5.869×10^{-1}	5.879×10^{-1}	5.889×10^{-1}	5.899×10^{-1}	5.908×10^{-1}
5.918×10^{-1}	5.928×10^{-1}	5.938×10^{-1}	5.947×10^{-1}	5.957×10^{-1}
5.967×10^{-1}	5.977×10^{-1}	5.987×10^{-1}	5.996×10^{-1}	6.006×10^{-1}
6.016×10^{-1}	6.026×10^{-1}	6.035×10^{-1}	6.045×10^{-1}	6.055×10^{-1}
6.065×10^{-1}	6.074×10^{-1}	6.084×10^{-1}	6.094×10^{-1}	6.104×10^{-1}
6.113×10^{-1}	6.123×10^{-1}	6.133×10^{-1}	6.143×10^{-1}	6.153×10^{-1}
6.162×10^{-1}	6.172×10^{-1}	6.182×10^{-1}	6.192×10^{-1}	6.201×10^{-1}
6.211×10^{-1}	6.221×10^{-1}	6.231×10^{-1}	6.24×10^{-1}	6.25×10^{-1}
6.26×10^{-1}	6.27×10^{-1}	6.279×10^{-1}	6.289×10^{-1}	6.299×10^{-1}
6.309×10^{-1}	6.319×10^{-1}	6.328×10^{-1}	6.338×10^{-1}	6.348×10^{-1}
6.358×10^{-1}	6.367×10^{-1}	6.377×10^{-1}	6.387×10^{-1}	6.397×10^{-1}
6.406×10^{-1}	6.416×10^{-1}	6.426×10^{-1}	6.436×10^{-1}	6.445×10^{-1}
6.455×10^{-1}	6.465×10^{-1}	6.475×10^{-1}	6.485×10^{-1}	6.494×10^{-1}
6.504×10^{-1}	6.514×10^{-1}	6.524×10^{-1}	6.533×10^{-1}	6.543×10^{-1}
6.553×10^{-1}	6.563×10^{-1}	6.572×10^{-1}	6.582×10^{-1}	6.592×10^{-1}
6.602×10^{-1}	6.612×10^{-1}	6.621×10^{-1}	6.631×10^{-1}	6.641×10^{-1}
6.651×10^{-1}	6.66×10^{-1}	6.67×10^{-1}	6.68×10^{-1}	6.69×10^{-1}
6.699×10^{-1}	6.709×10^{-1}	6.719×10^{-1}	6.729×10^{-1}	6.738×10^{-1}
6.748×10^{-1}	6.758×10^{-1}	6.768×10^{-1}	6.778×10^{-1}	6.787×10^{-1}
6.797×10^{-1}	6.807×10^{-1}	6.817×10^{-1}	6.826×10^{-1}	6.836×10^{-1}
6.846×10^{-1}	6.856×10^{-1}	6.865×10^{-1}	6.875×10^{-1}	6.885×10^{-1}
6.895×10^{-1}	6.904×10^{-1}	6.914×10^{-1}	6.924×10^{-1}	6.934×10^{-1}
6.944×10^{-1}	6.953×10^{-1}	6.963×10^{-1}	6.973×10^{-1}	6.983×10^{-1}
6.992×10^{-1}	7.002×10^{-1}	7.012×10^{-1}	7.022×10^{-1}	7.031×10^{-1}
7.041×10^{-1}	7.051×10^{-1}	7.061×10^{-1}	7.07×10^{-1}	7.08×10^{-1}
7.09×10^{-1}	7.1×10^{-1}	7.11×10^{-1}	7.119×10^{-1}	7.129×10^{-1}
7.139×10^{-1}	7.149×10^{-1}	7.158×10^{-1}	7.168×10^{-1}	7.178×10^{-1}
7.188×10^{-1}	7.197×10^{-1}	7.207×10^{-1}	7.217×10^{-1}	7.227×10^{-1}
7.237×10^{-1}	7.246×10^{-1}	7.256×10^{-1}	7.266×10^{-1}	7.276×10^{-1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
7.285×10^{-1}	7.295×10^{-1}	7.305×10^{-1}	7.315×10^{-1}	7.324×10^{-1}
7.334×10^{-1}	7.344×10^{-1}	7.354×10^{-1}	7.363×10^{-1}	7.373×10^{-1}
7.383×10^{-1}	7.393×10^{-1}	7.403×10^{-1}	7.412×10^{-1}	7.422×10^{-1}
7.432×10^{-1}	7.442×10^{-1}	7.451×10^{-1}	7.461×10^{-1}	7.471×10^{-1}
7.481×10^{-1}	7.49×10^{-1}	7.5×10^{-1}	7.51×10^{-1}	7.52×10^{-1}
7.529×10^{-1}	7.539×10^{-1}	7.549×10^{-1}	7.559×10^{-1}	7.569×10^{-1}
7.578×10^{-1}	7.588×10^{-1}	7.598×10^{-1}	7.608×10^{-1}	7.617×10^{-1}
7.627×10^{-1}	7.637×10^{-1}	7.647×10^{-1}	7.656×10^{-1}	7.666×10^{-1}
7.676×10^{-1}	7.686×10^{-1}	7.695×10^{-1}	7.705×10^{-1}	7.715×10^{-1}
7.725×10^{-1}	7.735×10^{-1}	7.744×10^{-1}	7.754×10^{-1}	7.764×10^{-1}
7.774×10^{-1}	7.783×10^{-1}	7.793×10^{-1}	7.803×10^{-1}	7.813×10^{-1}
7.822×10^{-1}	7.832×10^{-1}	7.842×10^{-1}	7.852×10^{-1}	7.862×10^{-1}
7.871×10^{-1}	7.881×10^{-1}	7.891×10^{-1}	7.901×10^{-1}	7.91×10^{-1}
7.92×10^{-1}	7.93×10^{-1}	7.94×10^{-1}	7.949×10^{-1}	7.959×10^{-1}
7.969×10^{-1}	7.979×10^{-1}	7.988×10^{-1}	7.998×10^{-1}	8.008×10^{-1}
8.018×10^{-1}	8.028×10^{-1}	8.037×10^{-1}	8.047×10^{-1}	8.057×10^{-1}
8.067×10^{-1}	8.076×10^{-1}	8.086×10^{-1}	8.096×10^{-1}	8.106×10^{-1}
8.115×10^{-1}	8.125×10^{-1}	8.135×10^{-1}	8.145×10^{-1}	8.154×10^{-1}
8.164×10^{-1}	8.174×10^{-1}	8.184×10^{-1}	8.194×10^{-1}	8.203×10^{-1}
8.213×10^{-1}	8.223×10^{-1}	8.233×10^{-1}	8.242×10^{-1}	8.252×10^{-1}
8.262×10^{-1}	8.272×10^{-1}	8.281×10^{-1}	8.291×10^{-1}	8.301×10^{-1}
8.311×10^{-1}	8.32×10^{-1}	8.33×10^{-1}	8.34×10^{-1}	8.35×10^{-1}
8.36×10^{-1}	8.369×10^{-1}	8.379×10^{-1}	8.389×10^{-1}	8.399×10^{-1}
8.408×10^{-1}	8.418×10^{-1}	8.428×10^{-1}	8.438×10^{-1}	8.447×10^{-1}
8.457×10^{-1}	8.467×10^{-1}	8.477×10^{-1}	8.486×10^{-1}	8.496×10^{-1}
8.506×10^{-1}	8.516×10^{-1}	8.526×10^{-1}	8.535×10^{-1}	8.545×10^{-1}
8.555×10^{-1}	8.565×10^{-1}	8.574×10^{-1}	8.584×10^{-1}	8.594×10^{-1}
8.604×10^{-1}	8.613×10^{-1}	8.623×10^{-1}	8.633×10^{-1}	8.643×10^{-1}
8.653×10^{-1}	8.662×10^{-1}	8.672×10^{-1}	8.682×10^{-1}	8.692×10^{-1}
8.701×10^{-1}	8.711×10^{-1}	8.721×10^{-1}	8.731×10^{-1}	8.74×10^{-1}
8.75×10^{-1}	8.76×10^{-1}	8.77×10^{-1}	8.779×10^{-1}	8.789×10^{-1}
8.799×10^{-1}	8.809×10^{-1}	8.819×10^{-1}	8.828×10^{-1}	8.838×10^{-1}
8.848×10^{-1}	8.858×10^{-1}	8.867×10^{-1}	8.877×10^{-1}	8.887×10^{-1}
8.897×10^{-1}	8.906×10^{-1}	8.916×10^{-1}	8.926×10^{-1}	8.936×10^{-1}
8.945×10^{-1}	8.955×10^{-1}	8.965×10^{-1}	8.975×10^{-1}	8.985×10^{-1}
8.994×10^{-1}	9.004×10^{-1}	9.014×10^{-1}	9.024×10^{-1}	9.033×10^{-1}
9.043×10^{-1}	9.053×10^{-1}	9.063×10^{-1}	9.072×10^{-1}	9.082×10^{-1}
9.092×10^{-1}	9.102×10^{-1}	9.111×10^{-1}	9.121×10^{-1}	9.131×10^{-1}
9.141×10^{-1}	9.151×10^{-1}	9.16×10^{-1}	9.17×10^{-1}	9.18×10^{-1}
9.19×10^{-1}	9.199×10^{-1}	9.209×10^{-1}	9.219×10^{-1}	9.229×10^{-1}
9.238×10^{-1}	9.248×10^{-1}	9.258×10^{-1}	9.268×10^{-1}	9.278×10^{-1}
9.287×10^{-1}	9.297×10^{-1}	9.307×10^{-1}	9.317×10^{-1}	9.326×10^{-1}
9.336×10^{-1}	9.346×10^{-1}	9.356×10^{-1}	9.365×10^{-1}	9.375×10^{-1}
9.385×10^{-1}	9.395×10^{-1}	9.404×10^{-1}	9.414×10^{-1}	9.424×10^{-1}
9.434×10^{-1}	9.444×10^{-1}	9.453×10^{-1}	9.463×10^{-1}	9.473×10^{-1}
9.483×10^{-1}	9.492×10^{-1}	9.502×10^{-1}	9.512×10^{-1}	9.522×10^{-1}
9.531×10^{-1}	9.541×10^{-1}	9.551×10^{-1}	9.561×10^{-1}	9.57×10^{-1}
9.58×10^{-1}	9.59×10^{-1}	9.6×10^{-1}	9.61×10^{-1}	9.619×10^{-1}
9.629×10^{-1}	9.639×10^{-1}	9.649×10^{-1}	9.658×10^{-1}	9.668×10^{-1}
9.678×10^{-1}	9.688×10^{-1}	9.697×10^{-1}	9.707×10^{-1}	9.717×10^{-1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
9.727×10^{-1}	9.736×10^{-1}	9.746×10^{-1}	9.756×10^{-1}	9.766×10^{-1}
9.776×10^{-1}	9.785×10^{-1}	9.795×10^{-1}	9.805×10^{-1}	9.815×10^{-1}
9.824×10^{-1}	9.834×10^{-1}	9.844×10^{-1}	9.854×10^{-1}	9.863×10^{-1}
9.873×10^{-1}	9.883×10^{-1}	9.893×10^{-1}	9.903×10^{-1}	9.912×10^{-1}
9.922×10^{-1}	9.932×10^{-1}	9.942×10^{-1}	9.951×10^{-1}	9.961×10^{-1}
9.971×10^{-1}	9.981×10^{-1}	9.99×10^{-1}	1	1.001
1.002	1.003	1.004	1.005	1.006
1.007	1.008	1.009	1.01	1.011
1.012	1.013	1.014	1.015	1.016
1.017	1.018	1.019	1.02	1.021
1.021	1.022	1.023	1.024	1.025
1.026	1.027	1.028	1.029	1.03
1.031	1.032	1.033	1.034	1.035
1.036	1.037	1.038	1.039	1.04
1.041	1.042	1.043	1.044	1.045
1.046	1.047	1.048	1.049	1.05
1.051	1.052	1.053	1.054	1.055
1.056	1.057	1.058	1.059	1.06
1.061	1.062	1.063	1.063	1.064
1.065	1.066	1.067	1.068	1.069
1.07	1.071	1.072	1.073	1.074
1.075	1.076	1.077	1.078	1.079
1.08	1.081	1.082	1.083	1.084
1.085	1.086	1.087	1.088	1.089
1.09	1.091	1.092	1.093	1.094
1.095	1.096	1.097	1.098	1.099
1.1	1.101	1.102	1.103	1.104
1.105	1.105	1.106	1.107	1.108
1.109	1.11	1.111	1.112	1.113
1.114	1.115	1.116	1.117	1.118
1.119	1.12	1.121	1.122	1.123
1.124	1.125	1.126	1.127	1.128
1.129	1.13	1.131	1.132	1.133
1.134	1.135	1.136	1.137	1.138
1.139	1.14	1.141	1.142	1.143
1.144	1.145	1.146	1.146	1.147
1.148	1.149	1.15	1.151	1.152
1.153	1.154	1.155	1.156	1.157
1.158	1.159	1.16	1.161	1.162
1.163	1.164	1.165	1.166	1.167
1.168	1.169	1.17	1.171	1.172
1.173	1.174	1.175	1.176	1.177
1.178	1.179	1.18	1.181	1.182
1.183	1.184	1.185	1.186	1.187
1.188	1.188	1.189	1.19	1.191
1.192	1.193	1.194	1.195	1.196
1.197	1.198	1.199	1.2	1.201
1.202	1.203	1.204	1.205	1.206
1.207	1.208	1.209	1.21	1.211
1.212	1.213	1.214	1.215	1.216

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
1.217	1.218	1.219	1.22	1.221
1.222	1.223	1.224	1.225	1.226
1.227	1.228	1.229	1.23	1.23
1.231	1.232	1.233	1.234	1.235
1.236	1.237	1.238	1.239	1.24
1.241	1.242	1.243	1.244	1.245
1.246	1.247	1.248	1.249	1.25
1.251	1.252	1.253	1.254	1.255
1.256	1.257	1.258	1.259	1.26
1.261	1.262	1.263	1.264	1.265
1.266	1.267	1.268	1.269	1.27
1.271	1.271	1.272	1.273	1.274
1.275	1.276	1.277	1.278	1.279
1.28	1.281	1.282	1.283	1.284
1.285	1.286	1.287	1.288	1.289
1.29	1.291	1.292	1.293	1.294
1.295	1.296	1.297	1.298	1.299
1.3	1.301	1.302	1.303	1.304
1.305	1.306	1.307	1.308	1.309
1.31	1.311	1.312	1.313	1.313
1.314	1.315	1.316	1.317	1.318
1.319	1.32	1.321	1.322	1.323
1.324	1.325	1.326	1.327	1.328
1.329	1.33	1.331	1.332	1.333
1.334	1.335	1.336	1.337	1.338
1.339	1.34	1.341	1.342	1.343
1.344	1.345	1.346	1.347	1.348
1.349	1.35	1.351	1.352	1.353
1.354	1.355	1.355	1.356	1.357
1.358	1.359	1.36	1.361	1.362
1.363	1.364	1.365	1.366	1.367
1.368	1.369	1.37	1.371	1.372
1.373	1.374	1.375	1.376	1.377
1.378	1.379	1.38	1.381	1.382
1.383	1.384	1.385	1.386	1.387
1.388	1.389	1.39	1.391	1.392
1.393	1.394	1.395	1.396	1.396
1.397	1.398	1.399	1.4	1.401
1.402	1.403	1.404	1.405	1.406
1.407	1.408	1.409	1.41	1.411
1.412	1.413	1.414	1.415	1.416
1.417	1.418	1.419	1.42	1.421
1.422	1.423	1.424	1.425	1.426
1.427	1.428	1.429	1.43	1.431
1.432	1.433	1.434	1.435	1.436
1.437	1.438	1.438	1.439	1.44
1.441	1.442	1.443	1.444	1.445
1.446	1.447	1.448	1.449	1.45
1.451	1.452	1.453	1.454	1.455
1.456	1.457	1.458	1.459	1.46

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
1.461	1.462	1.463	1.464	1.465
1.466	1.467	1.468	1.469	1.47
1.471	1.472	1.473	1.474	1.475
1.476	1.477	1.478	1.479	1.48
1.48	1.481	1.482	1.483	1.484
1.485	1.486	1.487	1.488	1.489
1.49	1.491	1.492	1.493	1.494
1.495	1.496	1.497	1.498	1.499
1.5	1.501	1.502	1.503	1.504
1.505	1.506	1.507	1.508	1.509
1.51	1.511	1.512	1.513	1.514
1.515	1.516	1.517	1.518	1.519
1.52	1.521	1.521	1.522	1.523
1.524	1.525	1.526	1.527	1.528
1.529	1.53	1.531	1.532	1.533
1.534	1.535	1.536	1.537	1.538
1.539	1.54	1.541	1.542	1.543
1.544	1.545	1.546	1.547	1.548
1.549	1.55	1.551	1.552	1.553
1.554	1.555	1.556	1.557	1.558
1.559	1.56	1.561	1.562	1.563
1.563	1.564	1.565	1.566	1.567
1.568	1.569	1.57	1.571	1.572
1.573	1.574	1.575	1.576	1.577
1.578	1.579	1.58	1.581	1.582
1.583	1.584	1.585	1.586	1.587
1.588	1.589	1.59	1.591	1.592
1.593	1.594	1.595	1.596	1.597
1.598	1.599	1.6	1.601	1.602
1.603	1.604	1.605	1.605	1.606
1.607	1.608	1.609	1.61	1.611
1.612	1.613	1.614	1.615	1.616
1.617	1.618	1.619	1.62	1.621
1.622	1.623	1.624	1.625	1.626
1.627	1.628	1.629	1.63	1.631
1.632	1.633	1.634	1.635	1.636
1.637	1.638	1.639	1.64	1.641
1.642	1.643	1.644	1.645	1.646
1.646	1.647	1.648	1.649	1.65
1.651	1.652	1.653	1.654	1.655
1.656	1.657	1.658	1.659	1.66
1.661	1.662	1.663	1.664	1.665
1.666	1.667	1.668	1.669	1.67
1.671	1.672	1.673	1.674	1.675
1.676	1.677	1.678	1.679	1.68
1.681	1.682	1.683	1.684	1.685
1.686	1.687	1.688	1.688	1.689
1.69	1.691	1.692	1.693	1.694
1.695	1.696	1.697	1.698	1.699
1.7	1.701	1.702	1.703	1.704

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
1.705	1.706	1.707	1.708	1.709
1.71	1.711	1.712	1.713	1.714
1.715	1.716	1.717	1.718	1.719
1.72	1.721	1.722	1.723	1.724
1.725	1.726	1.727	1.728	1.729
1.73	1.73	1.731	1.732	1.733
1.734	1.735	1.736	1.737	1.738
1.739	1.74	1.741	1.742	1.743
1.744	1.745	1.746	1.747	1.748
1.749	1.75	1.751	1.752	1.753
1.754	1.755	1.756	1.757	1.758
1.759	1.76	1.761	1.762	1.763
1.764	1.765	1.766	1.767	1.768
1.769	1.77	1.771	1.771	1.772
1.773	1.774	1.775	1.776	1.777
1.778	1.779	1.78	1.781	1.782
1.783	1.784	1.785	1.786	1.787
1.788	1.789	1.79	1.791	1.792
1.793	1.794	1.795	1.796	1.797
1.798	1.799	1.8	1.801	1.802
1.803	1.804	1.805	1.806	1.807
1.808	1.809	1.81	1.811	1.812
1.813	1.813	1.814	1.815	1.816
1.817	1.818	1.819	1.82	1.821
1.822	1.823	1.824	1.825	1.826
1.827	1.828	1.829	1.83	1.831
1.832	1.833	1.834	1.835	1.836
1.837	1.838	1.839	1.84	1.841
1.842	1.843	1.844	1.845	1.846
1.847	1.848	1.849	1.85	1.851
1.852	1.853	1.854	1.855	1.855
1.856	1.857	1.858	1.859	1.86
1.861	1.862	1.863	1.864	1.865
1.866	1.867	1.868	1.869	1.87
1.871	1.872	1.873	1.874	1.875
1.876	1.877	1.878	1.879	1.88
1.881	1.882	1.883	1.884	1.885
1.886	1.887	1.888	1.889	1.89
1.891	1.892	1.893	1.894	1.895
1.896	1.896	1.897	1.898	1.899
1.9	1.901	1.902	1.903	1.904
1.905	1.906	1.907	1.908	1.909
1.91	1.911	1.912	1.913	1.914
1.915	1.916	1.917	1.918	1.919
1.92	1.921	1.922	1.923	1.924
1.925	1.926	1.927	1.928	1.929
1.93	1.931	1.932	1.933	1.934
1.935	1.936	1.937	1.938	1.938
1.939	1.94	1.941	1.942	1.943
1.944	1.945	1.946	1.947	1.948

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
1.949	1.95	1.951	1.952	1.953
1.954	1.955	1.956	1.957	1.958
1.959	1.96	1.961	1.962	1.963
1.964	1.965	1.966	1.967	1.968
1.969	1.97	1.971	1.972	1.973
1.974	1.975	1.976	1.977	1.978
1.979	1.98	1.98	1.981	1.982
1.983	1.984	1.985	1.986	1.987
1.988	1.989	1.99	1.991	1.992
1.993	1.994	1.995	1.996	1.997
1.998	1.999	2	2.001	2.002
2.003	2.004	2.005	2.006	2.007
2.008	2.009	2.01	2.011	2.012
2.013	2.014	2.015	2.016	2.017
2.018	2.019	2.02	2.021	2.021
2.022	2.023	2.024	2.025	2.026
2.027	2.028	2.029	2.03	2.031
2.032	2.033	2.034	2.035	2.036
2.037	2.038	2.039	2.04	2.041
2.042	2.043	2.044	2.045	2.046
2.047	2.048	2.049	2.05	2.051
2.052	2.053	2.054	2.055	2.056
2.057	2.058	2.059	2.06	2.061
2.062	2.063	2.063	2.064	2.065
2.066	2.067	2.068	2.069	2.07
2.071	2.072	2.073	2.074	2.075
2.076	2.077	2.078	2.079	2.08
2.081	2.082	2.083	2.084	2.085
2.086	2.087	2.088	2.089	2.09
2.091	2.092	2.093	2.094	2.095
2.096	2.097	2.098	2.099	2.1
2.101	2.102	2.103	2.104	2.104
2.105	2.106	2.107	2.108	2.109
2.11	2.111	2.112	2.113	2.114
2.115	2.116	2.117	2.118	2.119
2.12	2.121	2.122	2.123	2.124
2.125	2.126	2.127	2.128	2.129
2.13	2.131	2.132	2.133	2.134
2.135	2.136	2.137	2.138	2.139
2.14	2.141	2.142	2.143	2.144
2.145	2.146	2.146	2.147	2.148
2.149	2.15	2.151	2.152	2.153
2.154	2.155	2.156	2.157	2.158
2.159	2.16	2.161	2.162	2.163
2.164	2.165	2.166	2.167	2.168
2.169	2.17	2.171	2.172	2.173
2.174	2.175	2.176	2.177	2.178
2.179	2.18	2.181	2.182	2.183
2.184	2.185	2.186	2.187	2.188
2.188	2.189	2.19	2.191	2.192

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
2.193	2.194	2.195	2.196	2.197
2.198	2.199	2.2	2.201	2.202
2.203	2.204	2.205	2.206	2.207
2.208	2.209	2.21	2.211	2.212
2.213	2.214	2.215	2.216	2.217
2.218	2.219	2.22	2.221	2.222
2.223	2.224	2.225	2.226	2.227
2.228	2.229	2.229	2.23	2.231
2.232	2.233	2.234	2.235	2.236
2.237	2.238	2.239	2.24	2.241
2.242	2.243	2.244	2.245	2.246
2.247	2.248	2.249	2.25	2.251
2.252	2.253	2.254	2.255	2.256
2.257	2.258	2.259	2.26	2.261
2.262	2.263	2.264	2.265	2.266
2.267	2.268	2.269	2.27	2.271
2.271	2.272	2.273	2.274	2.275
2.276	2.277	2.278	2.279	2.28
2.281	2.282	2.283	2.284	2.285
2.286	2.287	2.288	2.289	2.29
2.291	2.292	2.293	2.294	2.295
2.296	2.297	2.298	2.299	2.3
2.301	2.302	2.303	2.304	2.305
2.306	2.307	2.308	2.309	2.31
2.311	2.312	2.313	2.313	2.314
2.315	2.316	2.317	2.318	2.319
2.32	2.321	2.322	2.323	2.324
2.325	2.326	2.327	2.328	2.329
2.33	2.331	2.332	2.333	2.334
2.335	2.336	2.337	2.338	2.339
2.34	2.341	2.342	2.343	2.344
2.345	2.346	2.347	2.348	2.349
2.35	2.351	2.352	2.353	2.354
2.354	2.355	2.356	2.357	2.358
2.359	2.36	2.361	2.362	2.363
2.364	2.365	2.366	2.367	2.368
2.369	2.37	2.371	2.372	2.373
2.374	2.375	2.376	2.377	2.378
2.379	2.38	2.381	2.382	2.383
2.384	2.385	2.386	2.387	2.388
2.389	2.39	2.391	2.392	2.393
2.394	2.395	2.396	2.396	2.397
2.398	2.399	2.4	2.401	2.402
2.403	2.404	2.405	2.406	2.407
2.408	2.409	2.41	2.411	2.412
2.413	2.414	2.415	2.416	2.417
2.418	2.419	2.42	2.421	2.422
2.423	2.424	2.425	2.426	2.427
2.428	2.429	2.43	2.431	2.432
2.433	2.434	2.435	2.436	2.437

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
2.438	2.438	2.439	2.44	2.441
2.442	2.443	2.444	2.445	2.446
2.447	2.448	2.449	2.45	2.451
2.452	2.453	2.454	2.455	2.456
2.457	2.458	2.459	2.46	2.461
2.462	2.463	2.464	2.465	2.466
2.467	2.468	2.469	2.47	2.471
2.472	2.473	2.474	2.475	2.476
2.477	2.478	2.479	2.479	2.48
2.481	2.482	2.483	2.484	2.485
2.486	2.487	2.488	2.489	2.49
2.491	2.492	2.493	2.494	2.495
2.496	2.497	2.498	2.499	2.5
2.501	2.502	2.503	2.504	2.505
2.506	2.507	2.508	2.509	2.51
2.511	2.512	2.513	2.514	2.515
2.516	2.517	2.518	2.519	2.52
2.521	2.521	2.522	2.523	2.524
2.525	2.526	2.527	2.528	2.529
2.53	2.531	2.532	2.533	2.534
2.535	2.536	2.537	2.538	2.539
2.54	2.541	2.542	2.543	2.544
2.545	2.546	2.547	2.548	2.549
2.55	2.551	2.552	2.553	2.554
2.555	2.556	2.557	2.558	2.559
2.56	2.561	2.562	2.563	2.563
2.564	2.565	2.566	2.567	2.568
2.569	2.57	2.571	2.572	2.573
2.574	2.575	2.576	2.577	2.578
2.579	2.58	2.581	2.582	2.583
2.584	2.585	2.586	2.587	2.588
2.589	2.59	2.591	2.592	2.593
2.594	2.595	2.596	2.597	2.598
2.599	2.6	2.601	2.602	2.603
2.604	2.604	2.605	2.606	2.607
2.608	2.609	2.61	2.611	2.612
2.613	2.614	2.615	2.616	2.617
2.618	2.619	2.62	2.621	2.622
2.623	2.624	2.625	2.626	2.627
2.628	2.629	2.63	2.631	2.632
2.633	2.634	2.635	2.636	2.637
2.638	2.639	2.64	2.641	2.642
2.643	2.644	2.645	2.646	2.646
2.647	2.648	2.649	2.65	2.651
2.652	2.653	2.654	2.655	2.656
2.657	2.658	2.659	2.66	2.661
2.662	2.663	2.664	2.665	2.666
2.667	2.668	2.669	2.67	2.671
2.672	2.673	2.674	2.675	2.676
2.677	2.678	2.679	2.68	2.681

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
2.682	2.683	2.684	2.685	2.686
2.687	2.688	2.688	2.689	2.69
2.691	2.692	2.693	2.694	2.695
2.696	2.697	2.698	2.699	2.7
2.701	2.702	2.703	2.704	2.705
2.706	2.707	2.708	2.709	2.71
2.711	2.712	2.713	2.714	2.715
2.716	2.717	2.718	2.719	2.72
2.721	2.722	2.723	2.724	2.725
2.726	2.727	2.728	2.729	2.729
2.73	2.731	2.732	2.733	2.734
2.735	2.736	2.737	2.738	2.739
2.74	2.741	2.742	2.743	2.744
2.745	2.746	2.747	2.748	2.749
2.75	2.751	2.752	2.753	2.754
2.755	2.756	2.757	2.758	2.759
2.76	2.761	2.762	2.763	2.764
2.765	2.766	2.767	2.768	2.769
2.77	2.771	2.771	2.772	2.773
2.774	2.775	2.776	2.777	2.778
2.779	2.78	2.781	2.782	2.783
2.784	2.785	2.786	2.787	2.788
2.789	2.79	2.791	2.792	2.793
2.794	2.795	2.796	2.797	2.798
2.799	2.8	2.801	2.802	2.803
2.804	2.805	2.806	2.807	2.808
2.809	2.81	2.811	2.812	2.813
2.813	2.814	2.815	2.816	2.817
2.818	2.819	2.82	2.821	2.822
2.823	2.824	2.825	2.826	2.827
2.828	2.829	2.83	2.831	2.832
2.833	2.834	2.835	2.836	2.837
2.838	2.839	2.84	2.841	2.842
2.843	2.844	2.845	2.846	2.847
2.848	2.849	2.85	2.851	2.852
2.853	2.854	2.854	2.855	2.856
2.857	2.858	2.859	2.86	2.861
2.862	2.863	2.864	2.865	2.866
2.867	2.868	2.869	2.87	2.871
2.872	2.873	2.874	2.875	2.876
2.877	2.878	2.879	2.88	2.881
2.882	2.883	2.884	2.885	2.886
2.887	2.888	2.889	2.89	2.891
2.892	2.893	2.894	2.895	2.896
2.896	2.897	2.898	2.899	2.9
2.901	2.902	2.903	2.904	2.905
2.906	2.907	2.908	2.909	2.91
2.911	2.912	2.913	2.914	2.915
2.916	2.917	2.918	2.919	2.92
2.921	2.922	2.923	2.924	2.925

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
2.926	2.927	2.928	2.929	2.93
2.931	2.932	2.933	2.934	2.935
2.936	2.937	2.938	2.938	2.939
2.94	2.941	2.942	2.943	2.944
2.945	2.946	2.947	2.948	2.949
2.95	2.951	2.952	2.953	2.954
2.955	2.956	2.957	2.958	2.959
2.96	2.961	2.962	2.963	2.964
2.965	2.966	2.967	2.968	2.969
2.97	2.971	2.972	2.973	2.974
2.975	2.976	2.977	2.978	2.979
2.979	2.98	2.981	2.982	2.983
2.984	2.985	2.986	2.987	2.988
2.989	2.99	2.991	2.992	2.993
2.994	2.995	2.996	2.997	2.998
2.999	3	3.002	3.004	3.006
3.008	3.01	3.012	3.014	3.016
3.018	3.02	3.021	3.023	3.025
3.027	3.029	3.031	3.033	3.035
3.037	3.039	3.041	3.043	3.045
3.047	3.049	3.051	3.053	3.055
3.057	3.059	3.061	3.062	3.064
3.066	3.068	3.07	3.072	3.074
3.076	3.078	3.08	3.082	3.084
3.086	3.088	3.09	3.092	3.094
3.096	3.098	3.1	3.102	3.104
3.105	3.107	3.109	3.111	3.113
3.115	3.117	3.119	3.121	3.123
3.125	3.127	3.129	3.131	3.133
3.135	3.137	3.139	3.141	3.143
3.145	3.146	3.148	3.15	3.152
3.154	3.156	3.158	3.16	3.162
3.164	3.166	3.168	3.17	3.172
3.174	3.176	3.178	3.18	3.182
3.184	3.186	3.188	3.189	3.191
3.193	3.195	3.197	3.199	3.201
3.203	3.205	3.207	3.209	3.211
3.213	3.215	3.217	3.219	3.221
3.223	3.225	3.227	3.229	3.23
3.232	3.234	3.236	3.238	3.24
3.242	3.244	3.246	3.248	3.25
3.252	3.254	3.256	3.258	3.26
3.262	3.264	3.266	3.268	3.27
3.271	3.273	3.275	3.277	3.279
3.281	3.283	3.285	3.287	3.289
3.291	3.293	3.295	3.297	3.299
3.301	3.303	3.305	3.307	3.309
3.311	3.312	3.314	3.316	3.318
3.32	3.322	3.324	3.326	3.328
3.33	3.332	3.334	3.336	3.338

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
3.34	3.342	3.344	3.346	3.348
3.35	3.352	3.354	3.355	3.357
3.359	3.361	3.363	3.365	3.367
3.369	3.371	3.373	3.375	3.377
3.379	3.381	3.383	3.385	3.387
3.389	3.391	3.393	3.395	3.396
3.398	3.4	3.402	3.404	3.406
3.408	3.41	3.412	3.414	3.416
3.418	3.42	3.422	3.424	3.426
3.428	3.43	3.432	3.434	3.436
3.438	3.439	3.441	3.443	3.445
3.447	3.449	3.451	3.453	3.455
3.457	3.459	3.461	3.463	3.465
3.467	3.469	3.471	3.473	3.475
3.477	3.479	3.48	3.482	3.484
3.486	3.488	3.49	3.492	3.494
3.496	3.498	3.5	3.502	3.504
3.506	3.508	3.51	3.512	3.514
3.516	3.518	3.52	3.521	3.523
3.525	3.527	3.529	3.531	3.533
3.535	3.537	3.539	3.541	3.543
3.545	3.547	3.549	3.551	3.553
3.555	3.557	3.559	3.561	3.562
3.564	3.566	3.568	3.57	3.572
3.574	3.576	3.578	3.58	3.582
3.584	3.586	3.588	3.59	3.592
3.594	3.596	3.598	3.6	3.602
3.604	3.605	3.607	3.609	3.611
3.613	3.615	3.617	3.619	3.621
3.623	3.625	3.627	3.629	3.631
3.633	3.635	3.637	3.639	3.641
3.643	3.645	3.646	3.648	3.65
3.652	3.654	3.656	3.658	3.66
3.662	3.664	3.666	3.668	3.67
3.672	3.674	3.676	3.678	3.68
3.682	3.684	3.686	3.688	3.689
3.691	3.693	3.695	3.697	3.699
3.701	3.703	3.705	3.707	3.709
3.711	3.713	3.715	3.717	3.719
3.721	3.723	3.725	3.727	3.729
3.73	3.732	3.734	3.736	3.738
3.74	3.742	3.744	3.746	3.748
3.75	3.752	3.754	3.756	3.758
3.76	3.762	3.764	3.766	3.768
3.77	3.771	3.773	3.775	3.777
3.779	3.781	3.783	3.785	3.787
3.789	3.791	3.793	3.795	3.797
3.799	3.801	3.803	3.805	3.807
3.809	3.811	3.812	3.814	3.816
3.818	3.82	3.822	3.824	3.826

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
3.828	3.83	3.832	3.834	3.836
3.838	3.84	3.842	3.844	3.846
3.848	3.85	3.852	3.854	3.855
3.857	3.859	3.861	3.863	3.865
3.867	3.869	3.871	3.873	3.875
3.877	3.879	3.881	3.883	3.885
3.887	3.889	3.891	3.893	3.895
3.896	3.898	3.9	3.902	3.904
3.906	3.908	3.91	3.912	3.914
3.916	3.918	3.92	3.922	3.924
3.926	3.928	3.93	3.932	3.934
3.936	3.938	3.939	3.941	3.943
3.945	3.947	3.949	3.951	3.953
3.955	3.957	3.959	3.961	3.963
3.965	3.967	3.969	3.971	3.973
3.975	3.977	3.979	3.98	3.982
3.984	3.986	3.988	3.99	3.992
3.994	3.996	3.998	4	4.002
4.004	4.006	4.008	4.01	4.012
4.014	4.016	4.018	4.02	4.021
4.023	4.025	4.027	4.029	4.031
4.033	4.035	4.037	4.039	4.041
4.043	4.045	4.047	4.049	4.051
4.053	4.055	4.057	4.059	4.061
4.062	4.064	4.066	4.068	4.07
4.072	4.074	4.076	4.078	4.08
4.082	4.084	4.086	4.088	4.09
4.092	4.094	4.096	4.098	4.1
4.102	4.104	4.105	4.107	4.109
4.111	4.113	4.115	4.117	4.119
4.121	4.123	4.125	4.127	4.129
4.131	4.133	4.135	4.137	4.139
4.141	4.143	4.145	4.146	4.148
4.15	4.152	4.154	4.156	4.158
4.16	4.162	4.164	4.166	4.168
4.17	4.172	4.174	4.176	4.178
4.18	4.182	4.184	4.186	4.188
4.189	4.191	4.193	4.195	4.197
4.199	4.201	4.203	4.205	4.207
4.209	4.211	4.213	4.215	4.217
4.219	4.221	4.223	4.225	4.227
4.229	4.23	4.232	4.234	4.236
4.238	4.24	4.242	4.244	4.246
4.248	4.25	4.252	4.254	4.256
4.258	4.26	4.262	4.264	4.266
4.268	4.27	4.271	4.273	4.275
4.277	4.279	4.281	4.283	4.285
4.287	4.289	4.291	4.293	4.295
4.297	4.299	4.301	4.303	4.305
4.307	4.309	4.311	4.312	4.314

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
4.316	4.318	4.32	4.322	4.324
4.326	4.328	4.33	4.332	4.334
4.336	4.338	4.34	4.342	4.344
4.346	4.348	4.35	4.352	4.354
4.355	4.357	4.359	4.361	4.363
4.365	4.367	4.369	4.371	4.373
4.375	4.377	4.379	4.381	4.383
4.385	4.387	4.389	4.391	4.393
4.395	4.396	4.398	4.4	4.402
4.404	4.406	4.408	4.41	4.412
4.414	4.416	4.418	4.42	4.422
4.424	4.426	4.428	4.43	4.432
4.434	4.436	4.438	4.439	4.441
4.443	4.445	4.447	4.449	4.451
4.453	4.455	4.457	4.459	4.461
4.463	4.465	4.467	4.469	4.471
4.473	4.475	4.477	4.479	4.48
4.482	4.484	4.486	4.488	4.49
4.492	4.494	4.496	4.498	4.5
4.502	4.504	4.506	4.508	4.51
4.512	4.514	4.516	4.518	4.52
4.521	4.523	4.525	4.527	4.529
4.531	4.533	4.535	4.537	4.539
4.541	4.543	4.545	4.547	4.549
4.551	4.553	4.555	4.557	4.559
4.561	4.562	4.564	4.566	4.568
4.57	4.572	4.574	4.576	4.578
4.58	4.582	4.584	4.586	4.588
4.59	4.592	4.594	4.596	4.598
4.6	4.602	4.604	4.605	4.607
4.609	4.611	4.613	4.615	4.617
4.619	4.621	4.623	4.625	4.627
4.629	4.631	4.633	4.635	4.637
4.639	4.641	4.643	4.645	4.646
4.648	4.65	4.652	4.654	4.656
4.658	4.66	4.662	4.664	4.666
4.668	4.67	4.672	4.674	4.676
4.678	4.68	4.682	4.684	4.686
4.688	4.689	4.691	4.693	4.695
4.697	4.699	4.701	4.703	4.705
4.707	4.709	4.711	4.713	4.715
4.717	4.719	4.721	4.723	4.725
4.727	4.729	4.73	4.732	4.734
4.736	4.738	4.74	4.742	4.744
4.746	4.748	4.75	4.752	4.754
4.756	4.758	4.76	4.762	4.764
4.766	4.768	4.77	4.771	4.773
4.775	4.777	4.779	4.781	4.783
4.785	4.787	4.789	4.791	4.793
4.795	4.797	4.799	4.801	4.803

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
4.805	4.807	4.809	4.811	4.812
4.814	4.816	4.818	4.82	4.822
4.824	4.826	4.828	4.83	4.832
4.834	4.836	4.838	4.84	4.842
4.844	4.846	4.848	4.85	4.852
4.854	4.855	4.857	4.859	4.861
4.863	4.865	4.867	4.869	4.871
4.873	4.875	4.877	4.879	4.881
4.883	4.885	4.887	4.889	4.891
4.893	4.895	4.896	4.898	4.9
4.902	4.904	4.906	4.908	4.91
4.912	4.914	4.916	4.918	4.92
4.922	4.924	4.926	4.928	4.93
4.932	4.934	4.936	4.938	4.939
4.941	4.943	4.945	4.947	4.949
4.951	4.953	4.955	4.957	4.959
4.961	4.963	4.965	4.967	4.969
4.971	4.973	4.975	4.977	4.979
4.98	4.982	4.984	4.986	4.988
4.99	4.992	4.994	4.996	4.998
5	5.002	5.004	5.006	5.008
5.01	5.012	5.014	5.016	5.018
5.02	5.021	5.023	5.025	5.027
5.029	5.031	5.033	5.035	5.037
5.039	5.041	5.043	5.045	5.047
5.049	5.051	5.053	5.055	5.057
5.059	5.061	5.062	5.064	5.066
5.068	5.07	5.072	5.074	5.076
5.078	5.08	5.082	5.084	5.086
5.088	5.09	5.092	5.094	5.096
5.098	5.1	5.102	5.104	5.105
5.107	5.109	5.111	5.113	5.115
5.117	5.119	5.121	5.123	5.125
5.127	5.129	5.131	5.133	5.135
5.137	5.139	5.141	5.143	5.145
5.146	5.148	5.15	5.152	5.154
5.156	5.158	5.16	5.162	5.164
5.166	5.168	5.17	5.172	5.174
5.176	5.178	5.18	5.182	5.184
5.186	5.188	5.189	5.191	5.193
5.195	5.197	5.199	5.201	5.203
5.205	5.207	5.209	5.211	5.213
5.215	5.217	5.219	5.221	5.223
5.225	5.227	5.229	5.23	5.232
5.234	5.236	5.238	5.24	5.242
5.244	5.246	5.248	5.25	5.252
5.254	5.256	5.258	5.26	5.262
5.264	5.266	5.268	5.27	5.271
5.273	5.275	5.277	5.279	5.281
5.283	5.285	5.287	5.289	5.291

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
5.293	5.295	5.297	5.299	5.301
5.303	5.305	5.307	5.309	5.311
5.312	5.314	5.316	5.318	5.32
5.322	5.324	5.326	5.328	5.33
5.332	5.334	5.336	5.338	5.34
5.342	5.344	5.346	5.348	5.35
5.352	5.354	5.355	5.357	5.359
5.361	5.363	5.365	5.367	5.369
5.371	5.373	5.375	5.377	5.379
5.381	5.383	5.385	5.387	5.389
5.391	5.393	5.395	5.396	5.398
5.4	5.402	5.404	5.406	5.408
5.41	5.412	5.414	5.416	5.418
5.42	5.422	5.424	5.426	5.428
5.43	5.432	5.434	5.436	5.438
5.439	5.441	5.443	5.445	5.447
5.449	5.451	5.453	5.455	5.457
5.459	5.461	5.463	5.465	5.467
5.469	5.471	5.473	5.475	5.477
5.479	5.48	5.482	5.484	5.486
5.488	5.49	5.492	5.494	5.496
5.498	5.5	5.502	5.504	5.506
5.508	5.51	5.512	5.514	5.516
5.518	5.52	5.521	5.523	5.525
5.527	5.529	5.531	5.533	5.535
5.537	5.539	5.541	5.543	5.545
5.547	5.549	5.551	5.553	5.555
5.557	5.559	5.561	5.562	5.564
5.566	5.568	5.57	5.572	5.574
5.576	5.578	5.58	5.582	5.584
5.586	5.588	5.59	5.592	5.594
5.596	5.598	5.6	5.602	5.604
5.605	5.607	5.609	5.611	5.613
5.615	5.617	5.619	5.621	5.623
5.625	5.627	5.629	5.631	5.633
5.635	5.637	5.639	5.641	5.643
5.645	5.646	5.648	5.65	5.652
5.654	5.656	5.658	5.66	5.662
5.664	5.666	5.668	5.67	5.672
5.674	5.676	5.678	5.68	5.682
5.684	5.686	5.688	5.689	5.691
5.693	5.695	5.697	5.699	5.701
5.703	5.705	5.707	5.709	5.711
5.713	5.715	5.717	5.719	5.721
5.723	5.725	5.727	5.729	5.73
5.732	5.734	5.736	5.738	5.74
5.742	5.744	5.746	5.748	5.75
5.752	5.754	5.756	5.758	5.76
5.762	5.764	5.766	5.768	5.77
5.771	5.773	5.775	5.777	5.779

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
5.781	5.783	5.785	5.787	5.789
5.791	5.793	5.795	5.797	5.799
5.801	5.803	5.805	5.807	5.809
5.811	5.812	5.814	5.816	5.818
5.82	5.822	5.824	5.826	5.828
5.83	5.832	5.834	5.836	5.838
5.84	5.842	5.844	5.846	5.848
5.85	5.852	5.854	5.855	5.857
5.859	5.861	5.863	5.865	5.867
5.869	5.871	5.873	5.875	5.877
5.879	5.881	5.883	5.885	5.887
5.889	5.891	5.893	5.895	5.896
5.898	5.9	5.902	5.904	5.906
5.908	5.91	5.912	5.914	5.916
5.918	5.92	5.922	5.924	5.926
5.928	5.93	5.932	5.934	5.936
5.938	5.939	5.941	5.943	5.945
5.947	5.949	5.951	5.953	5.955
5.957	5.959	5.961	5.963	5.965
5.967	5.969	5.971	5.973	5.975
5.977	5.979	5.98	5.982	5.984
5.986	5.988	5.99	5.992	5.994
5.996	5.998	6	6.004	6.008
6.012	6.016	6.02	6.023	6.027
6.031	6.035	6.039	6.043	6.047
6.051	6.055	6.059	6.062	6.066
6.07	6.074	6.078	6.082	6.086
6.09	6.094	6.098	6.102	6.105
6.109	6.113	6.117	6.121	6.125
6.129	6.133	6.137	6.141	6.145
6.148	6.152	6.156	6.16	6.164
6.168	6.172	6.176	6.18	6.184
6.188	6.191	6.195	6.199	6.203
6.207	6.211	6.215	6.219	6.223
6.227	6.23	6.234	6.238	6.242
6.246	6.25	6.254	6.258	6.262
6.266	6.27	6.273	6.277	6.281
6.285	6.289	6.293	6.297	6.301
6.305	6.309	6.312	6.316	6.32
6.324	6.328	6.332	6.336	6.34
6.344	6.348	6.352	6.355	6.359
6.363	6.367	6.371	6.375	6.379
6.383	6.387	6.391	6.395	6.398
6.402	6.406	6.41	6.414	6.418
6.422	6.426	6.43	6.434	6.438
6.441	6.445	6.449	6.453	6.457
6.461	6.465	6.469	6.473	6.477
6.48	6.484	6.488	6.492	6.496
6.5	6.504	6.508	6.512	6.516
6.52	6.523	6.527	6.531	6.535

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
6.539	6.543	6.547	6.551	6.555
6.559	6.562	6.566	6.57	6.574
6.578	6.582	6.586	6.59	6.594
6.598	6.602	6.605	6.609	6.613
6.617	6.621	6.625	6.629	6.633
6.637	6.641	6.645	6.648	6.652
6.656	6.66	6.664	6.668	6.672
6.676	6.68	6.684	6.688	6.691
6.695	6.699	6.703	6.707	6.711
6.715	6.719	6.723	6.727	6.73
6.734	6.738	6.742	6.746	6.75
6.754	6.758	6.762	6.766	6.77
6.773	6.777	6.781	6.785	6.789
6.793	6.797	6.801	6.805	6.809
6.812	6.816	6.82	6.824	6.828
6.832	6.836	6.84	6.844	6.848
6.852	6.855	6.859	6.863	6.867
6.871	6.875	6.879	6.883	6.887
6.891	6.895	6.898	6.902	6.906
6.91	6.914	6.918	6.922	6.926
6.93	6.934	6.938	6.941	6.945
6.949	6.953	6.957	6.961	6.965
6.969	6.973	6.977	6.98	6.984
6.988	6.992	6.996	7	7.004
7.008	7.012	7.016	7.02	7.023
7.027	7.031	7.035	7.039	7.043
7.047	7.051	7.055	7.059	7.062
7.066	7.07	7.074	7.078	7.082
7.086	7.09	7.094	7.098	7.102
7.105	7.109	7.113	7.117	7.121
7.125	7.129	7.133	7.137	7.141
7.145	7.148	7.152	7.156	7.16
7.164	7.168	7.172	7.176	7.18
7.184	7.188	7.191	7.195	7.199
7.203	7.207	7.211	7.215	7.219
7.223	7.227	7.23	7.234	7.238
7.242	7.246	7.25	7.254	7.258
7.262	7.266	7.27	7.273	7.277
7.281	7.285	7.289	7.293	7.297
7.301	7.305	7.309	7.312	7.316
7.32	7.324	7.328	7.332	7.336
7.34	7.344	7.348	7.352	7.355
7.359	7.363	7.367	7.371	7.375
7.379	7.383	7.387	7.391	7.395
7.398	7.402	7.406	7.41	7.414
7.418	7.422	7.426	7.43	7.434
7.438	7.441	7.445	7.449	7.453
7.457	7.461	7.465	7.469	7.473
7.477	7.48	7.484	7.488	7.492
7.496	7.5	7.504	7.508	7.512

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
7.516	7.52	7.523	7.527	7.531
7.535	7.539	7.543	7.547	7.551
7.555	7.559	7.562	7.566	7.57
7.574	7.578	7.582	7.586	7.59
7.594	7.598	7.602	7.605	7.609
7.613	7.617	7.621	7.625	7.629
7.633	7.637	7.641	7.645	7.648
7.652	7.656	7.66	7.664	7.668
7.672	7.676	7.68	7.684	7.688
7.691	7.695	7.699	7.703	7.707
7.711	7.715	7.719	7.723	7.727
7.73	7.734	7.738	7.742	7.746
7.75	7.754	7.758	7.762	7.766
7.77	7.773	7.777	7.781	7.785
7.789	7.793	7.797	7.801	7.805
7.809	7.812	7.816	7.82	7.824
7.828	7.832	7.836	7.84	7.844
7.848	7.852	7.855	7.859	7.863
7.867	7.871	7.875	7.879	7.883
7.887	7.891	7.895	7.898	7.902
7.906	7.91	7.914	7.918	7.922
7.926	7.93	7.934	7.938	7.941
7.945	7.949	7.953	7.957	7.961
7.965	7.969	7.973	7.977	7.98
7.984	7.988	7.992	7.996	8
8.004	8.008	8.012	8.016	8.02
8.023	8.027	8.031	8.035	8.039
8.043	8.047	8.051	8.055	8.059
8.062	8.066	8.07	8.074	8.078
8.082	8.086	8.09	8.094	8.098
8.102	8.105	8.109	8.113	8.117
8.121	8.125	8.129	8.133	8.137
8.141	8.145	8.148	8.152	8.156
8.16	8.164	8.168	8.172	8.176
8.18	8.184	8.188	8.191	8.195
8.199	8.203	8.207	8.211	8.215
8.219	8.223	8.227	8.23	8.234
8.238	8.242	8.246	8.25	8.254
8.258	8.262	8.266	8.27	8.273
8.277	8.281	8.285	8.289	8.293
8.297	8.301	8.305	8.309	8.312
8.316	8.32	8.324	8.328	8.332
8.336	8.34	8.344	8.348	8.352
8.355	8.359	8.363	8.367	8.371
8.375	8.379	8.383	8.387	8.391
8.395	8.398	8.402	8.406	8.41
8.414	8.418	8.422	8.426	8.43
8.434	8.438	8.441	8.445	8.449
8.453	8.457	8.461	8.465	8.469
8.473	8.477	8.48	8.484	8.488

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
8.492	8.496	8.5	8.504	8.508
8.512	8.516	8.52	8.523	8.527
8.531	8.535	8.539	8.543	8.547
8.551	8.555	8.559	8.562	8.566
8.57	8.574	8.578	8.582	8.586
8.59	8.594	8.598	8.602	8.605
8.609	8.613	8.617	8.621	8.625
8.629	8.633	8.637	8.641	8.645
8.648	8.652	8.656	8.66	8.664
8.668	8.672	8.676	8.68	8.684
8.688	8.691	8.695	8.699	8.703
8.707	8.711	8.715	8.719	8.723
8.727	8.73	8.734	8.738	8.742
8.746	8.75	8.754	8.758	8.762
8.766	8.77	8.773	8.777	8.781
8.785	8.789	8.793	8.797	8.801
8.805	8.809	8.812	8.816	8.82
8.824	8.828	8.832	8.836	8.84
8.844	8.848	8.852	8.855	8.859
8.863	8.867	8.871	8.875	8.879
8.883	8.887	8.891	8.895	8.898
8.902	8.906	8.91	8.914	8.918
8.922	8.926	8.93	8.934	8.938
8.941	8.945	8.949	8.953	8.957
8.961	8.965	8.969	8.973	8.977
8.98	8.984	8.988	8.992	8.996
9	9.004	9.008	9.012	9.016
9.02	9.023	9.027	9.031	9.035
9.039	9.043	9.047	9.051	9.055
9.059	9.062	9.066	9.07	9.074
9.078	9.082	9.086	9.09	9.094
9.098	9.102	9.105	9.109	9.113
9.117	9.121	9.125	9.129	9.133
9.137	9.141	9.145	9.148	9.152
9.156	9.16	9.164	9.168	9.172
9.176	9.18	9.184	9.188	9.191
9.195	9.199	9.203	9.207	9.211
9.215	9.219	9.223	9.227	9.23
9.234	9.238	9.242	9.246	9.25
9.254	9.258	9.262	9.266	9.27
9.273	9.277	9.281	9.285	9.289
9.293	9.297	9.301	9.305	9.309
9.312	9.316	9.32	9.324	9.328
9.332	9.336	9.34	9.344	9.348
9.352	9.355	9.359	9.363	9.367
9.371	9.375	9.379	9.383	9.387
9.391	9.395	9.398	9.402	9.406
9.41	9.414	9.418	9.422	9.426
9.43	9.434	9.438	9.441	9.445
9.449	9.453	9.457	9.461	9.465

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
9.469	9.473	9.477	9.48	9.484
9.488	9.492	9.496	9.5	9.504
9.508	9.512	9.516	9.52	9.523
9.527	9.531	9.535	9.539	9.543
9.547	9.551	9.555	9.559	9.562
9.566	9.57	9.574	9.578	9.582
9.586	9.59	9.594	9.598	9.602
9.605	9.609	9.613	9.617	9.621
9.625	9.629	9.633	9.637	9.641
9.645	9.648	9.652	9.656	9.66
9.664	9.668	9.672	9.676	9.68
9.684	9.688	9.691	9.695	9.699
9.703	9.707	9.711	9.715	9.719
9.723	9.727	9.73	9.734	9.738
9.742	9.746	9.75	9.754	9.758
9.762	9.766	9.77	9.773	9.777
9.781	9.785	9.789	9.793	9.797
9.801	9.805	9.809	9.812	9.816
9.82	9.824	9.828	9.832	9.836
9.84	9.844	9.848	9.852	9.855
9.859	9.863	9.867	9.871	9.875
9.879	9.883	9.887	9.891	9.895
9.898	9.902	9.906	9.91	9.914
9.918	9.922	9.926	9.93	9.934
9.938	9.941	9.945	9.949	9.953
9.957	9.961	9.965	9.969	9.973
9.977	9.98	9.984	9.988	9.992
9.996	1×10^1	1.002×10^1	1.003×10^1	1.005×10^1
1.006×10^1	1.008×10^1	1.009×10^1	1.011×10^1	1.012×10^1
1.014×10^1	1.016×10^1	1.017×10^1	1.019×10^1	1.02×10^1
1.022×10^1	1.023×10^1	1.025×10^1	1.027×10^1	1.028×10^1
1.03×10^1	1.031×10^1	1.033×10^1	1.034×10^1	1.036×10^1
1.038×10^1	1.039×10^1	1.041×10^1	1.042×10^1	1.044×10^1
1.045×10^1	1.047×10^1	1.048×10^1	1.05×10^1	1.052×10^1
1.053×10^1	1.055×10^1	1.056×10^1	1.058×10^1	1.059×10^1
1.061×10^1	1.062×10^1	1.064×10^1	1.066×10^1	1.067×10^1
1.069×10^1	1.07×10^1	1.072×10^1	1.073×10^1	1.075×10^1
1.077×10^1	1.078×10^1	1.08×10^1	1.081×10^1	1.083×10^1
1.084×10^1	1.086×10^1	1.087×10^1	1.089×10^1	1.091×10^1
1.092×10^1	1.094×10^1	1.095×10^1	1.097×10^1	1.098×10^1
1.1×10^1	1.102×10^1	1.103×10^1	1.105×10^1	1.106×10^1
1.108×10^1	1.109×10^1	1.111×10^1	1.113×10^1	1.114×10^1
1.116×10^1	1.117×10^1	1.119×10^1	1.12×10^1	1.122×10^1
1.123×10^1	1.125×10^1	1.127×10^1	1.128×10^1	1.13×10^1
1.131×10^1	1.133×10^1	1.134×10^1	1.136×10^1	1.137×10^1
1.139×10^1	1.141×10^1	1.142×10^1	1.144×10^1	1.145×10^1
1.147×10^1	1.148×10^1	1.15×10^1	1.152×10^1	1.153×10^1
1.155×10^1	1.156×10^1	1.158×10^1	1.159×10^1	1.161×10^1
1.163×10^1	1.164×10^1	1.166×10^1	1.167×10^1	1.169×10^1
1.17×10^1	1.172×10^1	1.173×10^1	1.175×10^1	1.177×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
1.178×10^{1}	1.18×10^{1}	1.181×10^{1}	1.183×10^{1}	1.184×10^{1}
1.186×10^{1}	1.188×10^{1}	1.189×10^{1}	1.191×10^{1}	1.192×10^{1}
1.194×10^{1}	1.195×10^{1}	1.197×10^{1}	1.198×10^{1}	1.2×10^{1}
1.202×10^{1}	1.203×10^{1}	1.205×10^{1}	1.206×10^{1}	1.208×10^{1}
1.209×10^{1}	1.211×10^{1}	1.212×10^{1}	1.214×10^{1}	1.216×10^{1}
1.217×10^{1}	1.219×10^{1}	1.22×10^{1}	1.222×10^{1}	1.223×10^{1}
1.225×10^{1}	1.227×10^{1}	1.228×10^{1}	1.23×10^{1}	1.231×10^{1}
1.233×10^{1}	1.234×10^{1}	1.236×10^{1}	1.238×10^{1}	1.239×10^{1}
1.241×10^{1}	1.242×10^{1}	1.244×10^{1}	1.245×10^{1}	1.247×10^{1}
1.248×10^{1}	1.25×10^{1}	1.252×10^{1}	1.253×10^{1}	1.255×10^{1}
1.256×10^{1}	1.258×10^{1}	1.259×10^{1}	1.261×10^{1}	1.262×10^{1}
1.264×10^{1}	1.266×10^{1}	1.267×10^{1}	1.269×10^{1}	1.27×10^{1}
1.272×10^{1}	1.273×10^{1}	1.275×10^{1}	1.277×10^{1}	1.278×10^{1}
1.28×10^{1}	1.281×10^{1}	1.283×10^{1}	1.284×10^{1}	1.286×10^{1}
1.288×10^{1}	1.289×10^{1}	1.291×10^{1}	1.292×10^{1}	1.294×10^{1}
1.295×10^{1}	1.297×10^{1}	1.298×10^{1}	1.3×10^{1}	1.302×10^{1}
1.303×10^{1}	1.305×10^{1}	1.306×10^{1}	1.308×10^{1}	1.309×10^{1}
1.311×10^{1}	1.312×10^{1}	1.314×10^{1}	1.316×10^{1}	1.317×10^{1}
1.319×10^{1}	1.32×10^{1}	1.322×10^{1}	1.323×10^{1}	1.325×10^{1}
1.327×10^{1}	1.328×10^{1}	1.33×10^{1}	1.331×10^{1}	1.333×10^{1}
1.334×10^{1}	1.336×10^{1}	1.337×10^{1}	1.339×10^{1}	1.341×10^{1}
1.342×10^{1}	1.344×10^{1}	1.345×10^{1}	1.347×10^{1}	1.348×10^{1}
1.35×10^{1}	1.352×10^{1}	1.353×10^{1}	1.355×10^{1}	1.356×10^{1}
1.358×10^{1}	1.359×10^{1}	1.361×10^{1}	1.363×10^{1}	1.364×10^{1}
1.366×10^{1}	1.367×10^{1}	1.369×10^{1}	1.37×10^{1}	1.372×10^{1}
1.373×10^{1}	1.375×10^{1}	1.377×10^{1}	1.378×10^{1}	1.38×10^{1}
1.381×10^{1}	1.383×10^{1}	1.384×10^{1}	1.386×10^{1}	1.387×10^{1}
1.389×10^{1}	1.391×10^{1}	1.392×10^{1}	1.394×10^{1}	1.395×10^{1}
1.397×10^{1}	1.398×10^{1}	1.4×10^{1}	1.402×10^{1}	1.403×10^{1}
1.405×10^{1}	1.406×10^{1}	1.408×10^{1}	1.409×10^{1}	1.411×10^{1}
1.413×10^{1}	1.414×10^{1}	1.416×10^{1}	1.417×10^{1}	1.419×10^{1}
1.42×10^{1}	1.422×10^{1}	1.423×10^{1}	1.425×10^{1}	1.427×10^{1}
1.428×10^{1}	1.43×10^{1}	1.431×10^{1}	1.433×10^{1}	1.434×10^{1}
1.436×10^{1}	1.438×10^{1}	1.439×10^{1}	1.441×10^{1}	1.442×10^{1}
1.444×10^{1}	1.445×10^{1}	1.447×10^{1}	1.448×10^{1}	1.45×10^{1}
1.452×10^{1}	1.453×10^{1}	1.455×10^{1}	1.456×10^{1}	1.458×10^{1}
1.459×10^{1}	1.461×10^{1}	1.462×10^{1}	1.464×10^{1}	1.466×10^{1}
1.467×10^{1}	1.469×10^{1}	1.47×10^{1}	1.472×10^{1}	1.473×10^{1}
1.475×10^{1}	1.477×10^{1}	1.478×10^{1}	1.48×10^{1}	1.481×10^{1}
1.483×10^{1}	1.484×10^{1}	1.486×10^{1}	1.488×10^{1}	1.489×10^{1}
1.491×10^{1}	1.492×10^{1}	1.494×10^{1}	1.495×10^{1}	1.497×10^{1}
1.498×10^{1}	1.5×10^{1}	1.502×10^{1}	1.503×10^{1}	1.505×10^{1}
1.506×10^{1}	1.508×10^{1}	1.509×10^{1}	1.511×10^{1}	1.512×10^{1}
1.514×10^{1}	1.516×10^{1}	1.517×10^{1}	1.519×10^{1}	1.52×10^{1}
1.522×10^{1}	1.523×10^{1}	1.525×10^{1}	1.527×10^{1}	1.528×10^{1}
1.53×10^{1}	1.531×10^{1}	1.533×10^{1}	1.534×10^{1}	1.536×10^{1}
1.538×10^{1}	1.539×10^{1}	1.541×10^{1}	1.542×10^{1}	1.544×10^{1}
1.545×10^{1}	1.547×10^{1}	1.548×10^{1}	1.55×10^{1}	1.552×10^{1}
1.553×10^{1}	1.555×10^{1}	1.556×10^{1}	1.558×10^{1}	1.559×10^{1}
1.561×10^{1}	1.562×10^{1}	1.564×10^{1}	1.566×10^{1}	1.567×10^{1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
1.569×10^{1}	1.57×10^{1}	1.572×10^{1}	1.573×10^{1}	1.575×10^{1}
1.577×10^{1}	1.578×10^{1}	1.58×10^{1}	1.581×10^{1}	1.583×10^{1}
1.584×10^{1}	1.586×10^{1}	1.587×10^{1}	1.589×10^{1}	1.591×10^{1}
1.592×10^{1}	1.594×10^{1}	1.595×10^{1}	1.597×10^{1}	1.598×10^{1}
1.6×10^{1}	1.602×10^{1}	1.603×10^{1}	1.605×10^{1}	1.606×10^{1}
1.608×10^{1}	1.609×10^{1}	1.611×10^{1}	1.613×10^{1}	1.614×10^{1}
1.616×10^{1}	1.617×10^{1}	1.619×10^{1}	1.62×10^{1}	1.622×10^{1}
1.623×10^{1}	1.625×10^{1}	1.627×10^{1}	1.628×10^{1}	1.63×10^{1}
1.631×10^{1}	1.633×10^{1}	1.634×10^{1}	1.636×10^{1}	1.637×10^{1}
1.639×10^{1}	1.641×10^{1}	1.642×10^{1}	1.644×10^{1}	1.645×10^{1}
1.647×10^{1}	1.648×10^{1}	1.65×10^{1}	1.652×10^{1}	1.653×10^{1}
1.655×10^{1}	1.656×10^{1}	1.658×10^{1}	1.659×10^{1}	1.661×10^{1}
1.663×10^{1}	1.664×10^{1}	1.666×10^{1}	1.667×10^{1}	1.669×10^{1}
1.67×10^{1}	1.672×10^{1}	1.673×10^{1}	1.675×10^{1}	1.677×10^{1}
1.678×10^{1}	1.68×10^{1}	1.681×10^{1}	1.683×10^{1}	1.684×10^{1}
1.686×10^{1}	1.688×10^{1}	1.689×10^{1}	1.691×10^{1}	1.692×10^{1}
1.694×10^{1}	1.695×10^{1}	1.697×10^{1}	1.698×10^{1}	1.7×10^{1}
1.702×10^{1}	1.703×10^{1}	1.705×10^{1}	1.706×10^{1}	1.708×10^{1}
1.709×10^{1}	1.711×10^{1}	1.712×10^{1}	1.714×10^{1}	1.716×10^{1}
1.717×10^{1}	1.719×10^{1}	1.72×10^{1}	1.722×10^{1}	1.723×10^{1}
1.725×10^{1}	1.727×10^{1}	1.728×10^{1}	1.73×10^{1}	1.731×10^{1}
1.733×10^{1}	1.734×10^{1}	1.736×10^{1}	1.738×10^{1}	1.739×10^{1}
1.741×10^{1}	1.742×10^{1}	1.744×10^{1}	1.745×10^{1}	1.747×10^{1}
1.748×10^{1}	1.75×10^{1}	1.752×10^{1}	1.753×10^{1}	1.755×10^{1}
1.756×10^{1}	1.758×10^{1}	1.759×10^{1}	1.761×10^{1}	1.762×10^{1}
1.764×10^{1}	1.766×10^{1}	1.767×10^{1}	1.769×10^{1}	1.77×10^{1}
1.772×10^{1}	1.773×10^{1}	1.775×10^{1}	1.777×10^{1}	1.778×10^{1}
1.78×10^{1}	1.781×10^{1}	1.783×10^{1}	1.784×10^{1}	1.786×10^{1}
1.788×10^{1}	1.789×10^{1}	1.791×10^{1}	1.792×10^{1}	1.794×10^{1}
1.795×10^{1}	1.797×10^{1}	1.798×10^{1}	1.8×10^{1}	1.802×10^{1}
1.803×10^{1}	1.805×10^{1}	1.806×10^{1}	1.808×10^{1}	1.809×10^{1}
1.811×10^{1}	1.812×10^{1}	1.814×10^{1}	1.816×10^{1}	1.817×10^{1}
1.819×10^{1}	1.82×10^{1}	1.822×10^{1}	1.823×10^{1}	1.825×10^{1}
1.827×10^{1}	1.828×10^{1}	1.83×10^{1}	1.831×10^{1}	1.833×10^{1}
1.834×10^{1}	1.836×10^{1}	1.837×10^{1}	1.839×10^{1}	1.841×10^{1}
1.842×10^{1}	1.844×10^{1}	1.845×10^{1}	1.847×10^{1}	1.848×10^{1}
1.85×10^{1}	1.852×10^{1}	1.853×10^{1}	1.855×10^{1}	1.856×10^{1}
1.858×10^{1}	1.859×10^{1}	1.861×10^{1}	1.863×10^{1}	1.864×10^{1}
1.866×10^{1}	1.867×10^{1}	1.869×10^{1}	1.87×10^{1}	1.872×10^{1}
1.873×10^{1}	1.875×10^{1}	1.877×10^{1}	1.878×10^{1}	1.88×10^{1}
1.881×10^{1}	1.883×10^{1}	1.884×10^{1}	1.886×10^{1}	1.887×10^{1}
1.889×10^{1}	1.891×10^{1}	1.892×10^{1}	1.894×10^{1}	1.895×10^{1}
1.897×10^{1}	1.898×10^{1}	1.9×10^{1}	1.902×10^{1}	1.903×10^{1}
1.905×10^{1}	1.906×10^{1}	1.908×10^{1}	1.909×10^{1}	1.911×10^{1}
1.913×10^{1}	1.914×10^{1}	1.916×10^{1}	1.917×10^{1}	1.919×10^{1}
1.92×10^{1}	1.922×10^{1}	1.923×10^{1}	1.925×10^{1}	1.927×10^{1}
1.928×10^{1}	1.93×10^{1}	1.931×10^{1}	1.933×10^{1}	1.934×10^{1}
1.936×10^{1}	1.938×10^{1}	1.939×10^{1}	1.941×10^{1}	1.942×10^{1}
1.944×10^{1}	1.945×10^{1}	1.947×10^{1}	1.948×10^{1}	1.95×10^{1}
1.952×10^{1}	1.953×10^{1}	1.955×10^{1}	1.956×10^{1}	1.958×10^{1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
1.959×10^1	1.961×10^1	1.962×10^1	1.964×10^1	1.966×10^1
1.967×10^1	1.969×10^1	1.97×10^1	1.972×10^1	1.973×10^1
1.975×10^1	1.977×10^1	1.978×10^1	1.98×10^1	1.981×10^1
1.983×10^1	1.984×10^1	1.986×10^1	1.988×10^1	1.989×10^1
1.991×10^1	1.992×10^1	1.994×10^1	1.995×10^1	1.997×10^1
1.998×10^1	2×10^1	2.003×10^1	2.006×10^1	2.009×10^1
2.013×10^1	2.016×10^1	2.019×10^1	2.022×10^1	2.025×10^1
2.028×10^1	2.031×10^1	2.034×10^1	2.038×10^1	2.041×10^1
2.044×10^1	2.047×10^1	2.05×10^1	2.053×10^1	2.056×10^1
2.059×10^1	2.062×10^1	2.066×10^1	2.069×10^1	2.072×10^1
2.075×10^1	2.078×10^1	2.081×10^1	2.084×10^1	2.087×10^1
2.091×10^1	2.094×10^1	2.097×10^1	2.1×10^1	2.103×10^1
2.106×10^1	2.109×10^1	2.112×10^1	2.116×10^1	2.119×10^1
2.122×10^1	2.125×10^1	2.128×10^1	2.131×10^1	2.134×10^1
2.138×10^1	2.141×10^1	2.144×10^1	2.147×10^1	2.15×10^1
2.153×10^1	2.156×10^1	2.159×10^1	2.163×10^1	2.166×10^1
2.169×10^1	2.172×10^1	2.175×10^1	2.178×10^1	2.181×10^1
2.184×10^1	2.188×10^1	2.191×10^1	2.194×10^1	2.197×10^1
2.2×10^1	2.203×10^1	2.206×10^1	2.209×10^1	2.212×10^1
2.216×10^1	2.219×10^1	2.222×10^1	2.225×10^1	2.228×10^1
2.231×10^1	2.234×10^1	2.237×10^1	2.241×10^1	2.244×10^1
2.247×10^1	2.25×10^1	2.253×10^1	2.256×10^1	2.259×10^1
2.263×10^1	2.266×10^1	2.269×10^1	2.272×10^1	2.275×10^1
2.278×10^1	2.281×10^1	2.284×10^1	2.288×10^1	2.291×10^1
2.294×10^1	2.297×10^1	2.3×10^1	2.303×10^1	2.306×10^1
2.309×10^1	2.312×10^1	2.316×10^1	2.319×10^1	2.322×10^1
2.325×10^1	2.328×10^1	2.331×10^1	2.334×10^1	2.337×10^1
2.341×10^1	2.344×10^1	2.347×10^1	2.35×10^1	2.353×10^1
2.356×10^1	2.359×10^1	2.362×10^1	2.366×10^1	2.369×10^1
2.372×10^1	2.375×10^1	2.378×10^1	2.381×10^1	2.384×10^1
2.388×10^1	2.391×10^1	2.394×10^1	2.397×10^1	2.4×10^1
2.403×10^1	2.406×10^1	2.409×10^1	2.413×10^1	2.416×10^1
2.419×10^1	2.422×10^1	2.425×10^1	2.428×10^1	2.431×10^1
2.434×10^1	2.438×10^1	2.441×10^1	2.444×10^1	2.447×10^1
2.45×10^1	2.453×10^1	2.456×10^1	2.459×10^1	2.462×10^1
2.466×10^1	2.469×10^1	2.472×10^1	2.475×10^1	2.478×10^1
2.481×10^1	2.484×10^1	2.487×10^1	2.491×10^1	2.494×10^1
2.497×10^1	2.5×10^1	2.503×10^1	2.506×10^1	2.509×10^1
2.513×10^1	2.516×10^1	2.519×10^1	2.522×10^1	2.525×10^1
2.528×10^1	2.531×10^1	2.534×10^1	2.538×10^1	2.541×10^1
2.544×10^1	2.547×10^1	2.55×10^1	2.553×10^1	2.556×10^1
2.559×10^1	2.562×10^1	2.566×10^1	2.569×10^1	2.572×10^1
2.575×10^1	2.578×10^1	2.581×10^1	2.584×10^1	2.587×10^1
2.591×10^1	2.594×10^1	2.597×10^1	2.6×10^1	2.603×10^1
2.606×10^1	2.609×10^1	2.612×10^1	2.616×10^1	2.619×10^1
2.622×10^1	2.625×10^1	2.628×10^1	2.631×10^1	2.634×10^1
2.638×10^1	2.641×10^1	2.644×10^1	2.647×10^1	2.65×10^1
2.653×10^1	2.656×10^1	2.659×10^1	2.663×10^1	2.666×10^1
2.669×10^1	2.672×10^1	2.675×10^1	2.678×10^1	2.681×10^1
2.684×10^1	2.688×10^1	2.691×10^1	2.694×10^1	2.697×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [**MeV**]	**Energy** [**MeV**]	**Energy** [**MeV**]	**Energy** [**MeV**]	**Energy** [**MeV**]
2.7×10^{1}	2.703×10^{1}	2.706×10^{1}	2.709×10^{1}	2.712×10^{1}
2.716×10^{1}	2.719×10^{1}	2.722×10^{1}	2.725×10^{1}	2.728×10^{1}
2.731×10^{1}	2.734×10^{1}	2.737×10^{1}	2.741×10^{1}	2.744×10^{1}
2.747×10^{1}	2.75×10^{1}	2.753×10^{1}	2.756×10^{1}	2.759×10^{1}
2.763×10^{1}	2.766×10^{1}	2.769×10^{1}	2.772×10^{1}	2.775×10^{1}
2.778×10^{1}	2.781×10^{1}	2.784×10^{1}	2.788×10^{1}	2.791×10^{1}
2.794×10^{1}	2.797×10^{1}	2.8×10^{1}	2.803×10^{1}	2.806×10^{1}
2.809×10^{1}	2.812×10^{1}	2.816×10^{1}	2.819×10^{1}	2.822×10^{1}
2.825×10^{1}	2.828×10^{1}	2.831×10^{1}	2.834×10^{1}	2.837×10^{1}
2.841×10^{1}	2.844×10^{1}	2.847×10^{1}	2.85×10^{1}	2.853×10^{1}
2.856×10^{1}	2.859×10^{1}	2.862×10^{1}	2.866×10^{1}	2.869×10^{1}
2.872×10^{1}	2.875×10^{1}	2.878×10^{1}	2.881×10^{1}	2.884×10^{1}
2.888×10^{1}	2.891×10^{1}	2.894×10^{1}	2.897×10^{1}	2.9×10^{1}
2.903×10^{1}	2.906×10^{1}	2.909×10^{1}	2.913×10^{1}	2.916×10^{1}
2.919×10^{1}	2.922×10^{1}	2.925×10^{1}	2.928×10^{1}	2.931×10^{1}
2.934×10^{1}	2.938×10^{1}	2.941×10^{1}	2.944×10^{1}	2.947×10^{1}
2.95×10^{1}	2.953×10^{1}	2.956×10^{1}	2.959×10^{1}	2.962×10^{1}
2.966×10^{1}	2.969×10^{1}	2.972×10^{1}	2.975×10^{1}	2.978×10^{1}
2.981×10^{1}	2.984×10^{1}	2.987×10^{1}	2.991×10^{1}	2.994×10^{1}
2.997×10^{1}	3×10^{1}	3.003×10^{1}	3.006×10^{1}	3.009×10^{1}
3.013×10^{1}	3.016×10^{1}	3.019×10^{1}	3.022×10^{1}	3.025×10^{1}
3.028×10^{1}	3.031×10^{1}	3.034×10^{1}	3.038×10^{1}	3.041×10^{1}
3.044×10^{1}	3.047×10^{1}	3.05×10^{1}	3.053×10^{1}	3.056×10^{1}
3.059×10^{1}	3.062×10^{1}	3.066×10^{1}	3.069×10^{1}	3.072×10^{1}
3.075×10^{1}	3.078×10^{1}	3.081×10^{1}	3.084×10^{1}	3.087×10^{1}
3.091×10^{1}	3.094×10^{1}	3.097×10^{1}	3.1×10^{1}	3.103×10^{1}
3.106×10^{1}	3.109×10^{1}	3.112×10^{1}	3.116×10^{1}	3.119×10^{1}
3.122×10^{1}	3.125×10^{1}	3.128×10^{1}	3.131×10^{1}	3.134×10^{1}
3.138×10^{1}	3.141×10^{1}	3.144×10^{1}	3.147×10^{1}	3.15×10^{1}
3.153×10^{1}	3.156×10^{1}	3.159×10^{1}	3.163×10^{1}	3.166×10^{1}
3.169×10^{1}	3.172×10^{1}	3.175×10^{1}	3.178×10^{1}	3.181×10^{1}
3.184×10^{1}	3.188×10^{1}	3.191×10^{1}	3.194×10^{1}	3.197×10^{1}
3.2×10^{1}	3.203×10^{1}	3.206×10^{1}	3.209×10^{1}	3.212×10^{1}
3.216×10^{1}	3.219×10^{1}	3.222×10^{1}	3.225×10^{1}	3.228×10^{1}
3.231×10^{1}	3.234×10^{1}	3.237×10^{1}	3.241×10^{1}	3.244×10^{1}
3.247×10^{1}	3.25×10^{1}	3.253×10^{1}	3.256×10^{1}	3.259×10^{1}
3.263×10^{1}	3.266×10^{1}	3.269×10^{1}	3.272×10^{1}	3.275×10^{1}
3.278×10^{1}	3.281×10^{1}	3.284×10^{1}	3.288×10^{1}	3.291×10^{1}
3.294×10^{1}	3.297×10^{1}	3.3×10^{1}	3.303×10^{1}	3.306×10^{1}
3.309×10^{1}	3.312×10^{1}	3.316×10^{1}	3.319×10^{1}	3.322×10^{1}
3.325×10^{1}	3.328×10^{1}	3.331×10^{1}	3.334×10^{1}	3.337×10^{1}
3.341×10^{1}	3.344×10^{1}	3.347×10^{1}	3.35×10^{1}	3.353×10^{1}
3.356×10^{1}	3.359×10^{1}	3.362×10^{1}	3.366×10^{1}	3.369×10^{1}
3.372×10^{1}	3.375×10^{1}	3.378×10^{1}	3.381×10^{1}	3.384×10^{1}
3.388×10^{1}	3.391×10^{1}	3.394×10^{1}	3.397×10^{1}	3.4×10^{1}
3.403×10^{1}	3.406×10^{1}	3.409×10^{1}	3.413×10^{1}	3.416×10^{1}
3.419×10^{1}	3.422×10^{1}	3.425×10^{1}	3.428×10^{1}	3.431×10^{1}
3.434×10^{1}	3.438×10^{1}	3.441×10^{1}	3.444×10^{1}	3.447×10^{1}
3.45×10^{1}	3.453×10^{1}	3.456×10^{1}	3.459×10^{1}	3.462×10^{1}
3.466×10^{1}	3.469×10^{1}	3.472×10^{1}	3.475×10^{1}	3.478×10^{1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
$3.481{\times}10^{1}$	$3.484{\times}10^{1}$	$3.487{\times}10^{1}$	$3.491{\times}10^{1}$	$3.494{\times}10^{1}$
$3.497{\times}10^{1}$	$3.5{\times}10^{1}$	$3.503{\times}10^{1}$	$3.506{\times}10^{1}$	$3.509{\times}10^{1}$
$3.513{\times}10^{1}$	$3.516{\times}10^{1}$	$3.519{\times}10^{1}$	$3.522{\times}10^{1}$	$3.525{\times}10^{1}$
$3.528{\times}10^{1}$	$3.531{\times}10^{1}$	$3.534{\times}10^{1}$	$3.538{\times}10^{1}$	$3.541{\times}10^{1}$
$3.544{\times}10^{1}$	$3.547{\times}10^{1}$	$3.55{\times}10^{1}$	$3.553{\times}10^{1}$	$3.556{\times}10^{1}$
$3.559{\times}10^{1}$	$3.562{\times}10^{1}$	$3.566{\times}10^{1}$	$3.569{\times}10^{1}$	$3.572{\times}10^{1}$
$3.575{\times}10^{1}$	$3.578{\times}10^{1}$	$3.581{\times}10^{1}$	$3.584{\times}10^{1}$	$3.587{\times}10^{1}$
$3.591{\times}10^{1}$	$3.594{\times}10^{1}$	$3.597{\times}10^{1}$	$3.6{\times}10^{1}$	$3.603{\times}10^{1}$
$3.606{\times}10^{1}$	$3.609{\times}10^{1}$	$3.612{\times}10^{1}$	$3.616{\times}10^{1}$	$3.619{\times}10^{1}$
$3.622{\times}10^{1}$	$3.625{\times}10^{1}$	$3.628{\times}10^{1}$	$3.631{\times}10^{1}$	$3.634{\times}10^{1}$
$3.638{\times}10^{1}$	$3.641{\times}10^{1}$	$3.644{\times}10^{1}$	$3.647{\times}10^{1}$	$3.65{\times}10^{1}$
$3.653{\times}10^{1}$	$3.656{\times}10^{1}$	$3.659{\times}10^{1}$	$3.663{\times}10^{1}$	$3.666{\times}10^{1}$
$3.669{\times}10^{1}$	$3.672{\times}10^{1}$	$3.675{\times}10^{1}$	$3.678{\times}10^{1}$	$3.681{\times}10^{1}$
$3.684{\times}10^{1}$	$3.688{\times}10^{1}$	$3.691{\times}10^{1}$	$3.694{\times}10^{1}$	$3.697{\times}10^{1}$
$3.7{\times}10^{1}$	$3.703{\times}10^{1}$	$3.706{\times}10^{1}$	$3.709{\times}10^{1}$	$3.712{\times}10^{1}$
$3.716{\times}10^{1}$	$3.719{\times}10^{1}$	$3.722{\times}10^{1}$	$3.725{\times}10^{1}$	$3.728{\times}10^{1}$
$3.731{\times}10^{1}$	$3.734{\times}10^{1}$	$3.737{\times}10^{1}$	$3.741{\times}10^{1}$	$3.744{\times}10^{1}$
$3.747{\times}10^{1}$	$3.75{\times}10^{1}$	$3.753{\times}10^{1}$	$3.756{\times}10^{1}$	$3.759{\times}10^{1}$
$3.763{\times}10^{1}$	$3.766{\times}10^{1}$	$3.769{\times}10^{1}$	$3.772{\times}10^{1}$	$3.775{\times}10^{1}$
$3.778{\times}10^{1}$	$3.781{\times}10^{1}$	$3.784{\times}10^{1}$	$3.788{\times}10^{1}$	$3.791{\times}10^{1}$
$3.794{\times}10^{1}$	$3.797{\times}10^{1}$	$3.8{\times}10^{1}$	$3.803{\times}10^{1}$	$3.806{\times}10^{1}$
$3.809{\times}10^{1}$	$3.812{\times}10^{1}$	$3.816{\times}10^{1}$	$3.819{\times}10^{1}$	$3.822{\times}10^{1}$
$3.825{\times}10^{1}$	$3.828{\times}10^{1}$	$3.831{\times}10^{1}$	$3.834{\times}10^{1}$	$3.837{\times}10^{1}$
$3.841{\times}10^{1}$	$3.844{\times}10^{1}$	$3.847{\times}10^{1}$	$3.85{\times}10^{1}$	$3.853{\times}10^{1}$
$3.856{\times}10^{1}$	$3.859{\times}10^{1}$	$3.862{\times}10^{1}$	$3.866{\times}10^{1}$	$3.869{\times}10^{1}$
$3.872{\times}10^{1}$	$3.875{\times}10^{1}$	$3.878{\times}10^{1}$	$3.881{\times}10^{1}$	$3.884{\times}10^{1}$
$3.888{\times}10^{1}$	$3.891{\times}10^{1}$	$3.894{\times}10^{1}$	$3.897{\times}10^{1}$	$3.9{\times}10^{1}$
$3.903{\times}10^{1}$	$3.906{\times}10^{1}$	$3.909{\times}10^{1}$	$3.913{\times}10^{1}$	$3.916{\times}10^{1}$
$3.919{\times}10^{1}$	$3.922{\times}10^{1}$	$3.925{\times}10^{1}$	$3.928{\times}10^{1}$	$3.931{\times}10^{1}$
$3.934{\times}10^{1}$	$3.938{\times}10^{1}$	$3.941{\times}10^{1}$	$3.944{\times}10^{1}$	$3.947{\times}10^{1}$
$3.95{\times}10^{1}$	$3.953{\times}10^{1}$	$3.956{\times}10^{1}$	$3.959{\times}10^{1}$	$3.962{\times}10^{1}$
$3.966{\times}10^{1}$	$3.969{\times}10^{1}$	$3.972{\times}10^{1}$	$3.975{\times}10^{1}$	$3.978{\times}10^{1}$
$3.981{\times}10^{1}$	$3.984{\times}10^{1}$	$3.987{\times}10^{1}$	$3.991{\times}10^{1}$	$3.994{\times}10^{1}$
$3.997{\times}10^{1}$	$4{\times}10^{1}$	$4.003{\times}10^{1}$	$4.006{\times}10^{1}$	$4.009{\times}10^{1}$
$4.013{\times}10^{1}$	$4.016{\times}10^{1}$	$4.019{\times}10^{1}$	$4.022{\times}10^{1}$	$4.025{\times}10^{1}$
$4.028{\times}10^{1}$	$4.031{\times}10^{1}$	$4.034{\times}10^{1}$	$4.037{\times}10^{1}$	$4.041{\times}10^{1}$
$4.044{\times}10^{1}$	$4.047{\times}10^{1}$	$4.05{\times}10^{1}$	$4.053{\times}10^{1}$	$4.056{\times}10^{1}$
$4.059{\times}10^{1}$	$4.062{\times}10^{1}$	$4.066{\times}10^{1}$	$4.069{\times}10^{1}$	$4.072{\times}10^{1}$
$4.075{\times}10^{1}$	$4.078{\times}10^{1}$	$4.081{\times}10^{1}$	$4.084{\times}10^{1}$	$4.088{\times}10^{1}$
$4.091{\times}10^{1}$	$4.094{\times}10^{1}$	$4.097{\times}10^{1}$	$4.1{\times}10^{1}$	$4.103{\times}10^{1}$
$4.106{\times}10^{1}$	$4.109{\times}10^{1}$	$4.112{\times}10^{1}$	$4.116{\times}10^{1}$	$4.119{\times}10^{1}$
$4.122{\times}10^{1}$	$4.125{\times}10^{1}$	$4.128{\times}10^{1}$	$4.131{\times}10^{1}$	$4.134{\times}10^{1}$
$4.138{\times}10^{1}$	$4.141{\times}10^{1}$	$4.144{\times}10^{1}$	$4.147{\times}10^{1}$	$4.15{\times}10^{1}$
$4.153{\times}10^{1}$	$4.156{\times}10^{1}$	$4.159{\times}10^{1}$	$4.162{\times}10^{1}$	$4.166{\times}10^{1}$
$4.169{\times}10^{1}$	$4.172{\times}10^{1}$	$4.175{\times}10^{1}$	$4.178{\times}10^{1}$	$4.181{\times}10^{1}$
$4.184{\times}10^{1}$	$4.188{\times}10^{1}$	$4.191{\times}10^{1}$	$4.194{\times}10^{1}$	$4.197{\times}10^{1}$
$4.2{\times}10^{1}$	$4.203{\times}10^{1}$	$4.206{\times}10^{1}$	$4.209{\times}10^{1}$	$4.213{\times}10^{1}$
$4.216{\times}10^{1}$	$4.219{\times}10^{1}$	$4.222{\times}10^{1}$	$4.225{\times}10^{1}$	$4.228{\times}10^{1}$
$4.231{\times}10^{1}$	$4.234{\times}10^{1}$	$4.237{\times}10^{1}$	$4.241{\times}10^{1}$	$4.244{\times}10^{1}$
$4.247{\times}10^{1}$	$4.25{\times}10^{1}$	$4.253{\times}10^{1}$	$4.256{\times}10^{1}$	$4.259{\times}10^{1}$

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
4.263×10^1	4.266×10^1	4.269×10^1	4.272×10^1	4.275×10^1
4.278×10^1	4.281×10^1	4.284×10^1	4.287×10^1	4.291×10^1
4.294×10^1	4.297×10^1	4.3×10^1	4.303×10^1	4.306×10^1
4.309×10^1	4.312×10^1	4.316×10^1	4.319×10^1	4.322×10^1
4.325×10^1	4.328×10^1	4.331×10^1	4.334×10^1	4.338×10^1
4.341×10^1	4.344×10^1	4.347×10^1	4.35×10^1	4.353×10^1
4.356×10^1	4.359×10^1	4.362×10^1	4.366×10^1	4.369×10^1
4.372×10^1	4.375×10^1	4.378×10^1	4.381×10^1	4.384×10^1
4.388×10^1	4.391×10^1	4.394×10^1	4.397×10^1	4.4×10^1
4.403×10^1	4.406×10^1	4.409×10^1	4.412×10^1	4.416×10^1
4.419×10^1	4.422×10^1	4.425×10^1	4.428×10^1	4.431×10^1
4.434×10^1	4.438×10^1	4.441×10^1	4.444×10^1	4.447×10^1
4.45×10^1	4.453×10^1	4.456×10^1	4.459×10^1	4.463×10^1
4.466×10^1	4.469×10^1	4.472×10^1	4.475×10^1	4.478×10^1
4.481×10^1	4.484×10^1	4.487×10^1	4.491×10^1	4.494×10^1
4.497×10^1	4.5×10^1	4.503×10^1	4.506×10^1	4.509×10^1
4.513×10^1	4.516×10^1	4.519×10^1	4.522×10^1	4.525×10^1
4.528×10^1	4.531×10^1	4.534×10^1	4.537×10^1	4.541×10^1
4.544×10^1	4.547×10^1	4.55×10^1	4.553×10^1	4.556×10^1
4.559×10^1	4.562×10^1	4.566×10^1	4.569×10^1	4.572×10^1
4.575×10^1	4.578×10^1	4.581×10^1	4.584×10^1	4.588×10^1
4.591×10^1	4.594×10^1	4.597×10^1	4.6×10^1	4.603×10^1
4.606×10^1	4.609×10^1	4.612×10^1	4.616×10^1	4.619×10^1
4.622×10^1	4.625×10^1	4.628×10^1	4.631×10^1	4.634×10^1
4.638×10^1	4.641×10^1	4.644×10^1	4.647×10^1	4.65×10^1
4.653×10^1	4.656×10^1	4.659×10^1	4.662×10^1	4.666×10^1
4.669×10^1	4.672×10^1	4.675×10^1	4.678×10^1	4.681×10^1
4.684×10^1	4.688×10^1	4.691×10^1	4.694×10^1	4.697×10^1
4.7×10^1	4.703×10^1	4.706×10^1	4.709×10^1	4.713×10^1
4.716×10^1	4.719×10^1	4.722×10^1	4.725×10^1	4.728×10^1
4.731×10^1	4.734×10^1	4.737×10^1	4.741×10^1	4.744×10^1
4.747×10^1	4.75×10^1	4.753×10^1	4.756×10^1	4.759×10^1
4.763×10^1	4.766×10^1	4.769×10^1	4.772×10^1	4.775×10^1
4.778×10^1	4.781×10^1	4.784×10^1	4.787×10^1	4.791×10^1
4.794×10^1	4.797×10^1	4.8×10^1	4.803×10^1	4.806×10^1
4.809×10^1	4.812×10^1	4.816×10^1	4.819×10^1	4.822×10^1
4.825×10^1	4.828×10^1	4.831×10^1	4.834×10^1	4.838×10^1
4.841×10^1	4.844×10^1	4.847×10^1	4.85×10^1	4.853×10^1
4.856×10^1	4.859×10^1	4.862×10^1	4.866×10^1	4.869×10^1
4.872×10^1	4.875×10^1	4.878×10^1	4.881×10^1	4.884×10^1
4.888×10^1	4.891×10^1	4.894×10^1	4.897×10^1	4.9×10^1
4.903×10^1	4.906×10^1	4.909×10^1	4.912×10^1	4.916×10^1
4.919×10^1	4.922×10^1	4.925×10^1	4.928×10^1	4.931×10^1
4.934×10^1	4.938×10^1	4.941×10^1	4.944×10^1	4.947×10^1
4.95×10^1	4.953×10^1	4.956×10^1	4.959×10^1	4.963×10^1
4.966×10^1	4.969×10^1	4.972×10^1	4.975×10^1	4.978×10^1
4.981×10^1	4.984×10^1	4.987×10^1	4.991×10^1	4.994×10^1
4.997×10^1	5×10^1	5.003×10^1	5.006×10^1	5.009×10^1
5.013×10^1	5.016×10^1	5.019×10^1	5.022×10^1	5.025×10^1
5.028×10^1	5.031×10^1	5.034×10^1	5.037×10^1	5.041×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
5.044×10^{1}	5.047×10^{1}	5.05×10^{1}	5.053×10^{1}	5.056×10^{1}
5.059×10^{1}	5.062×10^{1}	5.066×10^{1}	5.069×10^{1}	5.072×10^{1}
5.075×10^{1}	5.078×10^{1}	5.081×10^{1}	5.084×10^{1}	5.088×10^{1}
5.091×10^{1}	5.094×10^{1}	5.097×10^{1}	5.1×10^{1}	5.103×10^{1}
5.106×10^{1}	5.109×10^{1}	5.112×10^{1}	5.116×10^{1}	5.119×10^{1}
5.122×10^{1}	5.125×10^{1}	5.128×10^{1}	5.131×10^{1}	5.134×10^{1}
5.138×10^{1}	5.141×10^{1}	5.144×10^{1}	5.147×10^{1}	5.15×10^{1}
5.153×10^{1}	5.156×10^{1}	5.159×10^{1}	5.162×10^{1}	5.166×10^{1}
5.169×10^{1}	5.172×10^{1}	5.175×10^{1}	5.178×10^{1}	5.181×10^{1}
5.184×10^{1}	5.188×10^{1}	5.191×10^{1}	5.194×10^{1}	5.197×10^{1}
5.2×10^{1}	5.203×10^{1}	5.206×10^{1}	5.209×10^{1}	5.213×10^{1}
5.216×10^{1}	5.219×10^{1}	5.222×10^{1}	5.225×10^{1}	5.228×10^{1}
5.231×10^{1}	5.234×10^{1}	5.237×10^{1}	5.241×10^{1}	5.244×10^{1}
5.247×10^{1}	5.25×10^{1}	5.253×10^{1}	5.256×10^{1}	5.259×10^{1}
5.263×10^{1}	5.266×10^{1}	5.269×10^{1}	5.272×10^{1}	5.275×10^{1}
5.278×10^{1}	5.281×10^{1}	5.284×10^{1}	5.287×10^{1}	5.291×10^{1}
5.294×10^{1}	5.297×10^{1}	5.3×10^{1}	5.303×10^{1}	5.306×10^{1}
5.309×10^{1}	5.312×10^{1}	5.316×10^{1}	5.319×10^{1}	5.322×10^{1}
5.325×10^{1}	5.328×10^{1}	5.331×10^{1}	5.334×10^{1}	5.338×10^{1}
5.341×10^{1}	5.344×10^{1}	5.347×10^{1}	5.35×10^{1}	5.353×10^{1}
5.356×10^{1}	5.359×10^{1}	5.362×10^{1}	5.366×10^{1}	5.369×10^{1}
5.372×10^{1}	5.375×10^{1}	5.378×10^{1}	5.381×10^{1}	5.384×10^{1}
5.388×10^{1}	5.391×10^{1}	5.394×10^{1}	5.397×10^{1}	5.4×10^{1}
5.403×10^{1}	5.406×10^{1}	5.409×10^{1}	5.412×10^{1}	5.416×10^{1}
5.419×10^{1}	5.422×10^{1}	5.425×10^{1}	5.428×10^{1}	5.431×10^{1}
5.434×10^{1}	5.438×10^{1}	5.441×10^{1}	5.444×10^{1}	5.447×10^{1}
5.45×10^{1}	5.453×10^{1}	5.456×10^{1}	5.459×10^{1}	5.463×10^{1}
5.466×10^{1}	5.469×10^{1}	5.472×10^{1}	5.475×10^{1}	5.478×10^{1}
5.481×10^{1}	5.484×10^{1}	5.487×10^{1}	5.491×10^{1}	5.494×10^{1}
5.497×10^{1}	5.5×10^{1}	5.503×10^{1}	5.506×10^{1}	5.509×10^{1}
5.513×10^{1}	5.516×10^{1}	5.519×10^{1}	5.522×10^{1}	5.525×10^{1}
5.528×10^{1}	5.531×10^{1}	5.534×10^{1}	5.537×10^{1}	5.541×10^{1}
5.544×10^{1}	5.547×10^{1}	5.55×10^{1}	5.553×10^{1}	5.556×10^{1}
5.559×10^{1}	5.562×10^{1}	5.566×10^{1}	5.569×10^{1}	5.572×10^{1}
5.575×10^{1}	5.578×10^{1}	5.581×10^{1}	5.584×10^{1}	5.588×10^{1}
5.591×10^{1}	5.594×10^{1}	5.597×10^{1}	5.6×10^{1}	5.603×10^{1}
5.606×10^{1}	5.609×10^{1}	5.612×10^{1}	5.616×10^{1}	5.619×10^{1}
5.622×10^{1}	5.625×10^{1}	5.628×10^{1}	5.631×10^{1}	5.634×10^{1}
5.638×10^{1}	5.641×10^{1}	5.644×10^{1}	5.647×10^{1}	5.65×10^{1}
5.653×10^{1}	5.656×10^{1}	5.659×10^{1}	5.662×10^{1}	5.666×10^{1}
5.669×10^{1}	5.672×10^{1}	5.675×10^{1}	5.678×10^{1}	5.681×10^{1}
5.684×10^{1}	5.688×10^{1}	5.691×10^{1}	5.694×10^{1}	5.697×10^{1}
5.7×10^{1}	5.703×10^{1}	5.706×10^{1}	5.709×10^{1}	5.713×10^{1}
5.716×10^{1}	5.719×10^{1}	5.722×10^{1}	5.725×10^{1}	5.728×10^{1}
5.731×10^{1}	5.734×10^{1}	5.737×10^{1}	5.741×10^{1}	5.744×10^{1}
5.747×10^{1}	5.75×10^{1}	5.753×10^{1}	5.756×10^{1}	5.759×10^{1}
5.763×10^{1}	5.766×10^{1}	5.769×10^{1}	5.772×10^{1}	5.775×10^{1}
5.778×10^{1}	5.781×10^{1}	5.784×10^{1}	5.787×10^{1}	5.791×10^{1}
5.794×10^{1}	5.797×10^{1}	5.8×10^{1}	5.803×10^{1}	5.806×10^{1}
5.809×10^{1}	5.812×10^{1}	5.816×10^{1}	5.819×10^{1}	5.822×10^{1}

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
5.825×10^1	5.828×10^1	5.831×10^1	5.834×10^1	5.838×10^1
5.841×10^1	5.844×10^1	5.847×10^1	5.85×10^1	5.853×10^1
5.856×10^1	5.859×10^1	5.862×10^1	5.866×10^1	5.869×10^1
5.872×10^1	5.875×10^1	5.878×10^1	5.881×10^1	5.884×10^1
5.888×10^1	5.891×10^1	5.894×10^1	5.897×10^1	5.9×10^1
5.903×10^1	5.906×10^1	5.909×10^1	5.912×10^1	5.916×10^1
5.919×10^1	5.922×10^1	5.925×10^1	5.928×10^1	5.931×10^1
5.934×10^1	5.938×10^1	5.941×10^1	5.944×10^1	5.947×10^1
5.95×10^1	5.953×10^1	5.956×10^1	5.959×10^1	5.963×10^1
5.966×10^1	5.969×10^1	5.972×10^1	5.975×10^1	5.978×10^1
5.981×10^1	5.984×10^1	5.987×10^1	5.991×10^1	5.994×10^1
5.997×10^1	6×10^1	6.003×10^1	6.006×10^1	6.009×10^1
6.013×10^1	6.016×10^1	6.019×10^1	6.022×10^1	6.025×10^1
6.028×10^1	6.031×10^1	6.034×10^1	6.037×10^1	6.041×10^1
6.044×10^1	6.047×10^1	6.05×10^1	6.053×10^1	6.056×10^1
6.059×10^1	6.062×10^1	6.066×10^1	6.069×10^1	6.072×10^1
6.075×10^1	6.078×10^1	6.081×10^1	6.084×10^1	6.088×10^1
6.091×10^1	6.094×10^1	6.097×10^1	6.1×10^1	6.103×10^1
6.106×10^1	6.109×10^1	6.112×10^1	6.116×10^1	6.119×10^1
6.122×10^1	6.125×10^1	6.128×10^1	6.131×10^1	6.134×10^1
6.138×10^1	6.141×10^1	6.144×10^1	6.147×10^1	6.15×10^1
6.153×10^1	6.156×10^1	6.159×10^1	6.162×10^1	6.166×10^1
6.169×10^1	6.172×10^1	6.175×10^1	6.178×10^1	6.181×10^1
6.184×10^1	6.188×10^1	6.191×10^1	6.194×10^1	6.197×10^1
6.2×10^1	6.203×10^1	6.206×10^1	6.209×10^1	6.213×10^1
6.216×10^1	6.219×10^1	6.222×10^1	6.225×10^1	6.228×10^1
6.231×10^1	6.234×10^1	6.237×10^1	6.241×10^1	6.244×10^1
6.247×10^1	6.25×10^1	6.253×10^1	6.256×10^1	6.259×10^1
6.263×10^1	6.266×10^1	6.269×10^1	6.272×10^1	6.275×10^1
6.278×10^1	6.281×10^1	6.284×10^1	6.287×10^1	6.291×10^1
6.294×10^1	6.297×10^1	6.3×10^1	6.303×10^1	6.306×10^1
6.309×10^1	6.312×10^1	6.316×10^1	6.319×10^1	6.322×10^1
6.325×10^1	6.328×10^1	6.331×10^1	6.334×10^1	6.338×10^1
6.341×10^1	6.344×10^1	6.347×10^1	6.35×10^1	6.353×10^1
6.356×10^1	6.359×10^1	6.362×10^1	6.366×10^1	6.369×10^1
6.372×10^1	6.375×10^1	6.378×10^1	6.381×10^1	6.384×10^1
6.388×10^1	6.391×10^1	6.394×10^1	6.397×10^1	6.4×10^1
6.403×10^1	6.406×10^1	6.409×10^1	6.412×10^1	6.416×10^1
6.419×10^1	6.422×10^1	6.425×10^1	6.428×10^1	6.431×10^1
6.434×10^1	6.438×10^1	6.441×10^1	6.444×10^1	6.447×10^1
6.45×10^1	6.453×10^1	6.456×10^1	6.459×10^1	6.463×10^1
6.466×10^1	6.469×10^1	6.472×10^1	6.475×10^1	6.478×10^1
6.481×10^1	6.484×10^1	6.487×10^1	6.491×10^1	6.494×10^1
6.497×10^1	6.5×10^1	6.503×10^1	6.506×10^1	6.509×10^1
6.513×10^1	6.516×10^1	6.519×10^1	6.522×10^1	6.525×10^1
6.528×10^1	6.531×10^1	6.534×10^1	6.537×10^1	6.541×10^1
6.544×10^1	6.547×10^1	6.55×10^1	6.553×10^1	6.556×10^1
6.559×10^1	6.562×10^1	6.566×10^1	6.569×10^1	6.572×10^1
6.575×10^1	6.578×10^1	6.581×10^1	6.584×10^1	6.588×10^1
6.591×10^1	6.594×10^1	6.597×10^1	6.6×10^1	6.603×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]	**Energy** [MeV]
$6.606{\times}10^1$	$6.609{\times}10^1$	$6.612{\times}10^1$	$6.616{\times}10^1$	$6.619{\times}10^1$
$6.622{\times}10^1$	$6.625{\times}10^1$	$6.628{\times}10^1$	$6.631{\times}10^1$	$6.634{\times}10^1$
$6.638{\times}10^1$	$6.641{\times}10^1$	$6.644{\times}10^1$	$6.647{\times}10^1$	$6.65{\times}10^1$
$6.653{\times}10^1$	$6.656{\times}10^1$	$6.659{\times}10^1$	$6.662{\times}10^1$	$6.666{\times}10^1$
$6.669{\times}10^1$	$6.672{\times}10^1$	$6.675{\times}10^1$	$6.678{\times}10^1$	$6.681{\times}10^1$
$6.684{\times}10^1$	$6.688{\times}10^1$	$6.691{\times}10^1$	$6.694{\times}10^1$	$6.697{\times}10^1$
$6.7{\times}10^1$	$6.703{\times}10^1$	$6.706{\times}10^1$	$6.709{\times}10^1$	$6.713{\times}10^1$
$6.716{\times}10^1$	$6.719{\times}10^1$	$6.722{\times}10^1$	$6.725{\times}10^1$	$6.728{\times}10^1$
$6.731{\times}10^1$	$6.734{\times}10^1$	$6.737{\times}10^1$	$6.741{\times}10^1$	$6.744{\times}10^1$
$6.747{\times}10^1$	$6.75{\times}10^1$	$6.753{\times}10^1$	$6.756{\times}10^1$	$6.759{\times}10^1$
$6.763{\times}10^1$	$6.766{\times}10^1$	$6.769{\times}10^1$	$6.772{\times}10^1$	$6.775{\times}10^1$
$6.778{\times}10^1$	$6.781{\times}10^1$	$6.784{\times}10^1$	$6.787{\times}10^1$	$6.791{\times}10^1$
$6.794{\times}10^1$	$6.797{\times}10^1$	$6.8{\times}10^1$	$6.803{\times}10^1$	$6.806{\times}10^1$
$6.809{\times}10^1$	$6.812{\times}10^1$	$6.816{\times}10^1$	$6.819{\times}10^1$	$6.822{\times}10^1$
$6.825{\times}10^1$	$6.828{\times}10^1$	$6.831{\times}10^1$	$6.834{\times}10^1$	$6.838{\times}10^1$
$6.841{\times}10^1$	$6.844{\times}10^1$	$6.847{\times}10^1$	$6.85{\times}10^1$	$6.853{\times}10^1$
$6.856{\times}10^1$	$6.859{\times}10^1$	$6.862{\times}10^1$	$6.866{\times}10^1$	$6.869{\times}10^1$
$6.872{\times}10^1$	$6.875{\times}10^1$	$6.878{\times}10^1$	$6.881{\times}10^1$	$6.884{\times}10^1$
$6.888{\times}10^1$	$6.891{\times}10^1$	$6.894{\times}10^1$	$6.897{\times}10^1$	$6.9{\times}10^1$
$6.903{\times}10^1$	$6.906{\times}10^1$	$6.909{\times}10^1$	$6.912{\times}10^1$	$6.916{\times}10^1$
$6.919{\times}10^1$	$6.922{\times}10^1$	$6.925{\times}10^1$	$6.928{\times}10^1$	$6.931{\times}10^1$
$6.934{\times}10^1$	$6.938{\times}10^1$	$6.941{\times}10^1$	$6.944{\times}10^1$	$6.947{\times}10^1$
$6.95{\times}10^1$	$6.953{\times}10^1$	$6.956{\times}10^1$	$6.959{\times}10^1$	$6.963{\times}10^1$
$6.966{\times}10^1$	$6.969{\times}10^1$	$6.972{\times}10^1$	$6.975{\times}10^1$	$6.978{\times}10^1$
$6.981{\times}10^1$	$6.984{\times}10^1$	$6.987{\times}10^1$	$6.991{\times}10^1$	$6.994{\times}10^1$
$6.997{\times}10^1$	$7{\times}10^1$	$7.003{\times}10^1$	$7.006{\times}10^1$	$7.009{\times}10^1$
$7.013{\times}10^1$	$7.016{\times}10^1$	$7.019{\times}10^1$	$7.022{\times}10^1$	$7.025{\times}10^1$
$7.028{\times}10^1$	$7.031{\times}10^1$	$7.034{\times}10^1$	$7.037{\times}10^1$	$7.041{\times}10^1$
$7.044{\times}10^1$	$7.047{\times}10^1$	$7.05{\times}10^1$	$7.053{\times}10^1$	$7.056{\times}10^1$
$7.059{\times}10^1$	$7.062{\times}10^1$	$7.066{\times}10^1$	$7.069{\times}10^1$	$7.072{\times}10^1$
$7.075{\times}10^1$	$7.078{\times}10^1$	$7.081{\times}10^1$	$7.084{\times}10^1$	$7.088{\times}10^1$
$7.091{\times}10^1$	$7.094{\times}10^1$	$7.097{\times}10^1$	$7.1{\times}10^1$	$7.103{\times}10^1$
$7.106{\times}10^1$	$7.109{\times}10^1$	$7.112{\times}10^1$	$7.116{\times}10^1$	$7.119{\times}10^1$
$7.122{\times}10^1$	$7.125{\times}10^1$	$7.128{\times}10^1$	$7.131{\times}10^1$	$7.134{\times}10^1$
$7.138{\times}10^1$	$7.141{\times}10^1$	$7.144{\times}10^1$	$7.147{\times}10^1$	$7.15{\times}10^1$
$7.153{\times}10^1$	$7.156{\times}10^1$	$7.159{\times}10^1$	$7.162{\times}10^1$	$7.166{\times}10^1$
$7.169{\times}10^1$	$7.172{\times}10^1$	$7.175{\times}10^1$	$7.178{\times}10^1$	$7.181{\times}10^1$
$7.184{\times}10^1$	$7.188{\times}10^1$	$7.191{\times}10^1$	$7.194{\times}10^1$	$7.197{\times}10^1$
$7.2{\times}10^1$	$7.203{\times}10^1$	$7.206{\times}10^1$	$7.209{\times}10^1$	$7.213{\times}10^1$
$7.216{\times}10^1$	$7.219{\times}10^1$	$7.222{\times}10^1$	$7.225{\times}10^1$	$7.228{\times}10^1$
$7.231{\times}10^1$	$7.234{\times}10^1$	$7.237{\times}10^1$	$7.241{\times}10^1$	$7.244{\times}10^1$
$7.247{\times}10^1$	$7.25{\times}10^1$	$7.253{\times}10^1$	$7.256{\times}10^1$	$7.259{\times}10^1$
$7.263{\times}10^1$	$7.266{\times}10^1$	$7.269{\times}10^1$	$7.272{\times}10^1$	$7.275{\times}10^1$
$7.278{\times}10^1$	$7.281{\times}10^1$	$7.284{\times}10^1$	$7.287{\times}10^1$	$7.291{\times}10^1$
$7.294{\times}10^1$	$7.297{\times}10^1$	$7.3{\times}10^1$	$7.303{\times}10^1$	$7.306{\times}10^1$
$7.309{\times}10^1$	$7.312{\times}10^1$	$7.316{\times}10^1$	$7.319{\times}10^1$	$7.322{\times}10^1$
$7.325{\times}10^1$	$7.328{\times}10^1$	$7.331{\times}10^1$	$7.334{\times}10^1$	$7.338{\times}10^1$
$7.341{\times}10^1$	$7.344{\times}10^1$	$7.347{\times}10^1$	$7.35{\times}10^1$	$7.353{\times}10^1$
$7.356{\times}10^1$	$7.359{\times}10^1$	$7.362{\times}10^1$	$7.366{\times}10^1$	$7.369{\times}10^1$
$7.372{\times}10^1$	$7.375{\times}10^1$	$7.378{\times}10^1$	$7.381{\times}10^1$	$7.384{\times}10^1$

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
$7.388{\times}10^{1}$	$7.391{\times}10^{1}$	$7.394{\times}10^{1}$	$7.397{\times}10^{1}$	$7.4{\times}10^{1}$
$7.403{\times}10^{1}$	$7.406{\times}10^{1}$	$7.409{\times}10^{1}$	$7.412{\times}10^{1}$	$7.416{\times}10^{1}$
$7.419{\times}10^{1}$	$7.422{\times}10^{1}$	$7.425{\times}10^{1}$	$7.428{\times}10^{1}$	$7.431{\times}10^{1}$
$7.434{\times}10^{1}$	$7.438{\times}10^{1}$	$7.441{\times}10^{1}$	$7.444{\times}10^{1}$	$7.447{\times}10^{1}$
$7.45{\times}10^{1}$	$7.453{\times}10^{1}$	$7.456{\times}10^{1}$	$7.459{\times}10^{1}$	$7.463{\times}10^{1}$
$7.466{\times}10^{1}$	$7.469{\times}10^{1}$	$7.472{\times}10^{1}$	$7.475{\times}10^{1}$	$7.478{\times}10^{1}$
$7.481{\times}10^{1}$	$7.484{\times}10^{1}$	$7.487{\times}10^{1}$	$7.491{\times}10^{1}$	$7.494{\times}10^{1}$
$7.497{\times}10^{1}$	$7.5{\times}10^{1}$	$7.503{\times}10^{1}$	$7.506{\times}10^{1}$	$7.509{\times}10^{1}$
$7.513{\times}10^{1}$	$7.516{\times}10^{1}$	$7.519{\times}10^{1}$	$7.522{\times}10^{1}$	$7.525{\times}10^{1}$
$7.528{\times}10^{1}$	$7.531{\times}10^{1}$	$7.534{\times}10^{1}$	$7.537{\times}10^{1}$	$7.541{\times}10^{1}$
$7.544{\times}10^{1}$	$7.547{\times}10^{1}$	$7.55{\times}10^{1}$	$7.553{\times}10^{1}$	$7.556{\times}10^{1}$
$7.559{\times}10^{1}$	$7.562{\times}10^{1}$	$7.566{\times}10^{1}$	$7.569{\times}10^{1}$	$7.572{\times}10^{1}$
$7.575{\times}10^{1}$	$7.578{\times}10^{1}$	$7.581{\times}10^{1}$	$7.584{\times}10^{1}$	$7.588{\times}10^{1}$
$7.591{\times}10^{1}$	$7.594{\times}10^{1}$	$7.597{\times}10^{1}$	$7.6{\times}10^{1}$	$7.603{\times}10^{1}$
$7.606{\times}10^{1}$	$7.609{\times}10^{1}$	$7.612{\times}10^{1}$	$7.616{\times}10^{1}$	$7.619{\times}10^{1}$
$7.622{\times}10^{1}$	$7.625{\times}10^{1}$	$7.628{\times}10^{1}$	$7.631{\times}10^{1}$	$7.634{\times}10^{1}$
$7.638{\times}10^{1}$	$7.641{\times}10^{1}$	$7.644{\times}10^{1}$	$7.647{\times}10^{1}$	$7.65{\times}10^{1}$
$7.653{\times}10^{1}$	$7.656{\times}10^{1}$	$7.659{\times}10^{1}$	$7.662{\times}10^{1}$	$7.666{\times}10^{1}$
$7.669{\times}10^{1}$	$7.672{\times}10^{1}$	$7.675{\times}10^{1}$	$7.678{\times}10^{1}$	$7.681{\times}10^{1}$
$7.684{\times}10^{1}$	$7.688{\times}10^{1}$	$7.691{\times}10^{1}$	$7.694{\times}10^{1}$	$7.697{\times}10^{1}$
$7.7{\times}10^{1}$	$7.703{\times}10^{1}$	$7.706{\times}10^{1}$	$7.709{\times}10^{1}$	$7.713{\times}10^{1}$
$7.716{\times}10^{1}$	$7.719{\times}10^{1}$	$7.722{\times}10^{1}$	$7.725{\times}10^{1}$	$7.728{\times}10^{1}$
$7.731{\times}10^{1}$	$7.734{\times}10^{1}$	$7.737{\times}10^{1}$	$7.741{\times}10^{1}$	$7.744{\times}10^{1}$
$7.747{\times}10^{1}$	$7.75{\times}10^{1}$	$7.753{\times}10^{1}$	$7.756{\times}10^{1}$	$7.759{\times}10^{1}$
$7.763{\times}10^{1}$	$7.766{\times}10^{1}$	$7.769{\times}10^{1}$	$7.772{\times}10^{1}$	$7.775{\times}10^{1}$
$7.778{\times}10^{1}$	$7.781{\times}10^{1}$	$7.784{\times}10^{1}$	$7.787{\times}10^{1}$	$7.791{\times}10^{1}$
$7.794{\times}10^{1}$	$7.797{\times}10^{1}$	$7.8{\times}10^{1}$	$7.803{\times}10^{1}$	$7.806{\times}10^{1}$
$7.809{\times}10^{1}$	$7.812{\times}10^{1}$	$7.816{\times}10^{1}$	$7.819{\times}10^{1}$	$7.822{\times}10^{1}$
$7.825{\times}10^{1}$	$7.828{\times}10^{1}$	$7.831{\times}10^{1}$	$7.834{\times}10^{1}$	$7.838{\times}10^{1}$
$7.841{\times}10^{1}$	$7.844{\times}10^{1}$	$7.847{\times}10^{1}$	$7.85{\times}10^{1}$	$7.853{\times}10^{1}$
$7.856{\times}10^{1}$	$7.859{\times}10^{1}$	$7.862{\times}10^{1}$	$7.866{\times}10^{1}$	$7.869{\times}10^{1}$
$7.872{\times}10^{1}$	$7.875{\times}10^{1}$	$7.878{\times}10^{1}$	$7.881{\times}10^{1}$	$7.884{\times}10^{1}$
$7.888{\times}10^{1}$	$7.891{\times}10^{1}$	$7.894{\times}10^{1}$	$7.897{\times}10^{1}$	$7.9{\times}10^{1}$
$7.903{\times}10^{1}$	$7.906{\times}10^{1}$	$7.909{\times}10^{1}$	$7.912{\times}10^{1}$	$7.916{\times}10^{1}$
$7.919{\times}10^{1}$	$7.922{\times}10^{1}$	$7.925{\times}10^{1}$	$7.928{\times}10^{1}$	$7.931{\times}10^{1}$
$7.934{\times}10^{1}$	$7.938{\times}10^{1}$	$7.941{\times}10^{1}$	$7.944{\times}10^{1}$	$7.947{\times}10^{1}$
$7.95{\times}10^{1}$	$7.953{\times}10^{1}$	$7.956{\times}10^{1}$	$7.959{\times}10^{1}$	$7.963{\times}10^{1}$
$7.966{\times}10^{1}$	$7.969{\times}10^{1}$	$7.972{\times}10^{1}$	$7.975{\times}10^{1}$	$7.978{\times}10^{1}$
$7.981{\times}10^{1}$	$7.984{\times}10^{1}$	$7.987{\times}10^{1}$	$7.991{\times}10^{1}$	$7.994{\times}10^{1}$
$7.997{\times}10^{1}$	$8{\times}10^{1}$	$8.003{\times}10^{1}$	$8.006{\times}10^{1}$	$8.009{\times}10^{1}$
$8.012{\times}10^{1}$	$8.016{\times}10^{1}$	$8.019{\times}10^{1}$	$8.022{\times}10^{1}$	$8.025{\times}10^{1}$
$8.028{\times}10^{1}$	$8.031{\times}10^{1}$	$8.034{\times}10^{1}$	$8.037{\times}10^{1}$	$8.041{\times}10^{1}$
$8.044{\times}10^{1}$	$8.047{\times}10^{1}$	$8.05{\times}10^{1}$	$8.053{\times}10^{1}$	$8.056{\times}10^{1}$
$8.059{\times}10^{1}$	$8.062{\times}10^{1}$	$8.066{\times}10^{1}$	$8.069{\times}10^{1}$	$8.072{\times}10^{1}$
$8.075{\times}10^{1}$	$8.078{\times}10^{1}$	$8.081{\times}10^{1}$	$8.084{\times}10^{1}$	$8.088{\times}10^{1}$
$8.091{\times}10^{1}$	$8.094{\times}10^{1}$	$8.097{\times}10^{1}$	$8.1{\times}10^{1}$	$8.103{\times}10^{1}$
$8.106{\times}10^{1}$	$8.109{\times}10^{1}$	$8.113{\times}10^{1}$	$8.116{\times}10^{1}$	$8.119{\times}10^{1}$
$8.122{\times}10^{1}$	$8.125{\times}10^{1}$	$8.128{\times}10^{1}$	$8.131{\times}10^{1}$	$8.134{\times}10^{1}$
$8.137{\times}10^{1}$	$8.141{\times}10^{1}$	$8.144{\times}10^{1}$	$8.147{\times}10^{1}$	$8.15{\times}10^{1}$
$8.153{\times}10^{1}$	$8.156{\times}10^{1}$	$8.159{\times}10^{1}$	$8.162{\times}10^{1}$	$8.166{\times}10^{1}$

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
8.169×10^1	8.172×10^1	8.175×10^1	8.178×10^1	8.181×10^1
8.184×10^1	8.188×10^1	8.191×10^1	8.194×10^1	8.197×10^1
8.2×10^1	8.203×10^1	8.206×10^1	8.209×10^1	8.213×10^1
8.216×10^1	8.219×10^1	8.222×10^1	8.225×10^1	8.228×10^1
8.231×10^1	8.234×10^1	8.238×10^1	8.241×10^1	8.244×10^1
8.247×10^1	8.25×10^1	8.253×10^1	8.256×10^1	8.259×10^1
8.262×10^1	8.266×10^1	8.269×10^1	8.272×10^1	8.275×10^1
8.278×10^1	8.281×10^1	8.284×10^1	8.287×10^1	8.291×10^1
8.294×10^1	8.297×10^1	8.3×10^1	8.303×10^1	8.306×10^1
8.309×10^1	8.312×10^1	8.316×10^1	8.319×10^1	8.322×10^1
8.325×10^1	8.328×10^1	8.331×10^1	8.334×10^1	8.338×10^1
8.341×10^1	8.344×10^1	8.347×10^1	8.35×10^1	8.353×10^1
8.356×10^1	8.359×10^1	8.363×10^1	8.366×10^1	8.369×10^1
8.372×10^1	8.375×10^1	8.378×10^1	8.381×10^1	8.384×10^1
8.387×10^1	8.391×10^1	8.394×10^1	8.397×10^1	8.4×10^1
8.403×10^1	8.406×10^1	8.409×10^1	8.412×10^1	8.416×10^1
8.419×10^1	8.422×10^1	8.425×10^1	8.428×10^1	8.431×10^1
8.434×10^1	8.438×10^1	8.441×10^1	8.444×10^1	8.447×10^1
8.45×10^1	8.453×10^1	8.456×10^1	8.459×10^1	8.463×10^1
8.466×10^1	8.469×10^1	8.472×10^1	8.475×10^1	8.478×10^1
8.481×10^1	8.484×10^1	8.488×10^1	8.491×10^1	8.494×10^1
8.497×10^1	8.5×10^1	8.503×10^1	8.506×10^1	8.509×10^1
8.512×10^1	8.516×10^1	8.519×10^1	8.522×10^1	8.525×10^1
8.528×10^1	8.531×10^1	8.534×10^1	8.537×10^1	8.541×10^1
8.544×10^1	8.547×10^1	8.55×10^1	8.553×10^1	8.556×10^1
8.559×10^1	8.562×10^1	8.566×10^1	8.569×10^1	8.572×10^1
8.575×10^1	8.578×10^1	8.581×10^1	8.584×10^1	8.588×10^1
8.591×10^1	8.594×10^1	8.597×10^1	8.6×10^1	8.603×10^1
8.606×10^1	8.609×10^1	8.613×10^1	8.616×10^1	8.619×10^1
8.622×10^1	8.625×10^1	8.628×10^1	8.631×10^1	8.634×10^1
8.637×10^1	8.641×10^1	8.644×10^1	8.647×10^1	8.65×10^1
8.653×10^1	8.656×10^1	8.659×10^1	8.662×10^1	8.666×10^1
8.669×10^1	8.672×10^1	8.675×10^1	8.678×10^1	8.681×10^1
8.684×10^1	8.688×10^1	8.691×10^1	8.694×10^1	8.697×10^1
8.7×10^1	8.703×10^1	8.706×10^1	8.709×10^1	8.713×10^1
8.716×10^1	8.719×10^1	8.722×10^1	8.725×10^1	8.728×10^1
8.731×10^1	8.734×10^1	8.738×10^1	8.741×10^1	8.744×10^1
8.747×10^1	8.75×10^1	8.753×10^1	8.756×10^1	8.759×10^1
8.762×10^1	8.766×10^1	8.769×10^1	8.772×10^1	8.775×10^1
8.778×10^1	8.781×10^1	8.784×10^1	8.787×10^1	8.791×10^1
8.794×10^1	8.797×10^1	8.8×10^1	8.803×10^1	8.806×10^1
8.809×10^1	8.812×10^1	8.816×10^1	8.819×10^1	8.822×10^1
8.825×10^1	8.828×10^1	8.831×10^1	8.834×10^1	8.838×10^1
8.841×10^1	8.844×10^1	8.847×10^1	8.85×10^1	8.853×10^1
8.856×10^1	8.859×10^1	8.863×10^1	8.866×10^1	8.869×10^1
8.872×10^1	8.875×10^1	8.878×10^1	8.881×10^1	8.884×10^1
8.887×10^1	8.891×10^1	8.894×10^1	8.897×10^1	8.9×10^1
8.903×10^1	8.906×10^1	8.909×10^1	8.912×10^1	8.916×10^1
8.919×10^1	8.922×10^1	8.925×10^1	8.928×10^1	8.931×10^1
8.934×10^1	8.938×10^1	8.941×10^1	8.944×10^1	8.947×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]	Energy [MeV]
8.95×10^1	8.953×10^1	8.956×10^1	8.959×10^1	8.963×10^1
8.966×10^1	8.969×10^1	8.972×10^1	8.975×10^1	8.978×10^1
8.981×10^1	8.984×10^1	8.988×10^1	8.991×10^1	8.994×10^1
8.997×10^1	9×10^1	9.003×10^1	9.006×10^1	9.009×10^1
9.012×10^1	9.016×10^1	9.019×10^1	9.022×10^1	9.025×10^1
9.028×10^1	9.031×10^1	9.034×10^1	9.037×10^1	9.041×10^1
9.044×10^1	9.047×10^1	9.05×10^1	9.053×10^1	9.056×10^1
9.059×10^1	9.062×10^1	9.066×10^1	9.069×10^1	9.072×10^1
9.075×10^1	9.078×10^1	9.081×10^1	9.084×10^1	9.088×10^1
9.091×10^1	9.094×10^1	9.097×10^1	9.1×10^1	9.103×10^1
9.106×10^1	9.109×10^1	9.113×10^1	9.116×10^1	9.119×10^1
9.122×10^1	9.125×10^1	9.128×10^1	9.131×10^1	9.134×10^1
9.137×10^1	9.141×10^1	9.144×10^1	9.147×10^1	9.15×10^1
9.153×10^1	9.156×10^1	9.159×10^1	9.162×10^1	9.166×10^1
9.169×10^1	9.172×10^1	9.175×10^1	9.178×10^1	9.181×10^1
9.184×10^1	9.188×10^1	9.191×10^1	9.194×10^1	9.197×10^1
9.2×10^1	9.203×10^1	9.206×10^1	9.209×10^1	9.213×10^1
9.216×10^1	9.219×10^1	9.222×10^1	9.225×10^1	9.228×10^1
9.231×10^1	9.234×10^1	9.238×10^1	9.241×10^1	9.244×10^1
9.247×10^1	9.25×10^1	9.253×10^1	9.256×10^1	9.259×10^1
9.262×10^1	9.266×10^1	9.269×10^1	9.272×10^1	9.275×10^1
9.278×10^1	9.281×10^1	9.284×10^1	9.287×10^1	9.291×10^1
9.294×10^1	9.297×10^1	9.3×10^1	9.303×10^1	9.306×10^1
9.309×10^1	9.312×10^1	9.316×10^1	9.319×10^1	9.322×10^1
9.325×10^1	9.328×10^1	9.331×10^1	9.334×10^1	9.338×10^1
9.341×10^1	9.344×10^1	9.347×10^1	9.35×10^1	9.353×10^1
9.356×10^1	9.359×10^1	9.363×10^1	9.366×10^1	9.369×10^1
9.372×10^1	9.375×10^1	9.378×10^1	9.381×10^1	9.384×10^1
9.387×10^1	9.391×10^1	9.394×10^1	9.397×10^1	9.4×10^1
9.403×10^1	9.406×10^1	9.409×10^1	9.412×10^1	9.416×10^1
9.419×10^1	9.422×10^1	9.425×10^1	9.428×10^1	9.431×10^1
9.434×10^1	9.438×10^1	9.441×10^1	9.444×10^1	9.447×10^1
9.45×10^1	9.453×10^1	9.456×10^1	9.459×10^1	9.463×10^1
9.466×10^1	9.469×10^1	9.472×10^1	9.475×10^1	9.478×10^1
9.481×10^1	9.484×10^1	9.488×10^1	9.491×10^1	9.494×10^1
9.497×10^1	9.5×10^1	9.503×10^1	9.506×10^1	9.509×10^1
9.512×10^1	9.516×10^1	9.519×10^1	9.522×10^1	9.525×10^1
9.528×10^1	9.531×10^1	9.534×10^1	9.537×10^1	9.541×10^1
9.544×10^1	9.547×10^1	9.55×10^1	9.553×10^1	9.556×10^1
9.559×10^1	9.562×10^1	9.566×10^1	9.569×10^1	9.572×10^1
9.575×10^1	9.578×10^1	9.581×10^1	9.584×10^1	9.588×10^1
9.591×10^1	9.594×10^1	9.597×10^1	9.6×10^1	9.603×10^1
9.606×10^1	9.609×10^1	9.613×10^1	9.616×10^1	9.619×10^1
9.622×10^1	9.625×10^1	9.628×10^1	9.631×10^1	9.634×10^1
9.637×10^1	9.641×10^1	9.644×10^1	9.647×10^1	9.65×10^1
9.653×10^1	9.656×10^1	9.659×10^1	9.662×10^1	9.666×10^1
9.669×10^1	9.672×10^1	9.675×10^1	9.678×10^1	9.681×10^1
9.684×10^1	9.688×10^1	9.691×10^1	9.694×10^1	9.697×10^1
9.7×10^1	9.703×10^1	9.706×10^1	9.709×10^1	9.713×10^1
9.716×10^1	9.719×10^1	9.722×10^1	9.725×10^1	9.728×10^1

Continued on next page

Table A.2 – continued from previous page

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
9.731×10^1	9.734×10^1	9.738×10^1	9.741×10^1	9.744×10^1
9.747×10^1	9.75×10^1	9.753×10^1	9.756×10^1	9.759×10^1
9.762×10^1	9.766×10^1	9.769×10^1	9.772×10^1	9.775×10^1
9.778×10^1	9.781×10^1	9.784×10^1	9.787×10^1	9.791×10^1
9.794×10^1	9.797×10^1	9.8×10^1	9.803×10^1	9.806×10^1
9.809×10^1	9.812×10^1	9.816×10^1	9.819×10^1	9.822×10^1
9.825×10^1	9.828×10^1	9.831×10^1	9.834×10^1	9.838×10^1
9.841×10^1	9.844×10^1	9.847×10^1	9.85×10^1	9.853×10^1
9.856×10^1	9.859×10^1	9.863×10^1	9.866×10^1	9.869×10^1
9.872×10^1	9.875×10^1	9.878×10^1	9.881×10^1	9.884×10^1
9.887×10^1	9.891×10^1	9.894×10^1	9.897×10^1	9.9×10^1
9.903×10^1	9.906×10^1	9.909×10^1	9.912×10^1	9.916×10^1
9.919×10^1	9.922×10^1	9.925×10^1	9.928×10^1	9.931×10^1
9.934×10^1	9.938×10^1	9.941×10^1	9.944×10^1	9.947×10^1
9.95×10^1	9.953×10^1	9.956×10^1	9.959×10^1	9.963×10^1
9.966×10^1	9.969×10^1	9.972×10^1	9.975×10^1	9.978×10^1
9.981×10^1	9.984×10^1	9.988×10^1	9.991×10^1	9.994×10^1
9.997×10^1	1×10^2			

A.2 Neutron Energy and Time Structure

Table A.3: Neutron Tally Energy Bins

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
1×10^{-10}	1.39012×10^{-10}	1.93242×10^{-10}	2.68629×10^{-10}
3.73426×10^{-10}	5.19105×10^{-10}	7.21616×10^{-10}	1.00313×10^{-9}
1.39447×10^{-9}	1.93847×10^{-9}	2.6947×10^{-9}	3.74594×10^{-9}
5.20729×10^{-9}	7.23874×10^{-9}	1.00627×10^{-8}	1.39883×10^{-8}
1.94454×10^{-8}	2.70313×10^{-8}	3.75767×10^{-8}	5.22359×10^{-8}
7.2614×10^{-8}	1.00942×10^{-7}	1.40321×10^{-7}	1.95062×10^{-7}
2.71159×10^{-7}	3.76943×10^{-7}	5.23994×10^{-7}	7.28412×10^{-7}
1.01258×10^{-6}	1.4076×10^{-6}	1.95673×10^{-6}	2.72008×10^{-6}
3.78122×10^{-6}	5.25634×10^{-6}	7.30692×10^{-6}	1.01575×10^{-5}
1.41201×10^{-5}	1.96285×10^{-5}	2.72859×10^{-5}	3.79306×10^{-5}
5.27279×10^{-5}	7.32979×10^{-5}	0.000101893	0.000141642
0.000196899	0.000273713	0.000380493	0.000528929
0.000735273	0.00102211	0.00142086	0.00197516
0.0027457	0.00381684	0.00530585	0.00737574
0.0102531	0.014253	0.0198134	0.0275429
0.0382878	0.0532245	0.0739883	0.102852
0.142977	0.198754	0.276291	0.384077
0.533911	0.742198	1.03174	1.43424
1.99376	2.77156	3.85279	5.35582
7.44521	10.3497	14.3873	20

Table A.4: Neutron Tally Energy Bins

Energy [MeV]	**Energy [MeV]**	**Energy [MeV]**	**Energy [MeV]**
0.0001	0.000132194	0.000174753	0.000231013
0.000305386	0.000403702	0.00053367	0.00070548
0.000932603	0.00123285	0.00162975	0.00215443
0.00284804	0.00376494	0.00497702	0.00657933
0.00869749	0.0114976	0.0151991	0.0200923
0.0265609	0.0351119	0.0464159	0.0613591
0.0811131	0.107227	0.141747	0.187382
0.247708	0.327455	0.432876	0.572237
0.756463	1	1.32194	1.74753
2.31013	3.05386	4.03702	5.3367
7.0548	9.32603	12.3285	16.2975
21.5443	28.4804	37.6494	49.7702
65.7933	86.9749	114.976	151.991
200.923	265.609	351.119	464.159
613.591	811.131	1072.27	1417.47
1873.82	2477.08	3274.55	4328.76
5722.37	7564.63	10000	13219.4
17475.3	23101.3	30538.6	40370.2
53367	70548	93260.3	123285
162975	215443	284804	376494
497702	657933	869749	1.14976×10^6
1.51991×10^6	2.00923×10^6	2.65609×10^6	3.51119×10^6
4.64159×10^6	6.13591×10^6	8.11131×10^6	1.07227×10^7
1.41747×10^7	1.87382×10^7	2.47708×10^7	3.27455×10^7
4.32876×10^7	5.72237×10^7	7.56463×10^7	1×10^8

A.3 MCNPX Input Files

Listing A.1: Example SNM Input File

```
Title
1 1 -18.95 -1
2 0 1

1 so 2.3268675430e+00

ctme 1440
dbcn 21932572048705
phys:p 100.1
mode p
sdef rad=d1 par=p erg=1.0000000000e-02
si1 0 2.3268675430e+00
sp1 -21 2
sb1 -21 5
imp:p 1 0
f1:p 1
ft1 tag 3
tf1 j j 1 j j j 2 j
fu1 -1      00000.00001  00000.00003  00000.00004      0              1e10
e0
       1.0000000000e-03
       1.9765548681e-03
       2.9531097362e-03
       3.9296646044e-03
       4.9062194725e-03
       5.8827743406e-03
       6.8593292087e-03
       7.8358840768e-03
       8.8124389450e-03
       9.7889938131e-03
       1.0765548681e-02
c Material Cards
m1 92000 -1.0
prdmp j j 1
vect V2 0 0 0
ext:p -SV2 0
wwg  1 0 0 j j j j 0
wwge:p 0.01 0.02 0.03 0.05 0.1 0.3 0.5 1.0 3.0 10.0 30.0 50.0 100.1
mesh geom=sph ref=0 0 0 origin=0 0 0
       imesh=2.3268775430e+00
       iints=100
       jmesh=0.5
       kmesh=1.0
```

Listing A.2: Example Shield Input File

```
Title
1 1 -11.36 -1
2 1 -11.36 1 -2
3 1 -11.36 -3 2
4 1 -11.36 -4 3
5 1 -11.36 -5 4
6 1 -11.36 -6 5
7 1 -11.36 -7 6
8 1 -11.36 -8 7
9 1 -11.36 -9 8
10 1 -11.36 -10 9
11 1 -11.36 -11 10
12 1 -11.36 -12 11
13 1 -11.36 -13 12
14 1 -11.36 -14 13
15 1 -11.36 -15 14
16 1 -11.36 -16 15
17 1 -11.36 -17 16
18 1 -11.36 -18 17
19 1 -11.36 -19 18
20 1 -11.36 -20 19
21 1 -11.36 -21 20
22 1 -11.36 -22 21
```

23	1	−11.36	−23	22
24	1	−11.36	−24	23
25	1	−11.36	−25	24
26	1	−11.36	−26	25
27	1	−11.36	−27	26
28	1	−11.36	−28	27
29	1	−11.36	−29	28
30	1	−11.36	−30	29
31	1	−11.36	−31	30
32	1	−11.36	−32	31
33	1	−11.36	−33	32
34	1	−11.36	−34	33
35	1	−11.36	−35	34
36	1	−11.36	−36	35
37	1	−11.36	−37	36
38	1	−11.36	−38	37
39	1	−11.36	−39	38
40	1	−11.36	−40	39
41	1	−11.36	−41	40
42	1	−11.36	−42	41
43	1	−11.36	−43	42
44	1	−11.36	−44	43
45	1	−11.36	−45	44
46	1	−11.36	−46	45
47	1	−11.36	−47	46
48	1	−11.36	−48	47
49	1	−11.36	−49	48
50	1	−11.36	−50	49
51	1	−11.36	−51	50
52	1	−11.36	−52	51
53	1	−11.36	−53	52
54	1	−11.36	−54	53
55	1	−11.36	−55	54
56	1	−11.36	−56	55
57	1	−11.36	−57	56
58	1	−11.36	−58	57
59	1	−11.36	−59	58
60	1	−11.36	−60	59
61	1	−11.36	−61	60
62	1	−11.36	−62	61
63	1	−11.36	−63	62
64	1	−11.36	−64	63
65	1	−11.36	−65	64
66	1	−11.36	−66	65
67	1	−11.36	−67	66
68	1	−11.36	−68	67
69	1	−11.36	−69	68
70	1	−11.36	−70	69
71	1	−11.36	−71	70
72	1	−11.36	−72	71
73	1	−11.36	−73	72
74	1	−11.36	−74	73
75	1	−11.36	−75	74
76	1	−11.36	−76	75
77	1	−11.36	−77	76
78	1	−11.36	−78	77
79	1	−11.36	−79	78
80	1	−11.36	−80	79
81	1	−11.36	−81	80
82	1	−11.36	−82	81
83	1	−11.36	−83	82
84	1	−11.36	−84	83
85	1	−11.36	−85	84
86	1	−11.36	−86	85
87	1	−11.36	−87	86
88	1	−11.36	−88	87
89	1	−11.36	−89	88
90	1	−11.36	−90	89
91	1	−11.36	−91	90
92	1	−11.36	−92	91
93	1	−11.36	−93	92
94	1	−11.36	−94	93
95	1	−11.36	−95	94
96	1	−11.36	−96	95
97	1	−11.36	−97	96
98	1	−11.36	−98	97

99	1	−11.36	−99	98
100	1	−11.36	−100	99
101	1	−11.36	−101	100
102	1	−11.36	−102	101
103	1	−11.36	−103	102
104	1	−11.36	−104	103
105	1	−11.36	−105	104
106	1	−11.36	−106	105
107	1	−11.36	−107	106
108	1	−11.36	−108	107
109	1	−11.36	−109	108
110	1	−11.36	−110	109
111	1	−11.36	−111	110
112	1	−11.36	−112	111
113	1	−11.36	−113	112
114	1	−11.36	−114	113
115	1	−11.36	−115	114
116	1	−11.36	−116	115
117	1	−11.36	−117	116
118	1	−11.36	−118	117
119	1	−11.36	−119	118
120	1	−11.36	−120	119
121	1	−11.36	−121	120
122	1	−11.36	−122	121
123	1	−11.36	−123	122
124	1	−11.36	−124	123
125	1	−11.36	−125	124
126	1	−11.36	−126	125
127	1	−11.36	−127	126
128	1	−11.36	−128	127
129	1	−11.36	−129	128
130	1	−11.36	−130	129
131	1	−11.36	−131	130
132	1	−11.36	−132	131
133	1	−11.36	−133	132
134	1	−11.36	−134	133
135	1	−11.36	−135	134
136	1	−11.36	−136	135
137	1	−11.36	−137	136
138	1	−11.36	−138	137
139	1	−11.36	−139	138
140	1	−11.36	−140	139
141	1	−11.36	−141	140
142	1	−11.36	−142	141
143	1	−11.36	−143	142
144	1	−11.36	−144	143
145	1	−11.36	−145	144
146	1	−11.36	−146	145
147	1	−11.36	−147	146
148	1	−11.36	−148	147
149	1	−11.36	−149	148
150	1	−11.36	−150	149
151	1	−11.36	−151	150
152	1	−11.36	−152	151
153	1	−11.36	−153	152
154	1	−11.36	−154	153
155	1	−11.36	−155	154
156	1	−11.36	−156	155
157	1	−11.36	−157	156
158	1	−11.36	−158	157
159	1	−11.36	−159	158
160	1	−11.36	−160	159
161	1	−11.36	−161	160
162	1	−11.36	−162	161
163	1	−11.36	−163	162
164	1	−11.36	−164	163
165	1	−11.36	−165	164
166	1	−11.36	−166	165
167	1	−11.36	−167	166
168	1	−11.36	−168	167
169	1	−11.36	−169	168
170	1	−11.36	−170	169
171	1	−11.36	−171	170
172	1	−11.36	−172	171
173	1	−11.36	−173	172
174	1	−11.36	−174	173

```
175 1 −11.36 −175 174
176 1 −11.36 −176 175
177 1 −11.36 −177 176
178 1 −11.36 −178 177
179 1 −11.36 −179 178
180 1 −11.36 −180 179
181 1 −11.36 −181 180
182 1 −11.36 −182 181
183 1 −11.36 −183 182
184 1 −11.36 −184 183
185 1 −11.36 −185 184
186 1 −11.36 −186 185
187 1 −11.36 −187 186
188 1 −11.36 −188 187
189 1 −11.36 −189 188
190 1 −11.36 −190 189
191 1 −11.36 −191 190
192 1 −11.36 −192 191
193 1 −11.36 −193 192
194 1 −11.36 −194 193
195 1 −11.36 −195 194
196 1 −11.36 −196 195
197 1 −11.36 −197 196
198 1 −11.36 −198 197
199 1 −11.36 −199 198
201 0 199

1 so 1.0000000000e−04
2 so 1.0010000000e−04
3 so 4.6577463482e−04
4 so 4.6587463482e−04
5 so 9.3630207204e−04
6 so 9.3640207204e−04
7 so 1.4068295093e−03
8 so 1.4069295093e−03
9 so 1.8773569465e−03
10 so 1.8774569465e−03
11 so 2.3478843837e−03
12 so 2.3479843837e−03
13 so 2.8184118209e−03
14 so 2.8185118209e−03
15 so 3.2889392581e−03
16 so 3.2890392581e−03
17 so 3.7594666954e−03
18 so 3.7595666954e−03
19 so 4.2299941326e−03
20 so 4.2300941326e−03
21 so 4.7005215698e−03
22 so 4.7006215698e−03
23 so 5.1710490070e−03
24 so 5.1711490070e−03
25 so 5.6415764442e−03
26 so 5.6416764442e−03
27 so 6.1121038815e−03
28 so 6.1122038815e−03
29 so 6.5826313187e−03
30 so 6.5827313187e−03
31 so 7.0531587559e−03
32 so 7.0532587559e−03
33 so 7.5236861931e−03
34 so 7.5237861931e−03
35 so 7.9942136303e−03
36 so 7.9943136303e−03
37 so 8.4647410676e−03
38 so 8.4648410676e−03
39 so 8.9352685048e−03
40 so 8.9353685048e−03
41 so 9.4057959420e−03
42 so 9.4058959420e−03
43 so 9.8763233792e−03
44 so 9.8764233792e−03
45 so 1.0346850816e−02
46 so 1.0346950816e−02
47 so 1.0817378254e−02
48 so 1.0817478254e−02
49 so 1.1287905691e−02
```

50 so 1.1288005691e−02
51 so 1.1758433128e−02
52 so 1.1758533128e−02
53 so 1.2228960565e−02
54 so 1.2229060565e−02
55 so 1.2699488003e−02
56 so 1.2699588003e−02
57 so 1.3170015440e−02
58 so 1.3170115440e−02
59 so 1.3640542877e−02
60 so 1.3640642877e−02
61 so 1.4111070314e−02
62 so 1.4111170314e−02
63 so 1.4581597751e−02
64 so 1.4581697751e−02
65 so 1.5052125189e−02
66 so 1.5052225189e−02
67 so 1.5522652626e−02
68 so 1.5522752626e−02
69 so 1.5993180063e−02
70 so 1.5993280063e−02
71 so 1.6463707500e−02
72 so 1.6463807500e−02
73 so 1.6934234938e−02
74 so 1.6934334938e−02
75 so 1.7404762375e−02
76 so 1.7404862375e−02
77 so 1.7875289812e−02
78 so 1.7875389812e−02
79 so 1.8345817249e−02
80 so 1.8345917249e−02
81 so 1.8816344686e−02
82 so 1.8816444686e−02
83 so 1.9286872124e−02
84 so 1.9286972124e−02
85 so 1.9757399561e−02
86 so 1.9757499561e−02
87 so 2.0227926998e−02
88 so 2.0228026998e−02
89 so 2.0698454435e−02
90 so 2.0698554435e−02
91 so 2.1168981873e−02
92 so 2.1169081873e−02
93 so 2.1639509310e−02
94 so 2.1639609310e−02
95 so 2.2110036747e−02
96 so 2.2110136747e−02
97 so 2.2580564184e−02
98 so 2.2580664184e−02
99 so 2.3051091621e−02
100 so 2.3051191621e−02
101 so 2.3521619059e−02
102 so 2.3521719059e−02
103 so 2.3992146496e−02
104 so 2.3992246496e−02
105 so 2.4462673933e−02
106 so 2.4462773933e−02
107 so 2.4933201370e−02
108 so 2.4933301370e−02
109 so 2.5403728808e−02
110 so 2.5403828808e−02
111 so 2.5874256245e−02
112 so 2.5874356245e−02
113 so 2.6344783682e−02
114 so 2.6344883682e−02
115 so 2.6815311119e−02
116 so 2.6815411119e−02
117 so 2.7285838556e−02
118 so 2.7285938556e−02
119 so 2.7756365994e−02
120 so 2.7756465994e−02
121 so 2.8226893431e−02
122 so 2.8226993431e−02
123 so 2.8697420868e−02
124 so 2.8697520868e−02
125 so 2.9167948305e−02

```
126 so 2.9168048305e-02
127 so 2.9638475742e-02
128 so 2.9638575742e-02
129 so 3.0109003180e-02
130 so 3.0109103180e-02
131 so 3.0579530617e-02
132 so 3.0579630617e-02
133 so 3.1050058054e-02
134 so 3.1050158054e-02
135 so 3.1520585491e-02
136 so 3.1520685491e-02
137 so 3.1991112929e-02
138 so 3.1991212929e-02
139 so 3.2461640366e-02
140 so 3.2461740366e-02
141 so 3.2932167803e-02
142 so 3.2932267803e-02
143 so 3.3402695240e-02
144 so 3.3402795240e-02
145 so 3.3873222677e-02
146 so 3.3873322677e-02
147 so 3.4343750115e-02
148 so 3.4343850115e-02
149 so 3.4814277552e-02
150 so 3.4814377552e-02
151 so 3.5284804989e-02
152 so 3.5284904989e-02
153 so 3.5755332426e-02
154 so 3.5755432426e-02
155 so 3.6225859864e-02
156 so 3.6225959864e-02
157 so 3.6696387301e-02
158 so 3.6696487301e-02
159 so 3.7166914738e-02
160 so 3.7167014738e-02
161 so 3.7637442175e-02
162 so 3.7637542175e-02
163 so 3.8107969612e-02
164 so 3.8108069612e-02
165 so 3.8578497050e-02
166 so 3.8578597050e-02
167 so 3.9049024487e-02
168 so 3.9049124487e-02
169 so 3.9519551924e-02
170 so 3.9519651924e-02
171 so 3.9990079361e-02
172 so 3.9990179361e-02
173 so 4.0460606799e-02
174 so 4.0460706799e-02
175 so 4.0931134236e-02
176 so 4.0931234236e-02
177 so 4.1401661673e-02
178 so 4.1401761673e-02
179 so 4.1872189110e-02
180 so 4.1872289110e-02
181 so 4.2342716547e-02
182 so 4.2342816547e-02
183 so 4.2813243985e-02
184 so 4.2813343985e-02
185 so 4.3283771422e-02
186 so 4.3283871422e-02
187 so 4.3754298859e-02
188 so 4.3754398859e-02
189 so 4.4224826296e-02
190 so 4.4224926296e-02
191 so 4.4695353734e-02
192 so 4.4695453734e-02
193 so 4.5165881171e-02
194 so 4.5165981171e-02
195 so 4.5636408608e-02
196 so 4.5636508608e-02
197 so 4.6106936045e-02
198 so 4.6107036045e-02
199 so 4.6577463482e-02

phys:p 100.1
```

```
cut:p j j -1e-29 -1e-30
mode p
sdef pos=0 0 0 par=p erg=1.0000000000e-02
f1:p 1
sf1 2
ft1 tag 3
fu1 -1      00000.00001 00000.00003 00000.00004      0              1e10
f11:p 3
sf11 4
ft11 tag 3
fu11 -1      00000.00001 00000.00003 00000.00004      0              1e10
f21:p 5
sf21 6
ft21 tag 3
fu21 -1      00000.00001 00000.00003 00000.00004      0              1e10
f31:p 7
sf31 8
ft31 tag 3
fu31 -1      00000.00001 00000.00003 00000.00004      0              1e10
f41:p 9
sf41 10
ft41 tag 3
fu41 -1      00000.00001 00000.00003 00000.00004      0              1e10
f51:p 11
sf51 12
ft51 tag 3
fu51 -1      00000.00001 00000.00003 00000.00004      0              1e10
f61:p 13
sf61 14
ft61 tag 3
fu61 -1      00000.00001 00000.00003 00000.00004      0              1e10
f71:p 15
sf71 16
ft71 tag 3
fu71 -1      00000.00001 00000.00003 00000.00004      0              1e10
f81:p 17
sf81 18
ft81 tag 3
fu81 -1      00000.00001 00000.00003 00000.00004      0              1e10
f91:p 19
sf91 20
ft91 tag 3
fu91 -1      00000.00001 00000.00003 00000.00004      0              1e10
f101:p 21
sf101 22
ft101 tag 3
fu101 -1      00000.00001 00000.00003 00000.00004      0              1e10
f111:p 23
sf111 24
ft111 tag 3
fu111 -1      00000.00001 00000.00003 00000.00004      0              1e10
f121:p 25
sf121 26
ft121 tag 3
fu121 -1      00000.00001 00000.00003 00000.00004      0              1e10
f131:p 27
sf131 28
ft131 tag 3
fu131 -1      00000.00001 00000.00003 00000.00004      0              1e10
f141:p 29
sf141 30
ft141 tag 3
fu141 -1      00000.00001 00000.00003 00000.00004      0              1e10
f151:p 31
sf151 32
ft151 tag 3
fu151 -1      00000.00001 00000.00003 00000.00004      0              1e10
f161:p 33
sf161 34
ft161 tag 3
fu161 -1      00000.00001 00000.00003 00000.00004      0              1e10
f171:p 35
sf171 36
ft171 tag 3
fu171 -1      00000.00001 00000.00003 00000.00004      0              1e10
f181:p 37
```

```
sf181 38
ft181 tag 3
fu181 -1      00000.00001 00000.00003 00000.00004    0          1e10
f191:p 39
sf191 40
ft191 tag 3
fu191 -1      00000.00001 00000.00003 00000.00004    0          1e10
f201:p 41
sf201 42
ft201 tag 3
fu201 -1      00000.00001 00000.00003 00000.00004    0          1e10
f211:p 43
sf211 44
ft211 tag 3
fu211 -1      00000.00001 00000.00003 00000.00004    0          1e10
f221:p 45
sf221 46
ft221 tag 3
fu221 -1      00000.00001 00000.00003 00000.00004    0          1e10
f231:p 47
sf231 48
ft231 tag 3
fu231 -1      00000.00001 00000.00003 00000.00004    0          1e10
f241:p 49
sf241 50
ft241 tag 3
fu241 -1      00000.00001 00000.00003 00000.00004    0          1e10
f251:p 51
sf251 52
ft251 tag 3
fu251 -1      00000.00001 00000.00003 00000.00004    0          1e10
f261:p 53
sf261 54
ft261 tag 3
fu261 -1      00000.00001 00000.00003 00000.00004    0          1e10
f271:p 55
sf271 56
ft271 tag 3
fu271 -1      00000.00001 00000.00003 00000.00004    0          1e10
f281:p 57
sf281 58
ft281 tag 3
fu281 -1      00000.00001 00000.00003 00000.00004    0          1e10
f291:p 59
sf291 60
ft291 tag 3
fu291 -1      00000.00001 00000.00003 00000.00004    0          1e10
f301:p 61
sf301 62
ft301 tag 3
fu301 -1      00000.00001 00000.00003 00000.00004    0          1e10
f311:p 63
sf311 64
ft311 tag 3
fu311 -1      00000.00001 00000.00003 00000.00004    0          1e10
f321:p 65
sf321 66
ft321 tag 3
fu321 -1      00000.00001 00000.00003 00000.00004    0          1e10
f331:p 67
sf331 68
ft331 tag 3
fu331 -1      00000.00001 00000.00003 00000.00004    0          1e10
f341:p 69
sf341 70
ft341 tag 3
fu341 -1      00000.00001 00000.00003 00000.00004    0          1e10
f351:p 71
sf351 72
ft351 tag 3
fu351 -1      00000.00001 00000.00003 00000.00004    0          1e10
f361:p 73
sf361 74
ft361 tag 3
fu361 -1      00000.00001 00000.00003 00000.00004    0          1e10
f371:p 75
```

```
sf371 76
ft371 tag 3
fu371 -1      00000.00001 00000.00003 00000.00004    0          1e10
f381:p 77
sf381 78
ft381 tag 3
fu381 -1      00000.00001 00000.00003 00000.00004    0          1e10
f391:p 79
sf391 80
ft391 tag 3
fu391 -1      00000.00001 00000.00003 00000.00004    0          1e10
f401:p 81
sf401 82
ft401 tag 3
fu401 -1      00000.00001 00000.00003 00000.00004    0          1e10
f411:p 83
sf411 84
ft411 tag 3
fu411 -1      00000.00001 00000.00003 00000.00004    0          1e10
f421:p 85
sf421 86
ft421 tag 3
fu421 -1      00000.00001 00000.00003 00000.00004    0          1e10
f431:p 87
sf431 88
ft431 tag 3
fu431 -1      00000.00001 00000.00003 00000.00004    0          1e10
f441:p 89
sf441 90
ft441 tag 3
fu441 -1      00000.00001 00000.00003 00000.00004    0          1e10
f451:p 91
sf451 92
ft451 tag 3
fu451 -1      00000.00001 00000.00003 00000.00004    0          1e10
f461:p 93
sf461 94
ft461 tag 3
fu461 -1      00000.00001 00000.00003 00000.00004    0          1e10
f471:p 95
sf471 96
ft471 tag 3
fu471 -1      00000.00001 00000.00003 00000.00004    0          1e10
f481:p 97
sf481 98
ft481 tag 3
fu481 -1      00000.00001 00000.00003 00000.00004    0          1e10
f491:p 99
sf491 100
ft491 tag 3
fu491 -1      00000.00001 00000.00003 00000.00004    0          1e10
f501:p 101
sf501 102
ft501 tag 3
fu501 -1      00000.00001 00000.00003 00000.00004    0          1e10
f511:p 103
sf511 104
ft511 tag 3
fu511 -1      00000.00001 00000.00003 00000.00004    0          1e10
f521:p 105
sf521 106
ft521 tag 3
fu521 -1      00000.00001 00000.00003 00000.00004    0          1e10
f531:p 107
sf531 108
ft531 tag 3
fu531 -1      00000.00001 00000.00003 00000.00004    0          1e10
f541:p 109
sf541 110
ft541 tag 3
fu541 -1      00000.00001 00000.00003 00000.00004    0          1e10
f551:p 111
sf551 112
ft551 tag 3
fu551 -1      00000.00001 00000.00003 00000.00004    0          1e10
f561:p 113
```

```
sf561 114
ft561 tag 3
fu561 −1      00000.00001 00000.00003 00000.00004    0          1e10
f571:p 115
sf571 116
ft571 tag 3
fu571 −1      00000.00001 00000.00003 00000.00004    0          1e10
f581:p 117
sf581 118
ft581 tag 3
fu581 −1      00000.00001 00000.00003 00000.00004    0          1e10
f591:p 119
sf591 120
ft591 tag 3
fu591 −1      00000.00001 00000.00003 00000.00004    0          1e10
f601:p 121
sf601 122
ft601 tag 3
fu601 −1      00000.00001 00000.00003 00000.00004    0          1e10
f611:p 123
sf611 124
ft611 tag 3
fu611 −1      00000.00001 00000.00003 00000.00004    0          1e10
f621:p 125
sf621 126
ft621 tag 3
fu621 −1      00000.00001 00000.00003 00000.00004    0          1e10
f631:p 127
sf631 128
ft631 tag 3
fu631 −1      00000.00001 00000.00003 00000.00004    0          1e10
f641:p 129
sf641 130
ft641 tag 3
fu641 −1      00000.00001 00000.00003 00000.00004    0          1e10
f651:p 131
sf651 132
ft651 tag 3
fu651 −1      00000.00001 00000.00003 00000.00004    0          1e10
f661:p 133
sf661 134
ft661 tag 3
fu661 −1      00000.00001 00000.00003 00000.00004    0          1e10
f671:p 135
sf671 136
ft671 tag 3
fu671 −1      00000.00001 00000.00003 00000.00004    0          1e10
f681:p 137
sf681 138
ft681 tag 3
fu681 −1      00000.00001 00000.00003 00000.00004    0          1e10
f691:p 139
sf691 140
ft691 tag 3
fu691 −1      00000.00001 00000.00003 00000.00004    0          1e10
f701:p 141
sf701 142
ft701 tag 3
fu701 −1      00000.00001 00000.00003 00000.00004    0          1e10
f711:p 143
sf711 144
ft711 tag 3
fu711 −1      00000.00001 00000.00003 00000.00004    0          1e10
f721:p 145
sf721 146
ft721 tag 3
fu721 −1      00000.00001 00000.00003 00000.00004    0          1e10
f731:p 147
sf731 148
ft731 tag 3
fu731 −1      00000.00001 00000.00003 00000.00004    0          1e10
f741:p 149
sf741 150
ft741 tag 3
fu741 −1      00000.00001 00000.00003 00000.00004    0          1e10
f751:p 151
```

```
sf751 152
ft751 tag 3
fu751 −1     00000.00001 00000.00003 00000.00004     0           1e10
f761:p 153
sf761 154
ft761 tag 3
fu761 −1     00000.00001 00000.00003 00000.00004     0           1e10
f771:p 155
sf771 156
ft771 tag 3
fu771 −1     00000.00001 00000.00003 00000.00004     0           1e10
f781:p 157
sf781 158
ft781 tag 3
fu781 −1     00000.00001 00000.00003 00000.00004     0           1e10
f791:p 159
sf791 160
ft791 tag 3
fu791 −1     00000.00001 00000.00003 00000.00004     0           1e10
f801:p 161
sf801 162
ft801 tag 3
fu801 −1     00000.00001 00000.00003 00000.00004     0           1e10
f811:p 163
sf811 164
ft811 tag 3
fu811 −1     00000.00001 00000.00003 00000.00004     0           1e10
f821:p 165
sf821 166
ft821 tag 3
fu821 −1     00000.00001 00000.00003 00000.00004     0           1e10
f831:p 167
sf831 168
ft831 tag 3
fu831 −1     00000.00001 00000.00003 00000.00004     0           1e10
f841:p 169
sf841 170
ft841 tag 3
fu841 −1     00000.00001 00000.00003 00000.00004     0           1e10
f851:p 171
sf851 172
ft851 tag 3
fu851 −1     00000.00001 00000.00003 00000.00004     0           1e10
f861:p 173
sf861 174
ft861 tag 3
fu861 −1     00000.00001 00000.00003 00000.00004     0           1e10
f871:p 175
sf871 176
ft871 tag 3
fu871 −1     00000.00001 00000.00003 00000.00004     0           1e10
f881:p 177
sf881 178
ft881 tag 3
fu881 −1     00000.00001 00000.00003 00000.00004     0           1e10
f891:p 179
sf891 180
ft891 tag 3
fu891 −1     00000.00001 00000.00003 00000.00004     0           1e10
f901:p 181
sf901 182
ft901 tag 3
fu901 −1     00000.00001 00000.00003 00000.00004     0           1e10
f911:p 183
sf911 184
ft911 tag 3
fu911 −1     00000.00001 00000.00003 00000.00004     0           1e10
f921:p 185
sf921 186
ft921 tag 3
fu921 −1     00000.00001 00000.00003 00000.00004     0           1e10
f931:p 187
sf931 188
ft931 tag 3
fu931 −1     00000.00001 00000.00003 00000.00004     0           1e10
f941:p 189
```

```
sf941 190
ft941 tag 3
fu941 -1      00000.00001 00000.00003 00000.00004      0             1e10
f951:p 191
sf951 192
ft951 tag 3
fu951 -1      00000.00001 00000.00003 00000.00004      0             1e10
f961:p 193
sf961 194
ft961 tag 3
fu961 -1      00000.00001 00000.00003 00000.00004      0             1e10
f971:p 195
sf971 196
ft971 tag 3
fu971 -1      00000.00001 00000.00003 00000.00004      0             1e10
f981:p 197
sf981 198
ft981 tag 3
fu981 -1      00000.00001 00000.00003 00000.00004      0             1e10
f991:p 199
tf991 j j 1 j j j 2 j
ft991 tag 3
fu991 -1      00000.00001 00000.00003 00000.00004      0             1e10
e0
      1.0000000000e-03
      1.9765548681e-03
      2.9531097362e-03
      3.9296646044e-03
      4.9062194725e-03
      5.8827743406e-03
      6.8593292087e-03
      7.8358840768e-03
      8.8124389450e-03
      9.7889938131e-03
      1.0765548681e-02
m1
      82000 -1
prdmp j j 1 3 0
wwg 991 0 0 j j j j 0
wwge:p 0.001 0.02 0.03 0.05 0.1 0.3 0.5 1.0 3.0 10.0 30.0 50.0 100.1
mesh geom=sph ref=0 0 0 origin=0 0 0
      imesh=1.0000000000e-04 100i 4.6587463482e-02
      jmesh=0.5
      kmesh=1.0
ctme 480
imp:p 1
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     2.0M     R     2.0M     R     2.0M
     R     2.0M     R     2.0M     R     2.0M     R
     2.0M     R     0
dbcn j j j j j j j 10 j j j j 1529175
```

Listing A.3: Example Truck Trailer Input File

```
Standard 53-foot truck-trailer , 102-inches wide
c General Orientation:
c     x-coord is the width-dimension of the trailer
c     y-coord is the length-dimension of the trailer
c     z-coord is up/down
c     origin is set in the center floor of the cargo
c
c Cell Cards
c
c        Wood Floor
  101    5 -0.8      401 -402    205 -421    102 -101
c
c        Driver Side Wall
  102    3 -2.7      201 -202    205 -207    101 -209
c        Passenger Side Wall
  103    3 -2.7      204 -203    205 -207    101 -209
c        Rear Wall
  104    3 -2.7      202 -204    205 -206    101 -209
c        Front Wall
  105    3 -2.7      202 -204    208 -207    101 -209
c        Roof
  106    3 -2.7      202 -204    206 -208    210 -209
c        Inside cargo air space
  107    5 -0.2      247 -251    206 -208    101 -210
c
c
c
c        Inner Walls
c        Aluminum Hat
c        Left Lip
  110    3 -2.7      236 -237    232 -233    230 -231    u=110
c        Right Lip
  111    3 -2.7      236 -237    234 -235    230 -231    u=110
c        Left Jut
  112    3 -2.7      236 -239    233 -240    230 -231    u=110
c        Right Jut
  113    3 -2.7      236 -239    241 -234    230 -231    u=110
c        Top
  114    3 -2.7      239 -238    233 -234    230 -231
                     (-242:243:244:-245)                 u=110
c        Air Space above Left Lip
  115    0           237 -238    232 -233    230 -231    u=110
c        Air Space above Right Lip
  116    0           237 -238    234 -235    230 -231    u=110
c        Air Space under hat
  117    0           236 -239    240 -241    230 -231    u=110
c        Air Space in cutout
  118    0           239 -238    242 -243   -244 245     u=110
c        Everything Else
  119    0           -236:238:-232:235:-230:231          u=110
c        Containing Box
  120    0           236 -238    232 -235    230 -231    u=120 lat=1
                     fill=0:0 0:0 -24:0
                     110 110 110 110 110
                     110 110 110 110 110
                     110 110 110 110 110
                     110 110 110 110 110
                     110 110 110 110 110
c        Row of Vertical hats
  130    0           236 -238    232 -235    230 -210    u=130 fill=120
c        Everything Else
c  131    0           (-236:238:-232:235:-230:210)        u=130
c        Air Space
  135    0           236 -238    235 -246    230 -210    u=130
c        Everything Else
  136    0           (-236:238:-232:246:-230:210)        u=130
c        Containing Box
  140    0           236 -238    232 -246    230 -210    u=140 lat=1
                     fill=0:0 -40:0 0:0
                     130 130 130 130 130   130 130 130 130 130
                     130 130 130 130 130   130 130 130 130 130
                     130 130 130 130 130   130 130 130 130 130
                     130 130 130 130 130   130 130 130 130 130   130
c                     130 135   130 135   130 135   130 135   130 135
c                     130 135   130 135   130 135   130 135   130 135
```

```
c                   130 135  130 135  130 135  130 135  130 135
c                   130 135  130 135  130 135  130 135  130 135
c                   130 135  130 135  130 135  130 135  130 135
c                   130 135  130 135  130 135  130 135  130 135 130
c
c       Row of Rows
  141   0          236 -238    232 -208    230 -210     fill=140
c       Plywood
  150   5 -0.681   238 -247    248 -249    230 -210    u=150
c       Air Space to Left of Plywood
  155   0          238 -247    242 -248    230 -210    u=150
c       Everything Else
  156   0          (-238:247:-242:249:-230:210)         u=150
c       Containing Box
  160   0          238 -247    242 -249    230 -210    u=160 lat=1
                   fill=0:0 -40:0 0:0
                   150 150 150 150 150   150 150 150 150 150
                   150 150 150 150 150   150 150 150 150 150
                   150 150 150 150 150   150 150 150 150 150
                   150 150 150 150 150   150 150 150 150 150 150
c       Row of Plywood
  165   0          238 -247    242 -250    230 -210     fill=160
c       First Air Space
  166   0          238 -247    232 -242    230 -210
c       Last Air Space
  167   0          238 -247    250 -208    230 -210
c
c       Reflect plywood walls to passenger side
  170              like 141 but trcl=141
  171              like 165 but trcl=141
  172              like 166 but trcl=141
  173              like 167 but trcl=141
c
c
c
c
c
c
c
c BEGIN UNDERCARRIAGE
c
c       Lateral Cross "I" Beams
c           Create Master Cross Beam (very back piece)
c           Top Piece
  401   1 -7.87    401 -402 405 -406 410 -404 u=400
c           Bottom Piece
  402   1 -7.87    401 -402 405 -406 403 -409 u=400
c           Middle Piece
  403   1 -7.87    401 -402 407 -408 409 -410 u=400
c           Inner Air Space 1
  404   0          401 -402 405 -407 409 -410 u=400
c           Inner Air Space 2
  405   0          401 -402 408 -406 409 -410 u=400
c           Outer Air Space
  406   0          401 -402 406 -411 403 -404 u=400
c           Everything Else
  407   0           -401:402:-405:411:-403:404 u=400
c           Repeat Master 45 times in the y-direction
  410   0          401 -402 405 -411 403 -404 u=300  lat=1
                   fill=0:0 -44:0 0:0
                   400 400 400 400 400   400 400 400 400 400
                   400 400 400 400 400   400 400 400 400 400
                   400 400 400 400 400   400 400 400 400 400
                   400 400 400 400 400   400 400 400 400 400
                   400 400 400 400 400
c           Cross Beam Universe
  411   0          401 -402 405 -421 403 -404  fill=300
c
c
c       Transverse Cross "S" Beams
c           Passenger-Side Beam
c           Top Piece
  450   1 -7.87    452 -454 405 -463 460 -459
c           Bottom Piece
  451   1 -7.87    455 -450 405 -463 458 -461
c           Middle Piece
```

```
  452   1 -7.87     455 -454 405 -463 461 -460
c            Air Space 1
  454   0           454 -450 405 -463 461 -459
c            Air Space 2
  455   0           450 -111 405 -463 458 -459
c            Air space 3
  456   0           462 -452 405 -463 460 -459
c            Repeat for Driver-Side Beam, just flip over x-axis
  457               like 450 but trcl=450
  458               like 451 but trcl=450
  459               like 452 but trcl=450
  461               like 454 but trcl=450
  462               like 455 but trcl=450
  463               like 456 but trcl=450
c
c
c
c        Rear Bumper
c            Sheet
  470   1 -7.87     110 -111 470 -405 458 -101
c            Tube, top piece
  471   1 -7.87     110 -111 472 -470 474 -473
c            Tube, bottom piece
  472   1 -7.87     110 -111 472 -470 476 -475
c            Tube, back piece
  473   1 -7.87     110 -111 471 -472 476 -473
c            Tube, inner air space
  474   0           110 -111 472 -470 475 -474
c            Passenger Leg, left piece
  475   1 -7.87     452 -477 471 -470 479 -476
c            Passenger Leg, right piece
  476   1 -7.87     478 -450 471 -470 479 -476
c            Passenger leg, back piece
  477   1 -7.87     477 -478 471 -472 479 -476
c            Passenger leg, inner air space
  478   1 -7.87     477 -478 472 -470 479 -476
c            Passenger leg, air space left
  479   0           462 -452 471 -470 479 -476
c            Passenger leg, air space right
  480   0           450 -111 471 -470 479 -476
c            Translate passenger leg for drivers leg
  481               like 475 but trcl=(-119.38 0 0)
  482               like 476 but trcl=(-119.38 0 0)
  483               like 477 but trcl=(-119.38 0 0)
  484               like 478 but trcl=(-119.38 0 0)
  485               like 479 but trcl=(-54.61 0 0)
  486               like 480 but trcl=(-194.31 0 0)
c           Air above tube
  487   0           110 -111 471 -470 473 -101
c           Tube at bottom of legs
c           Top piece
  488   1 -7.87     483 -484   471 -405   480 -479
c           Bottom piece
  489   1 -7.87     483 -484   471 -405   482 -481
c           Back piece
  490   1 -7.87     483 -484   471 -472   481 -480
c           Front piece
  491   1 -7.87     483 -484   470 -405   481 -480
c           Air space inside
  492   0           483 -484   472 -470   481 -480
c           Air space on driver side of tube
  493   0           110 -483   471 -405   482 -479
c           Air space on passenger side of tube
  494   0           484 -111   471 -405   482 -479
c
c           Air space above tube and rear sheet
  495   0           110 -111   471 -205   101 -209
c           Air above inner edge of tube
  496   0           110 -111   470 -405   479 -458
c
c
c       Transverse extension from S-beams, passenger side
c       2-in square tube, outer
  520   1 -7.87      -485 486
c       2-in square tube, inner
  521   0             -486
```

```
c      extension sheet downward
  522   1 -7.87      -487
c      extension sheet inward
  523   1 -7.87      -488
c      air space between tubes
  524   0            -489
c      Reflect over x-axis for driver side
  525               like 520 but trcl=520
  526               like 521 but trcl=520
  527               like 522 but trcl=520
  528               like 523 but trcl=520
  529               like 524 but trcl=520
c
c      I-beams running across transverse extension
c      First air space
  540  0            -500
c      First I-beam, top piece
  541  1 -7.87      -501
c      middle piece
  542  1 -7.87      -502
c      bottom piece
  543  1 -7.87      -503
c      first air space
  544  0            -504
c      second air space
  545  0            -505
c      Repeat 3 times
  546              like 540 but trcl=(0 60.7568 0)
  547              like 541 but trcl=(0 60.7568 0)
  548              like 542 but trcl=(0 60.7568 0)
  549              like 543 but trcl=(0 60.7568 0)
  550              like 544 but trcl=(0 60.7568 0)
  551              like 545 but trcl=(0 60.7568 0)
c
  552              like 540 but trcl=(0 121.5136 0)
  553              like 541 but trcl=(0 121.5136 0)
  554              like 542 but trcl=(0 121.5136 0)
  555              like 543 but trcl=(0 121.5136 0)
  556              like 544 but trcl=(0 121.5136 0)
  557              like 545 but trcl=(0 121.5136 0)
c
  558              like 540 but trcl=(0 182.2704 0)
  559              like 541 but trcl=(0 182.2704 0)
  560              like 542 but trcl=(0 182.2704 0)
  561              like 543 but trcl=(0 182.2704 0)
  562              like 544 but trcl=(0 182.2704 0)
  563              like 545 but trcl=(0 182.2704 0)
c      One last final air space
  564              like 540 but trcl=(0 243.0272 0)
c      Sheet of air between inward extensions
  565  0            -506
c
c      Tires/Axles
c      Rear Axle
  570  1 -7.87      -514 515
c      Rear Axle air space
  571  0            -515
c      Front Axle
  572               like 570 but trcl=(0 149.352 0)
  573               like 571 but trcl=(0 149.352 0)
c      Rear Driver Rim
  574  3 -2.7       -510 511
c      Air space in rim
  575  0            -511
c      Tire
  576  7 -0.8       -512 513 510
c      Tire air space
  577  0            -513 510
c      Box of air around tire
  578  0            -516 512
c      Translate tire for other 7 tires
  579               like 574 but trcl=(33.448 0 0)
  580               like 575 but trcl=(33.448 0 0)
  581               like 576 but trcl=(33.448 0 0)
  582               like 577 but trcl=(33.448 0 0)
  583               like 578 but trcl=(33.448 0 0)
```

```
c
  584              like 574 but trcl=(230.632 0 0)
  585              like 575 but trcl=(230.632 0 0)
  586              like 576 but trcl=(230.632 0 0)
  587              like 577 but trcl=(230.632 0 0)
  588              like 578 but trcl=(230.632 0 0)
c
  589              like 574 but trcl=(197.184 0 0)
  590              like 575 but trcl=(197.184 0 0)
  591              like 576 but trcl=(197.184 0 0)
  592              like 577 but trcl=(197.184 0 0)
  593              like 578 but trcl=(197.184 0 0)
c
  594              like 574 but trcl=(0 149.352 0)
  595              like 575 but trcl=(0 149.352 0)
  596              like 576 but trcl=(0 149.352 0)
  597              like 577 but trcl=(0 149.352 0)
  598              like 578 but trcl=(0 149.352 0)
c
  599              like 574 but trcl=(33.448 149.352 0)
  600              like 575 but trcl=(33.448 149.352 0)
  601              like 576 but trcl=(33.448 149.352 0)
  602              like 577 but trcl=(33.448 149.352 0)
  603              like 578 but trcl=(33.448 149.352 0)
c
  604              like 574 but trcl=(230.632 149.352 0)
  605              like 575 but trcl=(230.632 149.352 0)
  606              like 576 but trcl=(230.632 149.352 0)
  607              like 577 but trcl=(230.632 149.352 0)
  608              like 578 but trcl=(230.632 149.352 0)
c
  609              like 574 but trcl=(197.184 149.352 0)
  610              like 575 but trcl=(197.184 149.352 0)
  611              like 576 but trcl=(197.184 149.352 0)
  612              like 577 but trcl=(197.184 149.352 0)
  613              like 578 but trcl=(197.184 149.352 0)
c
c    Air between tires
  614  0           -517
  615              like 614 but trcl=(197.184 0 0)
  616              like 614 but trcl=(0 149.352 0)
  617              like 614 but trcl=(197.184 149.352 0)
c    Air on inner side of tires
  618  0           -518 514.1
  619              like 618 but trcl=(127.0165 0 0)
  620              like 618 but trcl=(0 149.352 0)
  621              like 618 but trcl=(127.0165 149.352 0)
c    Air around axles between tires
  622  0           -519 514.1
  623              like 622 but trcl=(0 149.352 0)
c
c    Air behind rear tires under S-bars
  624  0           110 -111    405 -521    520 -458
c    Air between S-bars behind rear tires
  625  0           462 -455    405 -521    458 -460
  626              like 625 but trcl=(-59.3725 0 0)
c    Air under rear bumper
  627  0           110 -111    471 -405    520 -482
c    Air between S-bars in front of rear tires
  628  0           462 -455    522 -463    458 -460
  629              like 628 but trcl=(-59.3725 0 0)
c    Air under S-bars in front of rear tires
  630  0           110 -111    522 -463    520 -458
c
c
c    Kick-Stands
c    Passenger-side support
  650  1 -7.87     -540 541 -403
c    Support air space
  651  0           -541 -403
c    Tube, left side
  652  1 -7.87     550 -551    554 -557    558 540 -403
c    Tube, right side
  653  1 -7.87     552 -553    554 -557    558 540 -403
c    Tube, rear side
  654  1 -7.87     551 -552    554 -555    558 540 -403
```

```
c     Tube, rear side
  655  1 -7.87        551 -552     556 -557     558 540 -403
c     Tube, inner air space
  656  0              551 -552     555 -556     558 540 -403
c     Air space below tube
  657  0              550 -553     554 -557     520 -558
c     Air around support and tube
  658  0              462 -203     544 -545     520 540  -403
                      (-550:553:-554:557)
c     Driver Side tube, flip
  659                 like 650 but trcl=650
  660                 like 651 but trcl=650
  661                 like 652 but trcl=650
  662                 like 653 but trcl=650
  663                 like 654 but trcl=650
  664                 like 655 but trcl=650
  665                 like 656 but trcl=650
  666                 like 657 but trcl=650
  667                 like 658 but trcl=650
c
c     Air space under midsection of trailer
  680 0               110 -111     463 -544     520 -403
c     Air space under front of trailer
  681 0               110 -111     545 -608     520 -403
c
c     Steel supports alongside lateral I beams
c     Passenger side
  690 1  -7.87        402 -111     405 -420     403 -101
c     Driver side
  691 1  -7.87        110 -401     405 -420     403 -101
c     Front of trailer
  692 1  -7.87        401 -402     421 -420     403 -101
c
c
c
c END UNDERCARRIAGE
c
c
c
c BEGIN TRACTOR
c
c      Transverse midsection
  700 1  -1.22       610 -611     608 -600     613 -614
c      Air beneath midsection
  701 0              610 -611     608 -600     520 -613
c      Air above midsection underneath trailer
  702 0              610 -611     608 -607     614 -403
c      Air on left side of midsection, front axle
  703 0              201 -610     622 -607     520 -403
                     630 631
c      Air on left side of midsection, rear axle
  704 0              201 -610     608 -622     520 -403
                     632 633
c      Air on right side of midsection, front axle
  705 0              611 -203     622 -607     520 -403
                     634 635
c      Air on right side of midsection, rear axle
  706 0              611 -203     608 -622     520 -403
                     636 637
c      Left front outer tire
  707                like 574 but trcl=(0 1370.592 0)
  708                like 575 but trcl=(0 1370.592 0)
  709                like 576 but trcl=(0 1370.592 0)
  710                like 577 but trcl=(0 1370.592 0)
c      Left front inner tire
  711                like 574 but trcl=(33.448 1370.592 0)
  712                like 575 but trcl=(33.448 1370.592 0)
  713                like 576 but trcl=(33.448 1370.592 0)
  714                like 577 but trcl=(33.448 1370.592 0)
c      Left rear outer tire
  715                like 574 but trcl=(0 1230.384 0)
  716                like 575 but trcl=(0 1230.384 0)
  717                like 576 but trcl=(0 1230.384 0)
  718                like 577 but trcl=(0 1230.384 0)
c      Left rear inner tire
  719                like 574 but trcl=(33.448 1230.384 0)
```

```
  720              like 575 but trcl=(33.448 1230.384 0)
  721              like 576 but trcl=(33.448 1230.384 0)
  722              like 577 but trcl=(33.448 1230.384 0)
c     Right front outer tire
  723              like 574 but trcl=(230.632 1370.592 0)
  724              like 575 but trcl=(230.632 1370.592 0)
  725              like 576 but trcl=(230.632 1370.592 0)
  726              like 577 but trcl=(230.632 1370.592 0)
c     Right front inner tire
  727              like 574 but trcl=(197.184 1370.592 0)
  728              like 575 but trcl=(197.184 1370.592 0)
  729              like 576 but trcl=(197.184 1370.592 0)
  730              like 577 but trcl=(197.184 1370.592 0)
c     Right rear outer tire
  731              like 574 but trcl=(230.632 1230.384 0)
  732              like 575 but trcl=(230.632 1230.384 0)
  733              like 576 but trcl=(230.632 1230.384 0)
  734              like 577 but trcl=(230.632 1230.384 0)
c     Right rear inner tire
  735              like 574 but trcl=(197.184 1230.384 0)
  736              like 575 but trcl=(197.184 1230.384 0)
  737              like 576 but trcl=(197.184 1230.384 0)
  738              like 577 but trcl=(197.184 1230.384 0)
c
c    Air space in front of trailer and behind cab
  740 0            201 -203    607 -606    614 -209
c    Cab
  741 1 -1.22      201 -203    606 -603    614 -621
                   (-609:612:-605:604:-615:620)
c    Inside Cab
  742 0            609 -612    605 -604    615 -620
c    Air above cab
  743 0            201 -203    606 -603    621 -209
c    Engine bay, hood
  744 1 -1.22      201 -203    603 -600    618 -619
c    Engine bay, front side, top of T
  745 1 -1.22      609 -612    601 -600    616 -618
c    Engine bay, front side, bottom of T
  746 1 -1.22      625 -628    601 -600    614 -616
c    Engine bay, left outer wall
  747 1 -1.22      201 -609    602 -600    616 -618
c    Engine bay, right outer wall
  748 1 -1.22      612 -203    602 -600    616 -618
c    Engine bay, left top of wheel well
  749 1 -1.22      609 -625    602 -601    616 -617
c    Engine bay, right top of wheel well
  750 1 -1.22      628 -612    602 -601    616 -617
c    Engine bay, left inner wheel well
  751 1 -1.22      625 -626    602 -601    614 -617
c    Engine bay, right inner wheel well
  752 1 -1.22      627 -628    602 -601    614 -617
c    Engine bay, back side, top of T
  753 1 -1.22      201 -203    603 -602    616 -618
c    Engine bay, back side, bottom of T
  754 1 -1.22      625 -628    603 -602    614 -616
c    Engine, 1000 lbs, fills most of engine bay, made of steel, lowered rho
  755 1 -0.145     609 -612    602 -601    617 -618
c    Engine bay, air inside bottom of T
  756 0            626 -627    602 -601    614 -617
c    Air above engine bay
  757 0            201 -203    603 -600    619 -209
c
c    Left front bumper, front side
  760 1 -1.22      201 -610    601 -600    613 -616 (-614:-625)
c    Left front bumper, left side
  761 1 -1.22      201 -609    623 -601    613 -616
c    Right front bumper, front side
  762 1 -1.22      611 -203    601 -600    613 -616 (-614:628)
c    Right front bumper, right side
  763 1 -1.22      612 -203    623 -601    613 -616
c    Air inside left front bumper
  771 0            609 -610    623 -601    613 -616 (-614:-625)
c    Air inside right front bumper
  773 0            611 -612    623 -601    613 -616 (-614:628)
c    Left side midsection shell
  774 1 -1.22      201 -610    607 -603    613 -614
```

```
                   (-609:614)
c     Air inside left midsection shell
  775 0              609 -610    607 -603    613 -614   650
c     Right side midsection shell
  776 1 -1.22        611 -203    607 -603    613 -614
                     (612:614)
c     Air inside right midsection shell
  777 0              611 -612    607 -603    613 -614   651
c     Air underneath left midsection shell
  778 0              201 -610    607 -603    520 -613
c     Air underneath right midsection shell
  779 0              611 -203    607 -603    520 -613
c     Air underneath left front bumper
  780 0              201 -610    623 -600    520 -613
c     Air underneath right front bumper
  781 0              611 -203    623 -600    520 -613
c     Air in left tire well
  782 0              201 -625    603 -623    520 -616
                     638
c     Air in right tire well
  783 0              628 -203     603 -623    520 -616
                     639
c     Midsection extension to left wheel well
  784 1 -1.22        625 -610    603 -623    613 -614
c     Midsection extension to right wheel well
  785 1 -1.22        611 -628    603 -623    613 -614
c     Air under left midsection extension
  786 0              625 -610    603 -623    520 -613
c     Air under right midsection extension
  787 0              611 -628    603 -623    520 -613
c
c      Left front tire
  788                like 574 but trcl=(0 1844.044 0)
  789                like 575 but trcl=(0 1844.044 0)
  790                like 576 but trcl=(0 1844.044 0)
  791                like 577 but trcl=(0 1844.044 0)
c      Right front tire
  792                like 574 but trcl=(230.632 1844.044 0)
  793                like 575 but trcl=(230.632 1844.044 0)
  794                like 576 but trcl=(230.632 1844.044 0)
  795                like 577 but trcl=(230.632 1844.044 0)
c
c      Left gas tank (150 gal of diesel)
  800 8 -0.95        -650
c      Right gas tank (150 gal of diesel)
  801 8 -0.95        -651
c
c
c
c
c Dirt
1000 4 -1.0 -2000 -520
c Air around truck (void for now)
2000 2 -0.0012 -2000 520 (-110:3311:4107:-3000:3020)
c
c
c
c Rest of World
2001 0 2000
c
c
c
c
c
c detectors
c
3001 0 3300 -3301 3200 -3201 3000 -3020
3002 0 3301 -3311 3200 -3201 3000 -3020
3003 0 3300 -3302 3201 -3202 3000 -3020
3004 0 3302 -3303 3201 -3202 3000 -3020
3005 0 3303 -3311 3201 -3202 3000 -3020
3006 0 3300 -3304 3202 -3203 3000 -3020
3007 0 3304 -3305 3202 -3203 3000 -3020
3008 0 3305 -3311 3202 -3203 3000 -3020
3009 0 3300 -3306 3203 -3204 3000 -3020
3010 0 3306 -3307 3203 -3204 3000 -3020
```

```
3011 0 3307 -3311 3203 -3204 3000 -3020
3012 0 3300 -3308 3204 -3205 3000 -3020
3013 0 3308 -3309 3204 -3205 3000 -3020
3014 0 3309 -3311 3204 -3205 3000 -3020
3015 0 3300 -3310 3205 -3206 3000 -3020
3016 0 3310 -3311 3205 -3206 3000 -3020
3501 0 4000 -4001 4100 -4101 3000 -3020
3502 0 4000 -4001 4101 -4107 3000 -3020
3503 0 4001 -4002 4100 -4102 3000 -3020
3504 0 4001 -4002 4102 -4103 3000 -3020
3505 0 4001 -4002 4103 -4107 3000 -3020
3506 0 4002 -4003 4100 -4104 3000 -3020
3507 0 4002 -4003 4104 -4105 3000 -3020
3508 0 4002 -4003 4105 -4107 3000 -3020
3509 0 4003 -4004 4100 -4106 3000 -3020
3510 0 4003 -4004 4106 -4107 3000 -3020
c
c space between side detectors and side of truck
4000 0   203 -3300   3000 -3020   3200 -3206
c space behind truck
4001 0   201 -203    3000 -471    3200 -209
c space in front of truck
4002 0   201 -203    600 -3020    3200 -209
c space above truck between top of truck and top detectors
4003 0   201 -203    3000 -3020   209 -4100
c space on driver side of detectors above truck (weird i know)
4004 0   201 -4000   3000 -3020   4100 -4107
c corner between detector arrays (weird i know)
4005 0   4004 -3311  3000 -3020   4100 -4107
c
c
c

c Surface Cards
c
c
c
c BEGIN TOP OF TRAILER
c     Wood floor top
  101  pz 0.0
c     Wood floor bottom
  102  pz -3.175
c
c     Aluminum Walls are 1.27mm thick, cargo is 102 inches high (259.08)
c     Driver Side Wall
c     Outer side
  201  px -129.54
c     Inner side
  202  px -129.413
c     Passenger Side Wall
c     Outer side
  203  px  129.54
c     Inner side
  204  px  129.413
c     Rear Wall
c     Outer Side
  205  py -807.72
c     Inner Side
  206  py -807.593
c     Front Wall
c     Outer Side
  207  py  807.72
c     Inner Side
  208  py  807.593
c     Roof
c     Outer Side
  209  pz  259.08
c     Inner Side
  210  pz  258.953
c
c     Walls
c     Aluminum Hats (start with driver-side far rear hat)
c
c        top
c        ----  jut
c     --|    |--
```

```
c      lip      lip
c
c      Bottom of Hat
  230  pz 0.0
c      Hat Height
  231  pz 10.35812
c      Left Lip, left side
  232  py -807.593
c      Left Lip, right side
  233  py -805.593
c      Right Lip, left side
  234  py -799.593
c      Right Lip, right side
  235  py -797.593
c      Lip inner surface
  236  px -129.413
c      Lip outer surface (top)
  237  px -129.254
c      Top outer surface
  238  px -124.413
c      Top inner surface
  239  px -124.572
c      Left Jut inner surface
  240  py -805.434
c      Right Jut inner surface
  241  py -799.752
c      Hole left side
  242  py -804.093
c      Hole right side
  243  py -801.093
c      Hole top side
  244  pz 7.67276
c      Hoel bottom side
  245  pz 2.67276
c      Air Spacing Right boundary
  246  py -767.46335
c      Plywood outer surface
  247  px  -123.778
c      Plywood Left Side
  248  py  -801.093
c      Plywood right side
  249  py  -763.96335
c      End of plywood series
  250  py  804.093
c      Passenger Side plywood boundary
  251  px 123.778
c
c
c
c
c END TOP OF TRAILER
c
c
c  GENERAL
c
c  Driver Side
  110  px -129.54
c  Passenger Side
  111  px 129.54
c
c
c
c
c
c BEGIN UNDERCARRIAGE
c
c      Lateral Cross Beams
c          These are 4-inches (10.16 cm) tall steel I-beams running width-wise across
c          the entire length of the trailer, spaced ~12-inches (31.507727273) cm apart
c          ( or step lengths of 36.587727273 cm ), and areade of 0.4 cm-thick steel
c      Width-wise outer edges (129.04 cm)
  401  px -129.04
  402  px 129.04
c      Bottom and top of beams
  403  pz -13.335
  404  pz -3.175     $ coincident w/ 102
```

```
c      Outer length-wise "I" walls at back of trailer
  405  py -807.72
  406  py -802.64
c      Inner length-wise "I" walls at back of trailer
  407  py -805.38
  408  py -804.98
c      Inner height-wise "I" walls
  409  pz -12.935  $ bottom
  410  pz -3.575   $ top
c      Beginning of next I beam
  411  py -771.132272727
c      Front of trailer
  420  py 807.72
c      End of I beams at front of trailer
  421  py 807.22
c
c      Transverse Cross Beams
c          These are two S-shaped beams 0.635 cm thick steel, with the tops
c          of the "S"s pointing inward
c      Outer width-wise S
  450  px 64.77
c      Inner width-wise S
  452  px 54.61
c      Middle width-wise S
  454  px 60.0075
  455  px 59.3725
c      Bottom and top
  458  pz -23.495
  459  pz -13.335
c      Inner bottom and top
  460  pz -13.97
  461  pz -22.86
c      Middle of Container
  462  px 0.0
c      End of S-beams (approximated)
  463  py -309.88
c
c
c      Rear Bumper
c          The rear bumper is a 3.8 mm sheet, with a 2 inch, 3.8 mm thick
c          square tube, with an extension toward the ground
c      Back of sheet
  470  py -808.1
c      Back of tube
  471  py -812.8
c      Inner back of tube
  472  py -812.42
c      Outer Top of tube
  473  pz -8.89
c      Inner Top of tube
  474  pz -9.27
c      Inner Bottom of tube
  475  pz -13.59
c      Outer Bottom of tube
  476  pz -13.97
c      Legs down to extension are 4 inches wide, two inches deep
c      Inner leg wall, left
  477  px 55.09
c      Inner leg wall, right
  478  px 64.29
c      Bottom of legs
  479  pz -46.99
c      Bottom tube, inner top
  480  pz -47.47
c      Bottom tube, inner bottom
  481  pz -51.59
c      Bottom tube, Outer bottom
  482  pz -52.07
c      Bottom tube, driver side boundary
  483  px -74.63
c      Bottom tube, passenger side boundary
  484  px 74.63
c
c
c      Rear Suspension
c      Transverse extension from S-beams 8mm thick
```

```
c     2-in square tube, outer
  485  rpp    54.2925 59.3725    -708.152  -409.448    -19.05  -13.97
c     2-in square tube, inner
  486  rpp    55.0925 58.5725    -708.152  -409.448    -18.25  -14.77
c     extension sheet downward
  487  rpp    58.5725 59.3725    -708.152  -409.448    -34.29  -19.05
c     extension sheet inward
  488  rpp    48.4125 58.5725    -708.152  -409.448    -34.29  -33.49
c     air space between tubes
  489  rpp    0 54.2925          -708.152  -409.448    -19.05  -13.97
c
c     Lateral I-beams which lie across transverse extension 6.35 mm thick
c     spacing between them and on edges is 55.6768 cm
c     First air spacing (from back)
  500  rpp    -58.5725 58.5725   -708.152  -652.4752    -33.49  -19.05
c     First I beam (from back), top piece
  501  rpp    -58.5725 58.5725   -652.4752 -647.3952   -19.685  -19.05
c     middle piece
  502  rpp    -58.5725 58.5725   -650.2527 -649.6177   -32.855  -19.685
c     bottom piece
  503  rpp    -58.5725 58.5725   -652.4752 -647.3952   -33.49  -32.855
c     air space 1
  504  rpp    -58.5725 58.5725   -652.4752 -650.2527   -32.855  -19.685
c     air space 2
  505  rpp    -58.5725 58.5725   -649.6177 -647.3952   -32.855  -19.685
c     sheet of air between inward extensions
  506  rpp    -48.4125 48.4125  -708.152  -409.448  -34.29  -33.49
c
c     Wheels outer diameter 40.8 inches (51.816 cm radius)
c     width 11.2 inches (28.448 cm), thickness of 1 cm, with aluminum rims
c     Aluminum Rim,  1.3 cm thick, 20.955 cm radius
  510  rcc    -129.54  -633.476  -80.391     28.448 0 0    20.955
c     Aluminum Rim inner air space
  511  rcc    -128.24  -633.476  -80.391     25.848 0 0    19.655
c     Tire
  512  rcc    -129.54  -633.476  -80.391     28.448 0 0    51.816
c     Tire air space
  513  rcc    -128.54  -633.476  -80.391     26.448 0 0    50.816
c     Axles(2)
c     Rear Axle   1.356 cm thick, 4 inch (10.16 cm) diameter
  514  rcc    -67.644  -633.476  -80.391    135.288 0 0    5.08
c     Axle inner air space
  515  rcc    -67.644  -633.476  -80.391    135.288 0 0    3.724
c     Box around tires to make geometry work faster
  516  rpp    -129.54  -101.092      -708.152  -558.8   -132.207  -23.495
c     Air between tires
  517  rpp    -101.092  -96.092      -708.152  -558.8   -132.207  -23.495
c     Air on inside (width-wise) of tires
  518  rpp    -67.644  -59.3725      -708.152  -558.8   -132.207  -23.495
c     Air around axles between tires, rear
  519  rpp    -59.3725 59.3725       -708.152  -558.8   -132.207  -34.29
c     Ground Floor
  520  pz     -132.207
c     Rear of extension
  521  py     -708.152
c     Front of extension
  522  py     -409.448
c
c
c     Kick stands, approximated as 4-inch, 0.8-cm thick square tubes
c     Joint, modeled as hemispheres 10 cm radius
c     Outer sphere
  540  s      119.04 515.62  -13.335    10.0
c     Inner sphere
  541  s      119.04 515.62  -13.335    9.2
c     x-y bounds around sphere
  542  px     109.04
  543  px     129.04
  544  py     505.62
  545  py     525.62
c     Square tube
c     left side outer
  550  px     113.96
c     left side inner
  551  px     114.76
c     right side inner
```

```
  552  px     123.32
c      right side outer
  553  px     124.12
c      rear side outer
  554  py     510.54
c      rear side inner
  555  py     511.34
c      front side inner
  556  py     519.9
c      front side outer
  557  py     520.7
c      bottom of stands
  558  pz     -70
c
c
c END UNDERCARRIAGE
c
c
c
c
c
c BEGIN TRACTOR
c      Front of tractor, engine bay
  600  py   1338.072
c      Inside front of engine bay
  601  py   1337.572
c      Inside back of engine bay
  602  py   1146.048
c      Outside front of cab
  603  py   1145.548
c      Inside front of cab
  604  py   1145.048
c      Inside back of cab
  605  py   905.256
c      Outside back of cab
  606  py   904.756
c      Divide between frontal/back section and front of trailer
  607  py   807.72
c      Rear of tractor
  608  py   526.804
c
c      Left inner wall
  609  px   -129.04
c      Left side of transverse midsection
  610  px   -43.2816
c      Right side of transverse midsection
  611  px   43.2816
c      Right inner wall
  612  px   129.04
c
c      Bottom of tractor
  613  pz   -104.651
c      Top of rear tractor
  614  pz   -25.151
c      Inside bottom of cab
  615  pz   -24.651
c      Top of front wheel well
  616  pz   -1.143
c      Inside bottom of engine bay
  617  pz   -0.643
c      Inside top of engine bay
  618  pz   61.029
c      Outside top of engine bay
  619  pz   61.529
c      Inside top of cab
  620  pz   162.949
c      Outside top of cab
  621  pz   163.449
c
c
c      Divider for rear axles
  622  py   667.012
c      Rear of front bumper
  623  py   1275.588
c      Top inside of bottom of tractor
  624  pz   -108.832
```

```
c      Left wheel well, inner right side
  625  px    -86
c      Left wheel well, outer right side
  626  px    -85
c      Right wheel well, outer left side
  627  px    85
c      Right wheel well, inner left side
  628  px    86
c      Inner midsection shell top
  629  pz    -65.651
c
c      Back of Tractor
c      Left front outer tire
  630  rcc    -129.54 737.116 -80.391     28.448 0 0    51.816
c      Left front inner tire
  631  rcc    -96.092 737.116 -80.391     28.448 0 0    51.816
c      Left rear outer tire
  632  rcc    -129.54 596.908 -80.391     28.448 0 0    51.816
c      Left rear inner tire
  633  rcc    -96.092 596.908 -80.391     28.448 0 0    51.816
c      Right front outer tire
  634  rcc    101.092 737.116 -80.391     28.448 0 0    51.816
c      Right front inner tire
  635  rcc    67.644 737.116 -80.391     28.448 0 0    51.816
c      Right rear outer tire
  636  rcc    101.092 596.908 -80.391     28.448 0 0    51.816
c      Right rear outer tire
  637  rcc    67.644 596.908 -80.391     28.448 0 0    51.816
c
c      Front tractor
c      Left front outer tire
  638  rcc    -129.54 1210.568 -80.391     28.448 0 0    51.816
c      Right front outer tire
  639  rcc    101.092 1210.568 -80.391     28.448 0 0    51.816
c
c      Left fuel tank
  650  rcc    -86.1608 910 -64.651     0 113 0      38
c      Right fuel tank
  651  rcc    86.1608 910 -64.651     0 113 0      38
c
c
c universe cylinder
2000 rcc 0 0 -232 0 0 700 4000
c
c
c
c detectors
3000 py -9.1500960000e+02
3001 py -7.9699104000e+02
3002 py -6.7897248000e+02
3003 py -5.6095392000e+02
3004 py -4.4293536000e+02
3005 py -3.2491680000e+02
3006 py -2.0689824000e+02
3007 py -8.8879680000e+01
3008 py 2.9138880000e+01
3009 py 1.4715744000e+02
3010 py 2.6517600000e+02
3011 py 3.8319456000e+02
3012 py 5.0121312000e+02
3013 py 6.1923168000e+02
3014 py 7.3725024000e+02
3015 py 8.5526880000e+02
3016 py 9.7328736000e+02
3017 py 1.0913059200e+03
3018 py 1.2093244800e+03
3019 py 1.3273430400e+03
3020 py 1.4453616000e+03
3200 pz -1.3220700000e+02
3201 pz -5.5565833333e+01
3202 pz 2.1075333333e+01
3203 pz 9.7716500000e+01
3204 pz 1.7435766667e+02
3205 pz 2.5099883333e+02
3206 pz 3.2764000000e+02
3300 px 1.9810000000e+02
```

```
3301  px  1.9910000000e+02
3302  px  1.9810000010e+02
3303  px  1.9910000010e+02
3304  px  1.9810000020e+02
3305  px  1.9910000020e+02
3306  px  1.9810000030e+02
3307  px  1.9910000030e+02
3308  px  1.9810000040e+02
3309  px  1.9910000040e+02
3310  px  1.9810000050e+02
3311  px  1.9910000050e+02
4000  px  -2.8300000000e+01
4001  px  2.8300000000e+01
4002  px  8.4900000000e+01
4003  px  1.4150000000e+02
4004  px  1.9810000000e+02
4100  pz  3.2764000000e+02
4101  pz  3.2864000000e+02
4102  pz  3.2764000010e+02
4103  pz  3.2864000010e+02
4104  pz  3.2764000020e+02
4105  pz  3.2864000020e+02
4106  pz  3.2764000030e+02
4107  pz  3.2864000030e+02

*tr450  0  -1117.6  0   180  90  90   90  180  90   90  90  0
*tr520  0  -1117.6  0   180  90  90   90  180  90   90  90  0
*tr650  0  1031.24  0   180  90  90   90  180  90   90  90  0
*tr141  0  0  0   180  90  90   90  180  90   90  90  0
imp:p      1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1  0
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1   1   1   1     1   1   1   1   1
           1   1
c           1   1   1   1   1     1   1   1   1   1
c           1   1   1   1   1
mode p
c Materials
c Structural_Steel
m1         14000  -1.012e-02 $ silicon-natural
           24000  -1.693e-01 $ chromium-natural
           25000  -1.996e-02 $ manganese-natural
           26000  -6.550e-01 $ iron-natural
           28000  -1.204e-01 $ nickel-natural
           42000  -2.514e-02 $ molybdenum-natural
       cond=0
       gas=0
```

```
c Air_STP
m2         7000  -7.808e-01 $ nitrogen-natural
           8000  -2.095e-01 $ oxygen-natural
          18000  -9.340e-03 $ argon-natural
      cond=0
      gas=1
c Aluminum
m3        13027  -1.000e+00 $ aluminum-27
      cond=0
      gas=0
c Average_US_Soil
m4         1001  -2.810e-02 $ hydrogen-1
           6000  -1.443e-01 $ carbon-natural
           7000  -1.000e-05 $ nitrogen-natural
           8000  -4.964e-01 $ oxygen-natural
          11000  -8.200e-03 $ sodium-natural
          13000  -8.930e-02 $ aluminum-natural
          14000  -2.132e-01 $ silicon-natural
          19000  -5.600e-03 $ potassium-natural
          20000  -5.400e-03 $ calcium-natural
          26000  -9.600e-03 $ iron-natural
      cond=0
      gas=0
c Wood
m5         1001  -5.789e-02 $ hydrogen-1
           6000  -4.800e-01 $ carbon-natural
           8000  -4.600e-01 $ oxygen-natural
      cond=0
      gas=0
c Butyl rubber
m7         1001  -1.437e-01 $ hydrogen-1
           6012  -8.563e-01 $ carbon-12
      cond=0
      gas=0
c Diesel Fuel
m8         1001   0.2299655 $ h-1
           1002   0.0000345 $ h-2
           6012   0.11868   $ c-12
           6013   0.00132   $ c-13
           cond=0
           gas=0
c prdmp j j 1
prdmp j j 1 3 0
f1:p 3300
ft1 scx 1
fs1
      -3001
      -3002
      -3003
      -3004
      -3005
      -3006
      -3007
      -3008
      -3009
      -3010
      -3011
      -3012
      -3013
      -3014
      -3015
      -3016
      -3017
      -3018
      -3019
      -3020
f11:p 3302
ft11 scx 1
fs11
      -3001
      -3002
      -3003
      -3004
      -3005
      -3006
      -3007
```

```
        -3008
        -3009
        -3010
        -3011
        -3012
        -3013
        -3014
        -3015
        -3016
        -3017
        -3018
        -3019
        -3020
f21:p 3304
ft21 scx 1
fs21
        -3001
        -3002
        -3003
        -3004
        -3005
        -3006
        -3007
        -3008
        -3009
        -3010
        -3011
        -3012
        -3013
        -3014
        -3015
        -3016
        -3017
        -3018
        -3019
        -3020
f31:p 3306
ft31 scx 1
fs31
        -3001
        -3002
        -3003
        -3004
        -3005
        -3006
        -3007
        -3008
        -3009
        -3010
        -3011
        -3012
        -3013
        -3014
        -3015
        -3016
        -3017
        -3018
        -3019
        -3020
f41:p 3308
ft41 scx 1
fs41
        -3001
        -3002
        -3003
        -3004
        -3005
        -3006
        -3007
        -3008
        -3009
        -3010
        -3011
        -3012
        -3013
        -3014
```

```
        −3015
        −3016
        −3017
        −3018
        −3019
        −3020
f51:p 3310
ft51 scx 1
fs51
        −3001
        −3002
        −3003
        −3004
        −3005
        −3006
        −3007
        −3008
        −3009
        −3010
        −3011
        −3012
        −3013
        −3014
        −3015
        −3016
        −3017
        −3018
        −3019
        −3020
f61:p 4100
ft61 scx 1
fs61
        −3001
        −3002
        −3003
        −3004
        −3005
        −3006
        −3007
        −3008
        −3009
        −3010
        −3011
        −3012
        −3013
        −3014
        −3015
        −3016
        −3017
        −3018
        −3019
        −3020
f71:p 4102
ft71 scx 1
fs71
        −3001
        −3002
        −3003
        −3004
        −3005
        −3006
        −3007
        −3008
        −3009
        −3010
        −3011
        −3012
        −3013
        −3014
        −3015
        −3016
        −3017
        −3018
        −3019
        −3020
f81:p 4104
```

```
ft81 scx 1
fs81
       -3001
       -3002
       -3003
       -3004
       -3005
       -3006
       -3007
       -3008
       -3009
       -3010
       -3011
       -3012
       -3013
       -3014
       -3015
       -3016
       -3017
       -3018
       -3019
       -3020
f91:p 4106
ft91 scx 1
fs91
       -3001
       -3002
       -3003
       -3004
       -3005
       -3006
       -3007
       -3008
       -3009
       -3010
       -3011
       -3012
       -3013
       -3014
       -3015
       -3016
       -3017
       -3018
       -3019
       -3020
f991:p (3300 4100)
e0
       1.0000000000e-03
       1.9765548681e-03
       2.9531097362e-03
       3.9296646044e-03
       4.9062194725e-03
       5.8827743406e-03
       6.8593292087e-03
       7.8358840768e-03
       8.8124389450e-03
       9.7889938131e-03
       1.0765548681e-02
nps 1e9
wwg 991 0 0 j j j j 0
wwge:p 0.001 0.02 0.03 0.05 0.1 0.3 0.5 1.0 3.0 10.0 30.0 50.0 100.1
mesh geom=rec ref=0 0 0.1 origin=-2000.1 -2000.1 -2000.1
       imesh=-130 200 2000.1
       iints=1 10 1
       jmesh=-920 1500 2000.1
       jints=1 80 1
       kmesh=-140 330 2000.1
       kints=1 15 1
sdef par=p pos=d1 erg=0.01
si1 L
       -123 -619.963 0.1
       -123 -619.963 129.05
       -123 -619.963 258
       -123 -383.926 0.1
       -123 -383.926 129.05
       -123 -383.926 258
```

```
        −123  −147.889  0.1
        −123  −147.889  129.05
        −123  −147.889  258
        −123  88.1482  0.1
        −123  88.1482  129.05
        −123  88.1482  258
        −123  324.185  0.1
        −123  324.185  129.05
        −123  324.185  258
        −123  560.222  0.1
        −123  560.222  129.05
        −123  560.222  258
        −123  796.26  0.1
        −123  796.26  129.05
        −123  796.26  258
        0  −619.963  0.1
        0  −619.963  129.05
        0  −619.963  258
        0  −383.926  0.1
        0  −383.926  129.05
        0  −383.926  258
        0  −147.889  0.1
        0  −147.889  129.05
        0  −147.889  258
        0  88.1482  0.1
        0  88.1482  129.05
        0  88.1482  258
        0  324.185  0.1
        0  324.185  129.05
        0  324.185  258
        0  560.222  0.1
        0  560.222  129.05
        0  560.222  258
        0  796.26  0.1
        0  796.26  129.05
        0  796.26  258
        123  −619.963  0.1
        123  −619.963  129.05
        123  −619.963  258
        123  −383.926  0.1
        123  −383.926  129.05
        123  −383.926  258
        123  −147.889  0.1
        123  −147.889  129.05
        123  −147.889  258
        123  88.1482  0.1
        123  88.1482  129.05
        123  88.1482  258
        123  324.185  0.1
        123  324.185  129.05
        123  324.185  258
        123  560.222  0.1
        123  560.222  129.05
        123  560.222  258
        123  796.26  0.1
        123  796.26  129.05
        123  796.26  258
sp1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1
```

Listing A.4: Example Detector Input File

```
hpge 0
c
250  0  7  −999  101  −107  201  −207  imp:p=0
1  1  −5.3   1  −2  101  −102  201  −202  trcl=(0.0  0.0  0.0)  imp:p=1
251  0  1  −7  101  −107  201  −207  #1  trcl=(0.0  0.0  0.0)  imp:p=0
```

```
c
254 like 250 but trcl=(201.0 0.0 0.0)
2 1 -5.3  1 -2 101 -102 201 -203 trcl=(201.0 0.0 0.0) imp:p=1
253 0 1 -7 101 -107 201 -207 #2 trcl=(201.0 0.0 0.0) imp:p=0
c
256 like 250 but trcl=(402.0 0.0 0.0)
3 1 -5.3  1 -2 101 -102 201 -204 trcl=(402.0 0.0 0.0) imp:p=1
255 0 1 -7 101 -107 201 -207 #3 trcl=(402.0 0.0 0.0) imp:p=0
c
258 like 250 but trcl=(603.0 0.0 0.0)
4 1 -5.3  1 -2 101 -102 201 -205 trcl=(603.0 0.0 0.0) imp:p=1
257 0 1 -7 101 -107 201 -207 #4 trcl=(603.0 0.0 0.0) imp:p=0
c
260 like 250 but trcl=(804.0 0.0 0.0)
5 1 -5.3  1 -2 101 -102 201 -206 trcl=(804.0 0.0 0.0) imp:p=1
259 0 1 -7 101 -107 201 -207 #5 trcl=(804.0 0.0 0.0) imp:p=0
c
262 like 250 but trcl=(1005.0 0.0 0.0)
6 1 -5.3  1 -2 101 -102 201 -207 trcl=(1005.0 0.0 0.0) imp:p=1
261 0 1 -7 101 -107 201 -207 #6 trcl=(1005.0 0.0 0.0) imp:p=0
c
264 like 250 but trcl=(1206.0 0.0 0.0)
7 1 -5.3  1 -2 101 -103 201 -202 trcl=(1206.0 0.0 0.0) imp:p=1
263 0 1 -7 101 -107 201 -207 #7 trcl=(1206.0 0.0 0.0) imp:p=0
c
266 like 250 but trcl=(1407.0 0.0 0.0)
8 1 -5.3  1 -2 101 -103 201 -203 trcl=(1407.0 0.0 0.0) imp:p=1
265 0 1 -7 101 -107 201 -207 #8 trcl=(1407.0 0.0 0.0) imp:p=0
c
268 like 250 but trcl=(1608.0 0.0 0.0)
9 1 -5.3  1 -2 101 -103 201 -204 trcl=(1608.0 0.0 0.0) imp:p=1
267 0 1 -7 101 -107 201 -207 #9 trcl=(1608.0 0.0 0.0) imp:p=0
c
270 like 250 but trcl=(1809.0 0.0 0.0)
10 1 -5.3  1 -2 101 -103 201 -205 trcl=(1809.0 0.0 0.0) imp:p=1
269 0 1 -7 101 -107 201 -207 #10 trcl=(1809.0 0.0 0.0) imp:p=0
c
272 like 250 but trcl=(2010.0 0.0 0.0)
11 1 -5.3  1 -2 101 -103 201 -206 trcl=(2010.0 0.0 0.0) imp:p=1
271 0 1 -7 101 -107 201 -207 #11 trcl=(2010.0 0.0 0.0) imp:p=0
c
274 like 250 but trcl=(2211.0 0.0 0.0)
12 1 -5.3  1 -2 101 -103 201 -207 trcl=(2211.0 0.0 0.0) imp:p=1
273 0 1 -7 101 -107 201 -207 #12 trcl=(2211.0 0.0 0.0) imp:p=0
c
276 like 250 but trcl=(2412.0 0.0 0.0)
13 1 -5.3  1 -2 101 -104 201 -202 trcl=(2412.0 0.0 0.0) imp:p=1
275 0 1 -7 101 -107 201 -207 #13 trcl=(2412.0 0.0 0.0) imp:p=0
c
278 like 250 but trcl=(2613.0 0.0 0.0)
14 1 -5.3  1 -2 101 -104 201 -203 trcl=(2613.0 0.0 0.0) imp:p=1
277 0 1 -7 101 -107 201 -207 #14 trcl=(2613.0 0.0 0.0) imp:p=0
c
280 like 250 but trcl=(2814.0 0.0 0.0)
15 1 -5.3  1 -2 101 -104 201 -204 trcl=(2814.0 0.0 0.0) imp:p=1
279 0 1 -7 101 -107 201 -207 #15 trcl=(2814.0 0.0 0.0) imp:p=0
c
282 like 250 but trcl=(3015.0 0.0 0.0)
16 1 -5.3  1 -2 101 -104 201 -205 trcl=(3015.0 0.0 0.0) imp:p=1
281 0 1 -7 101 -107 201 -207 #16 trcl=(3015.0 0.0 0.0) imp:p=0
c
284 like 250 but trcl=(3216.0 0.0 0.0)
17 1 -5.3  1 -2 101 -104 201 -206 trcl=(3216.0 0.0 0.0) imp:p=1
283 0 1 -7 101 -107 201 -207 #17 trcl=(3216.0 0.0 0.0) imp:p=0
c
286 like 250 but trcl=(3417.0 0.0 0.0)
18 1 -5.3  1 -2 101 -104 201 -207 trcl=(3417.0 0.0 0.0) imp:p=1
285 0 1 -7 101 -107 201 -207 #18 trcl=(3417.0 0.0 0.0) imp:p=0
c
288 like 250 but trcl=(3618.0 0.0 0.0)
19 1 -5.3  1 -2 101 -105 201 -202 trcl=(3618.0 0.0 0.0) imp:p=1
287 0 1 -7 101 -107 201 -207 #19 trcl=(3618.0 0.0 0.0) imp:p=0
c
290 like 250 but trcl=(3819.0 0.0 0.0)
20 1 -5.3  1 -2 101 -105 201 -203 trcl=(3819.0 0.0 0.0) imp:p=1
289 0 1 -7 101 -107 201 -207 #20 trcl=(3819.0 0.0 0.0) imp:p=0
```

```
c
292 like 250 but trcl=(4020.0 0.0 0.0)
21 1 -5.3  1 -2 101 -105 201 -204 trcl=(4020.0 0.0 0.0) imp:p=1
291 0 1 -7 101 -107 201 -207 #21 trcl=(4020.0 0.0 0.0) imp:p=0
c
294 like 250 but trcl=(4221.0 0.0 0.0)
22 1 -5.3  1 -2 101 -105 201 -205 trcl=(4221.0 0.0 0.0) imp:p=1
293 0 1 -7 101 -107 201 -207 #22 trcl=(4221.0 0.0 0.0) imp:p=0
c
296 like 250 but trcl=(4422.0 0.0 0.0)
23 1 -5.3  1 -2 101 -105 201 -206 trcl=(4422.0 0.0 0.0) imp:p=1
295 0 1 -7 101 -107 201 -207 #23 trcl=(4422.0 0.0 0.0) imp:p=0
c
298 like 250 but trcl=(4623.0 0.0 0.0)
24 1 -5.3  1 -2 101 -105 201 -207 trcl=(4623.0 0.0 0.0) imp:p=1
297 0 1 -7 101 -107 201 -207 #24 trcl=(4623.0 0.0 0.0) imp:p=0
c
300 like 250 but trcl=(4824.0 0.0 0.0)
25 1 -5.3  1 -2 101 -106 201 -202 trcl=(4824.0 0.0 0.0) imp:p=1
299 0 1 -7 101 -107 201 -207 #25 trcl=(4824.0 0.0 0.0) imp:p=0
c
302 like 250 but trcl=(5025.0 0.0 0.0)
26 1 -5.3  1 -2 101 -106 201 -203 trcl=(5025.0 0.0 0.0) imp:p=1
301 0 1 -7 101 -107 201 -207 #26 trcl=(5025.0 0.0 0.0) imp:p=0
c
304 like 250 but trcl=(5226.0 0.0 0.0)
27 1 -5.3  1 -2 101 -106 201 -204 trcl=(5226.0 0.0 0.0) imp:p=1
303 0 1 -7 101 -107 201 -207 #27 trcl=(5226.0 0.0 0.0) imp:p=0
c
306 like 250 but trcl=(5427.0 0.0 0.0)
28 1 -5.3  1 -2 101 -106 201 -205 trcl=(5427.0 0.0 0.0) imp:p=1
305 0 1 -7 101 -107 201 -207 #28 trcl=(5427.0 0.0 0.0) imp:p=0
c
308 like 250 but trcl=(5628.0 0.0 0.0)
29 1 -5.3  1 -2 101 -106 201 -206 trcl=(5628.0 0.0 0.0) imp:p=1
307 0 1 -7 101 -107 201 -207 #29 trcl=(5628.0 0.0 0.0) imp:p=0
c
310 like 250 but trcl=(5829.0 0.0 0.0)
30 1 -5.3  1 -2 101 -106 201 -207 trcl=(5829.0 0.0 0.0) imp:p=1
309 0 1 -7 101 -107 201 -207 #30 trcl=(5829.0 0.0 0.0) imp:p=0
c
312 like 250 but trcl=(6030.0 0.0 0.0)
31 1 -5.3  1 -2 101 -107 201 -202 trcl=(6030.0 0.0 0.0) imp:p=1
311 0 1 -7 101 -107 201 -207 #31 trcl=(6030.0 0.0 0.0) imp:p=0
c
314 like 250 but trcl=(6231.0 0.0 0.0)
32 1 -5.3  1 -2 101 -107 201 -203 trcl=(6231.0 0.0 0.0) imp:p=1
313 0 1 -7 101 -107 201 -207 #32 trcl=(6231.0 0.0 0.0) imp:p=0
c
316 like 250 but trcl=(6432.0 0.0 0.0)
33 1 -5.3  1 -2 101 -107 201 -204 trcl=(6432.0 0.0 0.0) imp:p=1
315 0 1 -7 101 -107 201 -207 #33 trcl=(6432.0 0.0 0.0) imp:p=0
c
318 like 250 but trcl=(6633.0 0.0 0.0)
34 1 -5.3  1 -2 101 -107 201 -205 trcl=(6633.0 0.0 0.0) imp:p=1
317 0 1 -7 101 -107 201 -207 #34 trcl=(6633.0 0.0 0.0) imp:p=0
c
320 like 250 but trcl=(6834.0 0.0 0.0)
35 1 -5.3  1 -2 101 -107 201 -206 trcl=(6834.0 0.0 0.0) imp:p=1
319 0 1 -7 101 -107 201 -207 #35 trcl=(6834.0 0.0 0.0) imp:p=0
c
322 like 250 but trcl=(7035.0 0.0 0.0)
36 1 -5.3  1 -2 101 -107 201 -207 trcl=(7035.0 0.0 0.0) imp:p=1
321 0 1 -7 101 -107 201 -207 #36 trcl=(7035.0 0.0 0.0) imp:p=0
c
324 like 250 but trcl=(7236.0 0.0 0.0)
37 1 -5.3  1 -3 101 -102 201 -202 trcl=(7236.0 0.0 0.0) imp:p=1
323 0 1 -7 101 -107 201 -207 #37 trcl=(7236.0 0.0 0.0) imp:p=0
c
326 like 250 but trcl=(7437.0 0.0 0.0)
38 1 -5.3  1 -3 101 -102 201 -203 trcl=(7437.0 0.0 0.0) imp:p=1
325 0 1 -7 101 -107 201 -207 #38 trcl=(7437.0 0.0 0.0) imp:p=0
c
328 like 250 but trcl=(7638.0 0.0 0.0)
39 1 -5.3  1 -3 101 -102 201 -204 trcl=(7638.0 0.0 0.0) imp:p=1
327 0 1 -7 101 -107 201 -207 #39 trcl=(7638.0 0.0 0.0) imp:p=0
```

```
c
330 like 250 but trcl=(7839.0 0.0 0.0)
40 1 −5.3  1 −3 101 −102 201 −205 trcl=(7839.0 0.0 0.0) imp:p=1
329 0 1 −7 101 −107 201 −207 #40 trcl=(7839.0 0.0 0.0) imp:p=0
c
332 like 250 but trcl=(8040.0 0.0 0.0)
41 1 −5.3  1 −3 101 −102 201 −206 trcl=(8040.0 0.0 0.0) imp:p=1
331 0 1 −7 101 −107 201 −207 #41 trcl=(8040.0 0.0 0.0) imp:p=0
c
334 like 250 but trcl=(8241.0 0.0 0.0)
42 1 −5.3  1 −3 101 −102 201 −207 trcl=(8241.0 0.0 0.0) imp:p=1
333 0 1 −7 101 −107 201 −207 #42 trcl=(8241.0 0.0 0.0) imp:p=0
c
336 like 250 but trcl=(8442.0 0.0 0.0)
43 1 −5.3  1 −3 101 −103 201 −202 trcl=(8442.0 0.0 0.0) imp:p=1
335 0 1 −7 101 −107 201 −207 #43 trcl=(8442.0 0.0 0.0) imp:p=0
c
338 like 250 but trcl=(8643.0 0.0 0.0)
44 1 −5.3  1 −3 101 −103 201 −203 trcl=(8643.0 0.0 0.0) imp:p=1
337 0 1 −7 101 −107 201 −207 #44 trcl=(8643.0 0.0 0.0) imp:p=0
c
340 like 250 but trcl=(8844.0 0.0 0.0)
45 1 −5.3  1 −3 101 −103 201 −204 trcl=(8844.0 0.0 0.0) imp:p=1
339 0 1 −7 101 −107 201 −207 #45 trcl=(8844.0 0.0 0.0) imp:p=0
c
342 like 250 but trcl=(9045.0 0.0 0.0)
46 1 −5.3  1 −3 101 −103 201 −205 trcl=(9045.0 0.0 0.0) imp:p=1
341 0 1 −7 101 −107 201 −207 #46 trcl=(9045.0 0.0 0.0) imp:p=0
c
344 like 250 but trcl=(9246.0 0.0 0.0)
47 1 −5.3  1 −3 101 −103 201 −206 trcl=(9246.0 0.0 0.0) imp:p=1
343 0 1 −7 101 −107 201 −207 #47 trcl=(9246.0 0.0 0.0) imp:p=0
c
346 like 250 but trcl=(9447.0 0.0 0.0)
48 1 −5.3  1 −3 101 −103 201 −207 trcl=(9447.0 0.0 0.0) imp:p=1
345 0 1 −7 101 −107 201 −207 #48 trcl=(9447.0 0.0 0.0) imp:p=0
c
348 like 250 but trcl=(9648.0 0.0 0.0)
49 1 −5.3  1 −3 101 −104 201 −202 trcl=(9648.0 0.0 0.0) imp:p=1
347 0 1 −7 101 −107 201 −207 #49 trcl=(9648.0 0.0 0.0) imp:p=0
c
350 like 250 but trcl=(9849.0 0.0 0.0)
50 1 −5.3  1 −3 101 −104 201 −203 trcl=(9849.0 0.0 0.0) imp:p=1
349 0 1 −7 101 −107 201 −207 #50 trcl=(9849.0 0.0 0.0) imp:p=0
c
352 like 250 but trcl=(10050.0 0.0 0.0)
51 1 −5.3  1 −3 101 −104 201 −204 trcl=(10050.0 0.0 0.0) imp:p=1
351 0 1 −7 101 −107 201 −207 #51 trcl=(10050.0 0.0 0.0) imp:p=0
c
354 like 250 but trcl=(10251.0 0.0 0.0)
52 1 −5.3  1 −3 101 −104 201 −205 trcl=(10251.0 0.0 0.0) imp:p=1
353 0 1 −7 101 −107 201 −207 #52 trcl=(10251.0 0.0 0.0) imp:p=0
c
356 like 250 but trcl=(10452.0 0.0 0.0)
53 1 −5.3  1 −3 101 −104 201 −206 trcl=(10452.0 0.0 0.0) imp:p=1
355 0 1 −7 101 −107 201 −207 #53 trcl=(10452.0 0.0 0.0) imp:p=0
c
358 like 250 but trcl=(10653.0 0.0 0.0)
54 1 −5.3  1 −3 101 −104 201 −207 trcl=(10653.0 0.0 0.0) imp:p=1
357 0 1 −7 101 −107 201 −207 #54 trcl=(10653.0 0.0 0.0) imp:p=0
c
360 like 250 but trcl=(10854.0 0.0 0.0)
55 1 −5.3  1 −3 101 −105 201 −202 trcl=(10854.0 0.0 0.0) imp:p=1
359 0 1 −7 101 −107 201 −207 #55 trcl=(10854.0 0.0 0.0) imp:p=0
c
362 like 250 but trcl=(11055.0 0.0 0.0)
56 1 −5.3  1 −3 101 −105 201 −203 trcl=(11055.0 0.0 0.0) imp:p=1
361 0 1 −7 101 −107 201 −207 #56 trcl=(11055.0 0.0 0.0) imp:p=0
c
364 like 250 but trcl=(11256.0 0.0 0.0)
57 1 −5.3  1 −3 101 −105 201 −204 trcl=(11256.0 0.0 0.0) imp:p=1
363 0 1 −7 101 −107 201 −207 #57 trcl=(11256.0 0.0 0.0) imp:p=0
c
366 like 250 but trcl=(11457.0 0.0 0.0)
58 1 −5.3  1 −3 101 −105 201 −205 trcl=(11457.0 0.0 0.0) imp:p=1
365 0 1 −7 101 −107 201 −207 #58 trcl=(11457.0 0.0 0.0) imp:p=0
```

```
c
368 like 250 but trcl=(11658.0 0.0 0.0)
59 1 −5.3  1 −3 101 −105 201 −206 trcl=(11658.0 0.0 0.0) imp:p=1
367 0 1 −7 101 −107 201 −207 #59 trcl=(11658.0 0.0 0.0) imp:p=0
c
370 like 250 but trcl=(11859.0 0.0 0.0)
60 1 −5.3  1 −3 101 −105 201 −207 trcl=(11859.0 0.0 0.0) imp:p=1
369 0 1 −7 101 −107 201 −207 #60 trcl=(11859.0 0.0 0.0) imp:p=0
c
372 like 250 but trcl=(12060.0 0.0 0.0)
61 1 −5.3  1 −3 101 −106 201 −202 trcl=(12060.0 0.0 0.0) imp:p=1
371 0 1 −7 101 −107 201 −207 #61 trcl=(12060.0 0.0 0.0) imp:p=0
c
374 like 250 but trcl=(12261.0 0.0 0.0)
62 1 −5.3  1 −3 101 −106 201 −203 trcl=(12261.0 0.0 0.0) imp:p=1
373 0 1 −7 101 −107 201 −207 #62 trcl=(12261.0 0.0 0.0) imp:p=0
c
376 like 250 but trcl=(12462.0 0.0 0.0)
63 1 −5.3  1 −3 101 −106 201 −204 trcl=(12462.0 0.0 0.0) imp:p=1
375 0 1 −7 101 −107 201 −207 #63 trcl=(12462.0 0.0 0.0) imp:p=0
c
378 like 250 but trcl=(12663.0 0.0 0.0)
64 1 −5.3  1 −3 101 −106 201 −205 trcl=(12663.0 0.0 0.0) imp:p=1
377 0 1 −7 101 −107 201 −207 #64 trcl=(12663.0 0.0 0.0) imp:p=0
c
380 like 250 but trcl=(12864.0 0.0 0.0)
65 1 −5.3  1 −3 101 −106 201 −206 trcl=(12864.0 0.0 0.0) imp:p=1
379 0 1 −7 101 −107 201 −207 #65 trcl=(12864.0 0.0 0.0) imp:p=0
c
382 like 250 but trcl=(13065.0 0.0 0.0)
66 1 −5.3  1 −3 101 −106 201 −207 trcl=(13065.0 0.0 0.0) imp:p=1
381 0 1 −7 101 −107 201 −207 #66 trcl=(13065.0 0.0 0.0) imp:p=0
c
384 like 250 but trcl=(13266.0 0.0 0.0)
67 1 −5.3  1 −3 101 −107 201 −202 trcl=(13266.0 0.0 0.0) imp:p=1
383 0 1 −7 101 −107 201 −207 #67 trcl=(13266.0 0.0 0.0) imp:p=0
c
386 like 250 but trcl=(13467.0 0.0 0.0)
68 1 −5.3  1 −3 101 −107 201 −203 trcl=(13467.0 0.0 0.0) imp:p=1
385 0 1 −7 101 −107 201 −207 #68 trcl=(13467.0 0.0 0.0) imp:p=0
c
388 like 250 but trcl=(13668.0 0.0 0.0)
69 1 −5.3  1 −3 101 −107 201 −204 trcl=(13668.0 0.0 0.0) imp:p=1
387 0 1 −7 101 −107 201 −207 #69 trcl=(13668.0 0.0 0.0) imp:p=0
c
390 like 250 but trcl=(13869.0 0.0 0.0)
70 1 −5.3  1 −3 101 −107 201 −205 trcl=(13869.0 0.0 0.0) imp:p=1
389 0 1 −7 101 −107 201 −207 #70 trcl=(13869.0 0.0 0.0) imp:p=0
c
392 like 250 but trcl=(14070.0 0.0 0.0)
71 1 −5.3  1 −3 101 −107 201 −206 trcl=(14070.0 0.0 0.0) imp:p=1
391 0 1 −7 101 −107 201 −207 #71 trcl=(14070.0 0.0 0.0) imp:p=0
c
394 like 250 but trcl=(14271.0 0.0 0.0)
72 1 −5.3  1 −3 101 −107 201 −207 trcl=(14271.0 0.0 0.0) imp:p=1
393 0 1 −7 101 −107 201 −207 #72 trcl=(14271.0 0.0 0.0) imp:p=0
c
396 like 250 but trcl=(14472.0 0.0 0.0)
73 1 −5.3  1 −4 101 −102 201 −202 trcl=(14472.0 0.0 0.0) imp:p=1
395 0 1 −7 101 −107 201 −207 #73 trcl=(14472.0 0.0 0.0) imp:p=0
c
398 like 250 but trcl=(14673.0 0.0 0.0)
74 1 −5.3  1 −4 101 −102 201 −203 trcl=(14673.0 0.0 0.0) imp:p=1
397 0 1 −7 101 −107 201 −207 #74 trcl=(14673.0 0.0 0.0) imp:p=0
c
400 like 250 but trcl=(14874.0 0.0 0.0)
75 1 −5.3  1 −4 101 −102 201 −204 trcl=(14874.0 0.0 0.0) imp:p=1
399 0 1 −7 101 −107 201 −207 #75 trcl=(14874.0 0.0 0.0) imp:p=0
c
402 like 250 but trcl=(15075.0 0.0 0.0)
76 1 −5.3  1 −4 101 −102 201 −205 trcl=(15075.0 0.0 0.0) imp:p=1
401 0 1 −7 101 −107 201 −207 #76 trcl=(15075.0 0.0 0.0) imp:p=0
c
404 like 250 but trcl=(15276.0 0.0 0.0)
77 1 −5.3  1 −4 101 −102 201 −206 trcl=(15276.0 0.0 0.0) imp:p=1
403 0 1 −7 101 −107 201 −207 #77 trcl=(15276.0 0.0 0.0) imp:p=0
```

```
c
406 like 250 but trcl=(15477.0 0.0 0.0)
78 1 -5.3  1 -4 101 -102 201 -207 trcl=(15477.0 0.0 0.0) imp:p=1
405 0 1 -7 101 -107 201 -207 #78 trcl=(15477.0 0.0 0.0) imp:p=0
c
408 like 250 but trcl=(15678.0 0.0 0.0)
79 1 -5.3  1 -4 101 -103 201 -202 trcl=(15678.0 0.0 0.0) imp:p=1
407 0 1 -7 101 -107 201 -207 #79 trcl=(15678.0 0.0 0.0) imp:p=0
c
410 like 250 but trcl=(15879.0 0.0 0.0)
80 1 -5.3  1 -4 101 -103 201 -203 trcl=(15879.0 0.0 0.0) imp:p=1
409 0 1 -7 101 -107 201 -207 #80 trcl=(15879.0 0.0 0.0) imp:p=0
c
412 like 250 but trcl=(16080.0 0.0 0.0)
81 1 -5.3  1 -4 101 -103 201 -204 trcl=(16080.0 0.0 0.0) imp:p=1
411 0 1 -7 101 -107 201 -207 #81 trcl=(16080.0 0.0 0.0) imp:p=0
c
414 like 250 but trcl=(16281.0 0.0 0.0)
82 1 -5.3  1 -4 101 -103 201 -205 trcl=(16281.0 0.0 0.0) imp:p=1
413 0 1 -7 101 -107 201 -207 #82 trcl=(16281.0 0.0 0.0) imp:p=0
c
416 like 250 but trcl=(16482.0 0.0 0.0)
83 1 -5.3  1 -4 101 -103 201 -206 trcl=(16482.0 0.0 0.0) imp:p=1
415 0 1 -7 101 -107 201 -207 #83 trcl=(16482.0 0.0 0.0) imp:p=0
c
418 like 250 but trcl=(16683.0 0.0 0.0)
84 1 -5.3  1 -4 101 -103 201 -207 trcl=(16683.0 0.0 0.0) imp:p=1
417 0 1 -7 101 -107 201 -207 #84 trcl=(16683.0 0.0 0.0) imp:p=0
c
420 like 250 but trcl=(16884.0 0.0 0.0)
85 1 -5.3  1 -4 101 -104 201 -202 trcl=(16884.0 0.0 0.0) imp:p=1
419 0 1 -7 101 -107 201 -207 #85 trcl=(16884.0 0.0 0.0) imp:p=0
c
422 like 250 but trcl=(17085.0 0.0 0.0)
86 1 -5.3  1 -4 101 -104 201 -203 trcl=(17085.0 0.0 0.0) imp:p=1
421 0 1 -7 101 -107 201 -207 #86 trcl=(17085.0 0.0 0.0) imp:p=0
c
424 like 250 but trcl=(17286.0 0.0 0.0)
87 1 -5.3  1 -4 101 -104 201 -204 trcl=(17286.0 0.0 0.0) imp:p=1
423 0 1 -7 101 -107 201 -207 #87 trcl=(17286.0 0.0 0.0) imp:p=0
c
426 like 250 but trcl=(17487.0 0.0 0.0)
88 1 -5.3  1 -4 101 -104 201 -205 trcl=(17487.0 0.0 0.0) imp:p=1
425 0 1 -7 101 -107 201 -207 #88 trcl=(17487.0 0.0 0.0) imp:p=0
c
428 like 250 but trcl=(17688.0 0.0 0.0)
89 1 -5.3  1 -4 101 -104 201 -206 trcl=(17688.0 0.0 0.0) imp:p=1
427 0 1 -7 101 -107 201 -207 #89 trcl=(17688.0 0.0 0.0) imp:p=0
c
430 like 250 but trcl=(17889.0 0.0 0.0)
90 1 -5.3  1 -4 101 -104 201 -207 trcl=(17889.0 0.0 0.0) imp:p=1
429 0 1 -7 101 -107 201 -207 #90 trcl=(17889.0 0.0 0.0) imp:p=0
c
432 like 250 but trcl=(18090.0 0.0 0.0)
91 1 -5.3  1 -4 101 -105 201 -202 trcl=(18090.0 0.0 0.0) imp:p=1
431 0 1 -7 101 -107 201 -207 #91 trcl=(18090.0 0.0 0.0) imp:p=0
c
434 like 250 but trcl=(18291.0 0.0 0.0)
92 1 -5.3  1 -4 101 -105 201 -203 trcl=(18291.0 0.0 0.0) imp:p=1
433 0 1 -7 101 -107 201 -207 #92 trcl=(18291.0 0.0 0.0) imp:p=0
c
436 like 250 but trcl=(18492.0 0.0 0.0)
93 1 -5.3  1 -4 101 -105 201 -204 trcl=(18492.0 0.0 0.0) imp:p=1
435 0 1 -7 101 -107 201 -207 #93 trcl=(18492.0 0.0 0.0) imp:p=0
c
438 like 250 but trcl=(18693.0 0.0 0.0)
94 1 -5.3  1 -4 101 -105 201 -205 trcl=(18693.0 0.0 0.0) imp:p=1
437 0 1 -7 101 -107 201 -207 #94 trcl=(18693.0 0.0 0.0) imp:p=0
c
440 like 250 but trcl=(18894.0 0.0 0.0)
95 1 -5.3  1 -4 101 -105 201 -206 trcl=(18894.0 0.0 0.0) imp:p=1
439 0 1 -7 101 -107 201 -207 #95 trcl=(18894.0 0.0 0.0) imp:p=0
c
442 like 250 but trcl=(19095.0 0.0 0.0)
96 1 -5.3  1 -4 101 -105 201 -207 trcl=(19095.0 0.0 0.0) imp:p=1
441 0 1 -7 101 -107 201 -207 #96 trcl=(19095.0 0.0 0.0) imp:p=0
```

```
c
444 like 250 but trcl=(19296.0 0.0 0.0)
97 1 -5.3  1 -4 101 -106 201 -202 trcl=(19296.0 0.0 0.0) imp:p=1
443 0 1 -7 101 -107 201 -207 #97 trcl=(19296.0 0.0 0.0) imp:p=0
c
446 like 250 but trcl=(19497.0 0.0 0.0)
98 1 -5.3  1 -4 101 -106 201 -203 trcl=(19497.0 0.0 0.0) imp:p=1
445 0 1 -7 101 -107 201 -207 #98 trcl=(19497.0 0.0 0.0) imp:p=0
c
448 like 250 but trcl=(19698.0 0.0 0.0)
99 1 -5.3  1 -4 101 -106 201 -204 trcl=(19698.0 0.0 0.0) imp:p=1
447 0 1 -7 101 -107 201 -207 #99 trcl=(19698.0 0.0 0.0) imp:p=0
c
450 like 250 but trcl=(19899.0 0.0 0.0)
100 1 -5.3  1 -4 101 -106 201 -205 trcl=(19899.0 0.0 0.0) imp:p=1
449 0 1 -7 101 -107 201 -207 #100 trcl=(19899.0 0.0 0.0) imp:p=0
c
452 like 250 but trcl=(20100.0 0.0 0.0)
101 1 -5.3  1 -4 101 -106 201 -206 trcl=(20100.0 0.0 0.0) imp:p=1
451 0 1 -7 101 -107 201 -207 #101 trcl=(20100.0 0.0 0.0) imp:p=0
c
454 like 250 but trcl=(20301.0 0.0 0.0)
102 1 -5.3  1 -4 101 -106 201 -207 trcl=(20301.0 0.0 0.0) imp:p=1
453 0 1 -7 101 -107 201 -207 #102 trcl=(20301.0 0.0 0.0) imp:p=0
c
456 like 250 but trcl=(20502.0 0.0 0.0)
103 1 -5.3  1 -4 101 -107 201 -202 trcl=(20502.0 0.0 0.0) imp:p=1
455 0 1 -7 101 -107 201 -207 #103 trcl=(20502.0 0.0 0.0) imp:p=0
c
458 like 250 but trcl=(20703.0 0.0 0.0)
104 1 -5.3  1 -4 101 -107 201 -203 trcl=(20703.0 0.0 0.0) imp:p=1
457 0 1 -7 101 -107 201 -207 #104 trcl=(20703.0 0.0 0.0) imp:p=0
c
460 like 250 but trcl=(20904.0 0.0 0.0)
105 1 -5.3  1 -4 101 -107 201 -204 trcl=(20904.0 0.0 0.0) imp:p=1
459 0 1 -7 101 -107 201 -207 #105 trcl=(20904.0 0.0 0.0) imp:p=0
c
462 like 250 but trcl=(21105.0 0.0 0.0)
106 1 -5.3  1 -4 101 -107 201 -205 trcl=(21105.0 0.0 0.0) imp:p=1
461 0 1 -7 101 -107 201 -207 #106 trcl=(21105.0 0.0 0.0) imp:p=0
c
464 like 250 but trcl=(21306.0 0.0 0.0)
107 1 -5.3  1 -4 101 -107 201 -206 trcl=(21306.0 0.0 0.0) imp:p=1
463 0 1 -7 101 -107 201 -207 #107 trcl=(21306.0 0.0 0.0) imp:p=0
c
466 like 250 but trcl=(21507.0 0.0 0.0)
108 1 -5.3  1 -4 101 -107 201 -207 trcl=(21507.0 0.0 0.0) imp:p=1
465 0 1 -7 101 -107 201 -207 #108 trcl=(21507.0 0.0 0.0) imp:p=0
c
468 like 250 but trcl=(21708.0 0.0 0.0)
109 1 -5.3  1 -5 101 -102 201 -202 trcl=(21708.0 0.0 0.0) imp:p=1
467 0 1 -7 101 -107 201 -207 #109 trcl=(21708.0 0.0 0.0) imp:p=0
c
470 like 250 but trcl=(21909.0 0.0 0.0)
110 1 -5.3  1 -5 101 -102 201 -203 trcl=(21909.0 0.0 0.0) imp:p=1
469 0 1 -7 101 -107 201 -207 #110 trcl=(21909.0 0.0 0.0) imp:p=0
c
472 like 250 but trcl=(22110.0 0.0 0.0)
111 1 -5.3  1 -5 101 -102 201 -204 trcl=(22110.0 0.0 0.0) imp:p=1
471 0 1 -7 101 -107 201 -207 #111 trcl=(22110.0 0.0 0.0) imp:p=0
c
474 like 250 but trcl=(22311.0 0.0 0.0)
112 1 -5.3  1 -5 101 -102 201 -205 trcl=(22311.0 0.0 0.0) imp:p=1
473 0 1 -7 101 -107 201 -207 #112 trcl=(22311.0 0.0 0.0) imp:p=0
c
476 like 250 but trcl=(22512.0 0.0 0.0)
113 1 -5.3  1 -5 101 -102 201 -206 trcl=(22512.0 0.0 0.0) imp:p=1
475 0 1 -7 101 -107 201 -207 #113 trcl=(22512.0 0.0 0.0) imp:p=0
c
478 like 250 but trcl=(22713.0 0.0 0.0)
114 1 -5.3  1 -5 101 -102 201 -207 trcl=(22713.0 0.0 0.0) imp:p=1
477 0 1 -7 101 -107 201 -207 #114 trcl=(22713.0 0.0 0.0) imp:p=0
c
480 like 250 but trcl=(22914.0 0.0 0.0)
115 1 -5.3  1 -5 101 -103 201 -202 trcl=(22914.0 0.0 0.0) imp:p=1
479 0 1 -7 101 -107 201 -207 #115 trcl=(22914.0 0.0 0.0) imp:p=0
```

```
c
482 like 250 but trcl=(23115.0 0.0 0.0)
116 1 -5.3  1 -5 101 -103 201 -203 trcl=(23115.0 0.0 0.0) imp:p=1
481 0 1 -7 101 -107 201 -207 #116 trcl=(23115.0 0.0 0.0) imp:p=0
c
484 like 250 but trcl=(23316.0 0.0 0.0)
117 1 -5.3  1 -5 101 -103 201 -204 trcl=(23316.0 0.0 0.0) imp:p=1
483 0 1 -7 101 -107 201 -207 #117 trcl=(23316.0 0.0 0.0) imp:p=0
c
486 like 250 but trcl=(23517.0 0.0 0.0)
118 1 -5.3  1 -5 101 -103 201 -205 trcl=(23517.0 0.0 0.0) imp:p=1
485 0 1 -7 101 -107 201 -207 #118 trcl=(23517.0 0.0 0.0) imp:p=0
c
488 like 250 but trcl=(23718.0 0.0 0.0)
119 1 -5.3  1 -5 101 -103 201 -206 trcl=(23718.0 0.0 0.0) imp:p=1
487 0 1 -7 101 -107 201 -207 #119 trcl=(23718.0 0.0 0.0) imp:p=0
c
490 like 250 but trcl=(23919.0 0.0 0.0)
120 1 -5.3  1 -5 101 -103 201 -207 trcl=(23919.0 0.0 0.0) imp:p=1
489 0 1 -7 101 -107 201 -207 #120 trcl=(23919.0 0.0 0.0) imp:p=0
c
492 like 250 but trcl=(24120.0 0.0 0.0)
121 1 -5.3  1 -5 101 -104 201 -202 trcl=(24120.0 0.0 0.0) imp:p=1
491 0 1 -7 101 -107 201 -207 #121 trcl=(24120.0 0.0 0.0) imp:p=0
c
494 like 250 but trcl=(24321.0 0.0 0.0)
122 1 -5.3  1 -5 101 -104 201 -203 trcl=(24321.0 0.0 0.0) imp:p=1
493 0 1 -7 101 -107 201 -207 #122 trcl=(24321.0 0.0 0.0) imp:p=0
c
496 like 250 but trcl=(24522.0 0.0 0.0)
123 1 -5.3  1 -5 101 -104 201 -204 trcl=(24522.0 0.0 0.0) imp:p=1
495 0 1 -7 101 -107 201 -207 #123 trcl=(24522.0 0.0 0.0) imp:p=0
c
498 like 250 but trcl=(24723.0 0.0 0.0)
124 1 -5.3  1 -5 101 -104 201 -205 trcl=(24723.0 0.0 0.0) imp:p=1
497 0 1 -7 101 -107 201 -207 #124 trcl=(24723.0 0.0 0.0) imp:p=0
c
500 like 250 but trcl=(24924.0 0.0 0.0)
125 1 -5.3  1 -5 101 -104 201 -206 trcl=(24924.0 0.0 0.0) imp:p=1
499 0 1 -7 101 -107 201 -207 #125 trcl=(24924.0 0.0 0.0) imp:p=0
c
502 like 250 but trcl=(25125.0 0.0 0.0)
126 1 -5.3  1 -5 101 -104 201 -207 trcl=(25125.0 0.0 0.0) imp:p=1
501 0 1 -7 101 -107 201 -207 #126 trcl=(25125.0 0.0 0.0) imp:p=0
c
504 like 250 but trcl=(25326.0 0.0 0.0)
127 1 -5.3  1 -5 101 -105 201 -202 trcl=(25326.0 0.0 0.0) imp:p=1
503 0 1 -7 101 -107 201 -207 #127 trcl=(25326.0 0.0 0.0) imp:p=0
c
506 like 250 but trcl=(25527.0 0.0 0.0)
128 1 -5.3  1 -5 101 -105 201 -203 trcl=(25527.0 0.0 0.0) imp:p=1
505 0 1 -7 101 -107 201 -207 #128 trcl=(25527.0 0.0 0.0) imp:p=0
c
508 like 250 but trcl=(25728.0 0.0 0.0)
129 1 -5.3  1 -5 101 -105 201 -204 trcl=(25728.0 0.0 0.0) imp:p=1
507 0 1 -7 101 -107 201 -207 #129 trcl=(25728.0 0.0 0.0) imp:p=0
c
510 like 250 but trcl=(25929.0 0.0 0.0)
130 1 -5.3  1 -5 101 -105 201 -205 trcl=(25929.0 0.0 0.0) imp:p=1
509 0 1 -7 101 -107 201 -207 #130 trcl=(25929.0 0.0 0.0) imp:p=0
c
512 like 250 but trcl=(26130.0 0.0 0.0)
131 1 -5.3  1 -5 101 -105 201 -206 trcl=(26130.0 0.0 0.0) imp:p=1
511 0 1 -7 101 -107 201 -207 #131 trcl=(26130.0 0.0 0.0) imp:p=0
c
514 like 250 but trcl=(26331.0 0.0 0.0)
132 1 -5.3  1 -5 101 -105 201 -207 trcl=(26331.0 0.0 0.0) imp:p=1
513 0 1 -7 101 -107 201 -207 #132 trcl=(26331.0 0.0 0.0) imp:p=0
c
516 like 250 but trcl=(26532.0 0.0 0.0)
133 1 -5.3  1 -5 101 -106 201 -202 trcl=(26532.0 0.0 0.0) imp:p=1
515 0 1 -7 101 -107 201 -207 #133 trcl=(26532.0 0.0 0.0) imp:p=0
c
518 like 250 but trcl=(26733.0 0.0 0.0)
134 1 -5.3  1 -5 101 -106 201 -203 trcl=(26733.0 0.0 0.0) imp:p=1
517 0 1 -7 101 -107 201 -207 #134 trcl=(26733.0 0.0 0.0) imp:p=0
```

```
c
520 like 250 but trcl=(26934.0 0.0 0.0)
135 1 −5.3  1 −5 101 −106 201 −204 trcl=(26934.0 0.0 0.0) imp:p=1
519 0 1 −7 101 −107 201 −207 #135 trcl=(26934.0 0.0 0.0) imp:p=0
c
522 like 250 but trcl=(27135.0 0.0 0.0)
136 1 −5.3  1 −5 101 −106 201 −205 trcl=(27135.0 0.0 0.0) imp:p=1
521 0 1 −7 101 −107 201 −207 #136 trcl=(27135.0 0.0 0.0) imp:p=0
c
524 like 250 but trcl=(27336.0 0.0 0.0)
137 1 −5.3  1 −5 101 −106 201 −206 trcl=(27336.0 0.0 0.0) imp:p=1
523 0 1 −7 101 −107 201 −207 #137 trcl=(27336.0 0.0 0.0) imp:p=0
c
526 like 250 but trcl=(27537.0 0.0 0.0)
138 1 −5.3  1 −5 101 −106 201 −207 trcl=(27537.0 0.0 0.0) imp:p=1
525 0 1 −7 101 −107 201 −207 #138 trcl=(27537.0 0.0 0.0) imp:p=0
c
528 like 250 but trcl=(27738.0 0.0 0.0)
139 1 −5.3  1 −5 101 −107 201 −202 trcl=(27738.0 0.0 0.0) imp:p=1
527 0 1 −7 101 −107 201 −207 #139 trcl=(27738.0 0.0 0.0) imp:p=0
c
530 like 250 but trcl=(27939.0 0.0 0.0)
140 1 −5.3  1 −5 101 −107 201 −203 trcl=(27939.0 0.0 0.0) imp:p=1
529 0 1 −7 101 −107 201 −207 #140 trcl=(27939.0 0.0 0.0) imp:p=0
c
532 like 250 but trcl=(28140.0 0.0 0.0)
141 1 −5.3  1 −5 101 −107 201 −204 trcl=(28140.0 0.0 0.0) imp:p=1
531 0 1 −7 101 −107 201 −207 #141 trcl=(28140.0 0.0 0.0) imp:p=0
c
534 like 250 but trcl=(28341.0 0.0 0.0)
142 1 −5.3  1 −5 101 −107 201 −205 trcl=(28341.0 0.0 0.0) imp:p=1
533 0 1 −7 101 −107 201 −207 #142 trcl=(28341.0 0.0 0.0) imp:p=0
c
536 like 250 but trcl=(28542.0 0.0 0.0)
143 1 −5.3  1 −5 101 −107 201 −206 trcl=(28542.0 0.0 0.0) imp:p=1
535 0 1 −7 101 −107 201 −207 #143 trcl=(28542.0 0.0 0.0) imp:p=0
c
538 like 250 but trcl=(28743.0 0.0 0.0)
144 1 −5.3  1 −5 101 −107 201 −207 trcl=(28743.0 0.0 0.0) imp:p=1
537 0 1 −7 101 −107 201 −207 #144 trcl=(28743.0 0.0 0.0) imp:p=0
c
540 like 250 but trcl=(28944.0 0.0 0.0)
145 1 −5.3  1 −6 101 −102 201 −202 trcl=(28944.0 0.0 0.0) imp:p=1
539 0 1 −7 101 −107 201 −207 #145 trcl=(28944.0 0.0 0.0) imp:p=0
c
542 like 250 but trcl=(29145.0 0.0 0.0)
146 1 −5.3  1 −6 101 −102 201 −203 trcl=(29145.0 0.0 0.0) imp:p=1
541 0 1 −7 101 −107 201 −207 #146 trcl=(29145.0 0.0 0.0) imp:p=0
c
544 like 250 but trcl=(29346.0 0.0 0.0)
147 1 −5.3  1 −6 101 −102 201 −204 trcl=(29346.0 0.0 0.0) imp:p=1
543 0 1 −7 101 −107 201 −207 #147 trcl=(29346.0 0.0 0.0) imp:p=0
c
546 like 250 but trcl=(29547.0 0.0 0.0)
148 1 −5.3  1 −6 101 −102 201 −205 trcl=(29547.0 0.0 0.0) imp:p=1
545 0 1 −7 101 −107 201 −207 #148 trcl=(29547.0 0.0 0.0) imp:p=0
c
548 like 250 but trcl=(29748.0 0.0 0.0)
149 1 −5.3  1 −6 101 −102 201 −206 trcl=(29748.0 0.0 0.0) imp:p=1
547 0 1 −7 101 −107 201 −207 #149 trcl=(29748.0 0.0 0.0) imp:p=0
c
550 like 250 but trcl=(29949.0 0.0 0.0)
150 1 −5.3  1 −6 101 −102 201 −207 trcl=(29949.0 0.0 0.0) imp:p=1
549 0 1 −7 101 −107 201 −207 #150 trcl=(29949.0 0.0 0.0) imp:p=0
c
552 like 250 but trcl=(30150.0 0.0 0.0)
151 1 −5.3  1 −6 101 −103 201 −202 trcl=(30150.0 0.0 0.0) imp:p=1
551 0 1 −7 101 −107 201 −207 #151 trcl=(30150.0 0.0 0.0) imp:p=0
c
554 like 250 but trcl=(30351.0 0.0 0.0)
152 1 −5.3  1 −6 101 −103 201 −203 trcl=(30351.0 0.0 0.0) imp:p=1
553 0 1 −7 101 −107 201 −207 #152 trcl=(30351.0 0.0 0.0) imp:p=0
c
556 like 250 but trcl=(30552.0 0.0 0.0)
153 1 −5.3  1 −6 101 −103 201 −204 trcl=(30552.0 0.0 0.0) imp:p=1
555 0 1 −7 101 −107 201 −207 #153 trcl=(30552.0 0.0 0.0) imp:p=0
```

```
c
558 like 250 but trcl=(30753.0 0.0 0.0)
154 1 −5.3 1 −6 101 −103 201 −205 trcl=(30753.0 0.0 0.0) imp:p=1
557 0 1 −7 101 −107 201 −207 #154 trcl=(30753.0 0.0 0.0) imp:p=0
c
560 like 250 but trcl=(30954.0 0.0 0.0)
155 1 −5.3 1 −6 101 −103 201 −206 trcl=(30954.0 0.0 0.0) imp:p=1
559 0 1 −7 101 −107 201 −207 #155 trcl=(30954.0 0.0 0.0) imp:p=0
c
562 like 250 but trcl=(31155.0 0.0 0.0)
156 1 −5.3 1 −6 101 −103 201 −207 trcl=(31155.0 0.0 0.0) imp:p=1
561 0 1 −7 101 −107 201 −207 #156 trcl=(31155.0 0.0 0.0) imp:p=0
c
564 like 250 but trcl=(31356.0 0.0 0.0)
157 1 −5.3 1 −6 101 −104 201 −202 trcl=(31356.0 0.0 0.0) imp:p=1
563 0 1 −7 101 −107 201 −207 #157 trcl=(31356.0 0.0 0.0) imp:p=0
c
566 like 250 but trcl=(31557.0 0.0 0.0)
158 1 −5.3 1 −6 101 −104 201 −203 trcl=(31557.0 0.0 0.0) imp:p=1
565 0 1 −7 101 −107 201 −207 #158 trcl=(31557.0 0.0 0.0) imp:p=0
c
568 like 250 but trcl=(31758.0 0.0 0.0)
159 1 −5.3 1 −6 101 −104 201 −204 trcl=(31758.0 0.0 0.0) imp:p=1
567 0 1 −7 101 −107 201 −207 #159 trcl=(31758.0 0.0 0.0) imp:p=0
c
570 like 250 but trcl=(31959.0 0.0 0.0)
160 1 −5.3 1 −6 101 −104 201 −205 trcl=(31959.0 0.0 0.0) imp:p=1
569 0 1 −7 101 −107 201 −207 #160 trcl=(31959.0 0.0 0.0) imp:p=0
c
572 like 250 but trcl=(32160.0 0.0 0.0)
161 1 −5.3 1 −6 101 −104 201 −206 trcl=(32160.0 0.0 0.0) imp:p=1
571 0 1 −7 101 −107 201 −207 #161 trcl=(32160.0 0.0 0.0) imp:p=0
c
574 like 250 but trcl=(32361.0 0.0 0.0)
162 1 −5.3 1 −6 101 −104 201 −207 trcl=(32361.0 0.0 0.0) imp:p=1
573 0 1 −7 101 −107 201 −207 #162 trcl=(32361.0 0.0 0.0) imp:p=0
c
576 like 250 but trcl=(32562.0 0.0 0.0)
163 1 −5.3 1 −6 101 −105 201 −202 trcl=(32562.0 0.0 0.0) imp:p=1
575 0 1 −7 101 −107 201 −207 #163 trcl=(32562.0 0.0 0.0) imp:p=0
c
578 like 250 but trcl=(32763.0 0.0 0.0)
164 1 −5.3 1 −6 101 −105 201 −203 trcl=(32763.0 0.0 0.0) imp:p=1
577 0 1 −7 101 −107 201 −207 #164 trcl=(32763.0 0.0 0.0) imp:p=0
c
580 like 250 but trcl=(32964.0 0.0 0.0)
165 1 −5.3 1 −6 101 −105 201 −204 trcl=(32964.0 0.0 0.0) imp:p=1
579 0 1 −7 101 −107 201 −207 #165 trcl=(32964.0 0.0 0.0) imp:p=0
c
582 like 250 but trcl=(33165.0 0.0 0.0)
166 1 −5.3 1 −6 101 −105 201 −205 trcl=(33165.0 0.0 0.0) imp:p=1
581 0 1 −7 101 −107 201 −207 #166 trcl=(33165.0 0.0 0.0) imp:p=0
c
584 like 250 but trcl=(33366.0 0.0 0.0)
167 1 −5.3 1 −6 101 −105 201 −206 trcl=(33366.0 0.0 0.0) imp:p=1
583 0 1 −7 101 −107 201 −207 #167 trcl=(33366.0 0.0 0.0) imp:p=0
c
586 like 250 but trcl=(33567.0 0.0 0.0)
168 1 −5.3 1 −6 101 −105 201 −207 trcl=(33567.0 0.0 0.0) imp:p=1
585 0 1 −7 101 −107 201 −207 #168 trcl=(33567.0 0.0 0.0) imp:p=0
c
588 like 250 but trcl=(33768.0 0.0 0.0)
169 1 −5.3 1 −6 101 −106 201 −202 trcl=(33768.0 0.0 0.0) imp:p=1
587 0 1 −7 101 −107 201 −207 #169 trcl=(33768.0 0.0 0.0) imp:p=0
c
590 like 250 but trcl=(33969.0 0.0 0.0)
170 1 −5.3 1 −6 101 −106 201 −203 trcl=(33969.0 0.0 0.0) imp:p=1
589 0 1 −7 101 −107 201 −207 #170 trcl=(33969.0 0.0 0.0) imp:p=0
c
592 like 250 but trcl=(34170.0 0.0 0.0)
171 1 −5.3 1 −6 101 −106 201 −204 trcl=(34170.0 0.0 0.0) imp:p=1
591 0 1 −7 101 −107 201 −207 #171 trcl=(34170.0 0.0 0.0) imp:p=0
c
594 like 250 but trcl=(34371.0 0.0 0.0)
172 1 −5.3 1 −6 101 −106 201 −205 trcl=(34371.0 0.0 0.0) imp:p=1
593 0 1 −7 101 −107 201 −207 #172 trcl=(34371.0 0.0 0.0) imp:p=0
```

```
c
596 like 250 but trcl=(34572.0 0.0 0.0)
173 1 −5.3  1 −6 101 −106 201 −206 trcl=(34572.0 0.0 0.0) imp:p=1
595 0 1 −7 101 −107 201 −207 #173 trcl=(34572.0 0.0 0.0) imp:p=0
c
598 like 250 but trcl=(34773.0 0.0 0.0)
174 1 −5.3  1 −6 101 −106 201 −207 trcl=(34773.0 0.0 0.0) imp:p=1
597 0 1 −7 101 −107 201 −207 #174 trcl=(34773.0 0.0 0.0) imp:p=0
c
600 like 250 but trcl=(34974.0 0.0 0.0)
175 1 −5.3  1 −6 101 −107 201 −202 trcl=(34974.0 0.0 0.0) imp:p=1
599 0 1 −7 101 −107 201 −207 #175 trcl=(34974.0 0.0 0.0) imp:p=0
c
602 like 250 but trcl=(35175.0 0.0 0.0)
176 1 −5.3  1 −6 101 −107 201 −203 trcl=(35175.0 0.0 0.0) imp:p=1
601 0 1 −7 101 −107 201 −207 #176 trcl=(35175.0 0.0 0.0) imp:p=0
c
604 like 250 but trcl=(35376.0 0.0 0.0)
177 1 −5.3  1 −6 101 −107 201 −204 trcl=(35376.0 0.0 0.0) imp:p=1
603 0 1 −7 101 −107 201 −207 #177 trcl=(35376.0 0.0 0.0) imp:p=0
c
606 like 250 but trcl=(35577.0 0.0 0.0)
178 1 −5.3  1 −6 101 −107 201 −205 trcl=(35577.0 0.0 0.0) imp:p=1
605 0 1 −7 101 −107 201 −207 #178 trcl=(35577.0 0.0 0.0) imp:p=0
c
608 like 250 but trcl=(35778.0 0.0 0.0)
179 1 −5.3  1 −6 101 −107 201 −206 trcl=(35778.0 0.0 0.0) imp:p=1
607 0 1 −7 101 −107 201 −207 #179 trcl=(35778.0 0.0 0.0) imp:p=0
c
610 like 250 but trcl=(35979.0 0.0 0.0)
180 1 −5.3  1 −6 101 −107 201 −207 trcl=(35979.0 0.0 0.0) imp:p=1
609 0 1 −7 101 −107 201 −207 #180 trcl=(35979.0 0.0 0.0) imp:p=0
c
612 like 250 but trcl=(36180.0 0.0 0.0)
181 1 −5.3  1 −7 101 −102 201 −202 trcl=(36180.0 0.0 0.0) imp:p=1
611 0 1 −7 101 −107 201 −207 #181 trcl=(36180.0 0.0 0.0) imp:p=0
c
614 like 250 but trcl=(36381.0 0.0 0.0)
182 1 −5.3  1 −7 101 −102 201 −203 trcl=(36381.0 0.0 0.0) imp:p=1
613 0 1 −7 101 −107 201 −207 #182 trcl=(36381.0 0.0 0.0) imp:p=0
c
616 like 250 but trcl=(36582.0 0.0 0.0)
183 1 −5.3  1 −7 101 −102 201 −204 trcl=(36582.0 0.0 0.0) imp:p=1
615 0 1 −7 101 −107 201 −207 #183 trcl=(36582.0 0.0 0.0) imp:p=0
c
618 like 250 but trcl=(36783.0 0.0 0.0)
184 1 −5.3  1 −7 101 −102 201 −205 trcl=(36783.0 0.0 0.0) imp:p=1
617 0 1 −7 101 −107 201 −207 #184 trcl=(36783.0 0.0 0.0) imp:p=0
c
620 like 250 but trcl=(36984.0 0.0 0.0)
185 1 −5.3  1 −7 101 −102 201 −206 trcl=(36984.0 0.0 0.0) imp:p=1
619 0 1 −7 101 −107 201 −207 #185 trcl=(36984.0 0.0 0.0) imp:p=0
c
622 like 250 but trcl=(37185.0 0.0 0.0)
186 1 −5.3  1 −7 101 −102 201 −207 trcl=(37185.0 0.0 0.0) imp:p=1
621 0 1 −7 101 −107 201 −207 #186 trcl=(37185.0 0.0 0.0) imp:p=0
c
624 like 250 but trcl=(37386.0 0.0 0.0)
187 1 −5.3  1 −7 101 −103 201 −202 trcl=(37386.0 0.0 0.0) imp:p=1
623 0 1 −7 101 −107 201 −207 #187 trcl=(37386.0 0.0 0.0) imp:p=0
c
626 like 250 but trcl=(37587.0 0.0 0.0)
188 1 −5.3  1 −7 101 −103 201 −203 trcl=(37587.0 0.0 0.0) imp:p=1
625 0 1 −7 101 −107 201 −207 #188 trcl=(37587.0 0.0 0.0) imp:p=0
c
628 like 250 but trcl=(37788.0 0.0 0.0)
189 1 −5.3  1 −7 101 −103 201 −204 trcl=(37788.0 0.0 0.0) imp:p=1
627 0 1 −7 101 −107 201 −207 #189 trcl=(37788.0 0.0 0.0) imp:p=0
c
630 like 250 but trcl=(37989.0 0.0 0.0)
190 1 −5.3  1 −7 101 −103 201 −205 trcl=(37989.0 0.0 0.0) imp:p=1
629 0 1 −7 101 −107 201 −207 #190 trcl=(37989.0 0.0 0.0) imp:p=0
c
632 like 250 but trcl=(38190.0 0.0 0.0)
191 1 −5.3  1 −7 101 −103 201 −206 trcl=(38190.0 0.0 0.0) imp:p=1
631 0 1 −7 101 −107 201 −207 #191 trcl=(38190.0 0.0 0.0) imp:p=0
```

```
c
634 like 250 but trcl=(38391.0 0.0 0.0)
192 1 -5.3  1 -7 101 -103 201 -207 trcl=(38391.0 0.0 0.0) imp:p=1
633 0 1 -7 101 -107 201 -207 #192 trcl=(38391.0 0.0 0.0) imp:p=0
c
636 like 250 but trcl=(38592.0 0.0 0.0)
193 1 -5.3  1 -7 101 -104 201 -202 trcl=(38592.0 0.0 0.0) imp:p=1
635 0 1 -7 101 -107 201 -207 #193 trcl=(38592.0 0.0 0.0) imp:p=0
c
638 like 250 but trcl=(38793.0 0.0 0.0)
194 1 -5.3  1 -7 101 -104 201 -203 trcl=(38793.0 0.0 0.0) imp:p=1
637 0 1 -7 101 -107 201 -207 #194 trcl=(38793.0 0.0 0.0) imp:p=0
c
640 like 250 but trcl=(38994.0 0.0 0.0)
195 1 -5.3  1 -7 101 -104 201 -204 trcl=(38994.0 0.0 0.0) imp:p=1
639 0 1 -7 101 -107 201 -207 #195 trcl=(38994.0 0.0 0.0) imp:p=0
c
642 like 250 but trcl=(39195.0 0.0 0.0)
196 1 -5.3  1 -7 101 -104 201 -205 trcl=(39195.0 0.0 0.0) imp:p=1
641 0 1 -7 101 -107 201 -207 #196 trcl=(39195.0 0.0 0.0) imp:p=0
c
644 like 250 but trcl=(39396.0 0.0 0.0)
197 1 -5.3  1 -7 101 -104 201 -206 trcl=(39396.0 0.0 0.0) imp:p=1
643 0 1 -7 101 -107 201 -207 #197 trcl=(39396.0 0.0 0.0) imp:p=0
c
646 like 250 but trcl=(39597.0 0.0 0.0)
198 1 -5.3  1 -7 101 -104 201 -207 trcl=(39597.0 0.0 0.0) imp:p=1
645 0 1 -7 101 -107 201 -207 #198 trcl=(39597.0 0.0 0.0) imp:p=0
c
648 like 250 but trcl=(39798.0 0.0 0.0)
199 1 -5.3  1 -7 101 -105 201 -202 trcl=(39798.0 0.0 0.0) imp:p=1
647 0 1 -7 101 -107 201 -207 #199 trcl=(39798.0 0.0 0.0) imp:p=0
c
650 like 250 but trcl=(39999.0 0.0 0.0)
200 1 -5.3  1 -7 101 -105 201 -203 trcl=(39999.0 0.0 0.0) imp:p=1
649 0 1 -7 101 -107 201 -207 #200 trcl=(39999.0 0.0 0.0) imp:p=0
c
652 like 250 but trcl=(40200.0 0.0 0.0)
201 1 -5.3  1 -7 101 -105 201 -204 trcl=(40200.0 0.0 0.0) imp:p=1
651 0 1 -7 101 -107 201 -207 #201 trcl=(40200.0 0.0 0.0) imp:p=0
c
654 like 250 but trcl=(40401.0 0.0 0.0)
202 1 -5.3  1 -7 101 -105 201 -205 trcl=(40401.0 0.0 0.0) imp:p=1
653 0 1 -7 101 -107 201 -207 #202 trcl=(40401.0 0.0 0.0) imp:p=0
c
656 like 250 but trcl=(40602.0 0.0 0.0)
203 1 -5.3  1 -7 101 -105 201 -206 trcl=(40602.0 0.0 0.0) imp:p=1
655 0 1 -7 101 -107 201 -207 #203 trcl=(40602.0 0.0 0.0) imp:p=0
c
658 like 250 but trcl=(40803.0 0.0 0.0)
204 1 -5.3  1 -7 101 -105 201 -207 trcl=(40803.0 0.0 0.0) imp:p=1
657 0 1 -7 101 -107 201 -207 #204 trcl=(40803.0 0.0 0.0) imp:p=0
c
660 like 250 but trcl=(41004.0 0.0 0.0)
205 1 -5.3  1 -7 101 -106 201 -202 trcl=(41004.0 0.0 0.0) imp:p=1
659 0 1 -7 101 -107 201 -207 #205 trcl=(41004.0 0.0 0.0) imp:p=0
c
662 like 250 but trcl=(41205.0 0.0 0.0)
206 1 -5.3  1 -7 101 -106 201 -203 trcl=(41205.0 0.0 0.0) imp:p=1
661 0 1 -7 101 -107 201 -207 #206 trcl=(41205.0 0.0 0.0) imp:p=0
c
664 like 250 but trcl=(41406.0 0.0 0.0)
207 1 -5.3  1 -7 101 -106 201 -204 trcl=(41406.0 0.0 0.0) imp:p=1
663 0 1 -7 101 -107 201 -207 #207 trcl=(41406.0 0.0 0.0) imp:p=0
c
666 like 250 but trcl=(41607.0 0.0 0.0)
208 1 -5.3  1 -7 101 -106 201 -205 trcl=(41607.0 0.0 0.0) imp:p=1
665 0 1 -7 101 -107 201 -207 #208 trcl=(41607.0 0.0 0.0) imp:p=0
c
668 like 250 but trcl=(41808.0 0.0 0.0)
209 1 -5.3  1 -7 101 -106 201 -206 trcl=(41808.0 0.0 0.0) imp:p=1
667 0 1 -7 101 -107 201 -207 #209 trcl=(41808.0 0.0 0.0) imp:p=0
c
670 like 250 but trcl=(42009.0 0.0 0.0)
210 1 -5.3  1 -7 101 -106 201 -207 trcl=(42009.0 0.0 0.0) imp:p=1
669 0 1 -7 101 -107 201 -207 #210 trcl=(42009.0 0.0 0.0) imp:p=0
```

```
c
672 like 250 but trcl=(42210.0 0.0 0.0)
211 1 -5.3  1 -7 101 -107 201 -202 trcl=(42210.0 0.0 0.0) imp:p=1
671 0 1 -7 101 -107 201 -207 #211 trcl=(42210.0 0.0 0.0) imp:p=0
c
674 like 250 but trcl=(42411.0 0.0 0.0)
212 1 -5.3  1 -7 101 -107 201 -203 trcl=(42411.0 0.0 0.0) imp:p=1
673 0 1 -7 101 -107 201 -207 #212 trcl=(42411.0 0.0 0.0) imp:p=0
c
676 like 250 but trcl=(42612.0 0.0 0.0)
213 1 -5.3  1 -7 101 -107 201 -204 trcl=(42612.0 0.0 0.0) imp:p=1
675 0 1 -7 101 -107 201 -207 #213 trcl=(42612.0 0.0 0.0) imp:p=0
c
678 like 250 but trcl=(42813.0 0.0 0.0)
214 1 -5.3  1 -7 101 -107 201 -205 trcl=(42813.0 0.0 0.0) imp:p=1
677 0 1 -7 101 -107 201 -207 #214 trcl=(42813.0 0.0 0.0) imp:p=0
c
680 like 250 but trcl=(43014.0 0.0 0.0)
215 1 -5.3  1 -7 101 -107 201 -206 trcl=(43014.0 0.0 0.0) imp:p=1
679 0 1 -7 101 -107 201 -207 #215 trcl=(43014.0 0.0 0.0) imp:p=0
c
682 like 250 but trcl=(43215.0 0.0 0.0)
216 1 -5.3  1 -7 101 -107 201 -207 trcl=(43215.0 0.0 0.0) imp:p=1
681 0 1 -7 101 -107 201 -207 #216 trcl=(43215.0 0.0 0.0) imp:p=0
998 0 -1:998:-101:107:-201:207 imp:p=0

1 px 0
2 px 1.0000000000e+00
3 px 5.0000000000e+00
4 px 2.0000000000e+01
5 px 5.0000000000e+01
6 px 1.0000000000e+02
7 px 2.0000000000e+02
101 py 0
102 py 1.0000000000e+00
103 py 5.0000000000e+00
104 py 2.0000000000e+01
105 py 5.0000000000e+01
106 py 1.0000000000e+02
107 py 2.0000000000e+02
201 pz 0
202 pz 1.0000000000e+00
203 pz 5.0000000000e+00
204 pz 2.0000000000e+01
205 pz 5.0000000000e+01
206 pz 1.0000000000e+02
207 pz 2.0000000000e+02
999 px 201
998 px 43416

m1
        32000 1.0
mode p
sdef par=p x=d998 y=0 z=fx d999 erg=1.0000000000e-02 vec=0 1 0 dir=1
si998 S     1     2     3     4     5     6
       7     8     9     10     11     12     13
       14     15     16     17     18     19     20
       21     22     23     24     25     26     27
       28     29     30     31     32     33     34
       35     36     37     38     39     40     41
       42     43     44     45     46     47     48
       49     50     51     52     53     54     55
       56     57     58     59     60     61     62
       63     64     65     66     67     68     69
       70     71     72     73     74     75     76
       77     78     79     80     81     82     83
       84     85     86     87     88     89     90
       91     92     93     94     95     96     97
       98     99     100     101     102     103     104
       105     106     107     108     109     110     111
       112     113     114     115     116     117     118
       119     120     121     122     123     124     125
       126     127     128     129     130     131     132
       133     134     135     136     137     138     139
       140     141     142     143     144     145     146
       147     148     149     150     151     152     153
```

```
      154      155      156      157      158      159      160
      161      162      163      164      165      166      167
      168      169      170      171      172      173      174
      175      176      177      178      179      180      181
      182      183      184      185      186      187      188
      189      190      191      192      193      194      195
      196      197      198      199      200      201      202
      203      204      205      206      207      208      209
      210      211      212      213      214      215      216
sp998
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1      1
      1      1      1      1      1      1
si1 0 1
sp1 0 1
si2 201 202
sp2 0 1
si3 402 403
sp3 0 1
si4 603 604
sp4 0 1
si5 804 805
sp5 0 1
si6 1005 1006
sp6 0 1
si7 1206 1207
sp7 0 1
si8 1407 1408
sp8 0 1
si9 1608 1609
sp9 0 1
si10 1809 1810
sp10 0 1
si11 2010 2011
sp11 0 1
si12 2211 2212
sp12 0 1
si13 2412 2413
sp13 0 1
si14 2613 2614
sp14 0 1
si15 2814 2815
sp15 0 1
si16 3015 3016
sp16 0 1
si17 3216 3217
sp17 0 1
si18 3417 3418
```

```
sp18 0 1
si19 3618 3619
sp19 0 1
si20 3819 3820
sp20 0 1
si21 4020 4021
sp21 0 1
si22 4221 4222
sp22 0 1
si23 4422 4423
sp23 0 1
si24 4623 4624
sp24 0 1
si25 4824 4825
sp25 0 1
si26 5025 5026
sp26 0 1
si27 5226 5227
sp27 0 1
si28 5427 5428
sp28 0 1
si29 5628 5629
sp29 0 1
si30 5829 5830
sp30 0 1
si31 6030 6031
sp31 0 1
si32 6231 6232
sp32 0 1
si33 6432 6433
sp33 0 1
si34 6633 6634
sp34 0 1
si35 6834 6835
sp35 0 1
si36 7035 7036
sp36 0 1
si37 7236 7241
sp37 0 1
si38 7437 7442
sp38 0 1
si39 7638 7643
sp39 0 1
si40 7839 7844
sp40 0 1
si41 8040 8045
sp41 0 1
si42 8241 8246
sp42 0 1
si43 8442 8447
sp43 0 1
si44 8643 8648
sp44 0 1
si45 8844 8849
sp45 0 1
si46 9045 9050
sp46 0 1
si47 9246 9251
sp47 0 1
si48 9447 9452
sp48 0 1
si49 9648 9653
sp49 0 1
si50 9849 9854
sp50 0 1
si51 10050 10055
sp51 0 1
si52 10251 10256
sp52 0 1
si53 10452 10457
sp53 0 1
si54 10653 10658
sp54 0 1
si55 10854 10859
sp55 0 1
si56 11055 11060
```

```
sp56  0  1
si57  11256  11261
sp57  0  1
si58  11457  11462
sp58  0  1
si59  11658  11663
sp59  0  1
si60  11859  11864
sp60  0  1
si61  12060  12065
sp61  0  1
si62  12261  12266
sp62  0  1
si63  12462  12467
sp63  0  1
si64  12663  12668
sp64  0  1
si65  12864  12869
sp65  0  1
si66  13065  13070
sp66  0  1
si67  13266  13271
sp67  0  1
si68  13467  13472
sp68  0  1
si69  13668  13673
sp69  0  1
si70  13869  13874
sp70  0  1
si71  14070  14075
sp71  0  1
si72  14271  14276
sp72  0  1
si73  14472  14492
sp73  0  1
si74  14673  14693
sp74  0  1
si75  14874  14894
sp75  0  1
si76  15075  15095
sp76  0  1
si77  15276  15296
sp77  0  1
si78  15477  15497
sp78  0  1
si79  15678  15698
sp79  0  1
si80  15879  15899
sp80  0  1
si81  16080  16100
sp81  0  1
si82  16281  16301
sp82  0  1
si83  16482  16502
sp83  0  1
si84  16683  16703
sp84  0  1
si85  16884  16904
sp85  0  1
si86  17085  17105
sp86  0  1
si87  17286  17306
sp87  0  1
si88  17487  17507
sp88  0  1
si89  17688  17708
sp89  0  1
si90  17889  17909
sp90  0  1
si91  18090  18110
sp91  0  1
si92  18291  18311
sp92  0  1
si93  18492  18512
sp93  0  1
si94  18693  18713
```

```
sp94 0 1
si95 18894 18914
sp95 0 1
si96 19095 19115
sp96 0 1
si97 19296 19316
sp97 0 1
si98 19497 19517
sp98 0 1
si99 19698 19718
sp99 0 1
si100 19899 19919
sp100 0 1
si101 20100 20120
sp101 0 1
si102 20301 20321
sp102 0 1
si103 20502 20522
sp103 0 1
si104 20703 20723
sp104 0 1
si105 20904 20924
sp105 0 1
si106 21105 21125
sp106 0 1
si107 21306 21326
sp107 0 1
si108 21507 21527
sp108 0 1
si109 21708 21758
sp109 0 1
si110 21909 21959
sp110 0 1
si111 22110 22160
sp111 0 1
si112 22311 22361
sp112 0 1
si113 22512 22562
sp113 0 1
si114 22713 22763
sp114 0 1
si115 22914 22964
sp115 0 1
si116 23115 23165
sp116 0 1
si117 23316 23366
sp117 0 1
si118 23517 23567
sp118 0 1
si119 23718 23768
sp119 0 1
si120 23919 23969
sp120 0 1
si121 24120 24170
sp121 0 1
si122 24321 24371
sp122 0 1
si123 24522 24572
sp123 0 1
si124 24723 24773
sp124 0 1
si125 24924 24974
sp125 0 1
si126 25125 25175
sp126 0 1
si127 25326 25376
sp127 0 1
si128 25527 25577
sp128 0 1
si129 25728 25778
sp129 0 1
si130 25929 25979
sp130 0 1
si131 26130 26180
sp131 0 1
si132 26331 26381
```

```
sp132 0 1
si133 26532 26582
sp133 0 1
si134 26733 26783
sp134 0 1
si135 26934 26984
sp135 0 1
si136 27135 27185
sp136 0 1
si137 27336 27386
sp137 0 1
si138 27537 27587
sp138 0 1
si139 27738 27788
sp139 0 1
si140 27939 27989
sp140 0 1
si141 28140 28190
sp141 0 1
si142 28341 28391
sp142 0 1
si143 28542 28592
sp143 0 1
si144 28743 28793
sp144 0 1
si145 28944 29044
sp145 0 1
si146 29145 29245
sp146 0 1
si147 29346 29446
sp147 0 1
si148 29547 29647
sp148 0 1
si149 29748 29848
sp149 0 1
si150 29949 30049
sp150 0 1
si151 30150 30250
sp151 0 1
si152 30351 30451
sp152 0 1
si153 30552 30652
sp153 0 1
si154 30753 30853
sp154 0 1
si155 30954 31054
sp155 0 1
si156 31155 31255
sp156 0 1
si157 31356 31456
sp157 0 1
si158 31557 31657
sp158 0 1
si159 31758 31858
sp159 0 1
si160 31959 32059
sp160 0 1
si161 32160 32260
sp161 0 1
si162 32361 32461
sp162 0 1
si163 32562 32662
sp163 0 1
si164 32763 32863
sp164 0 1
si165 32964 33064
sp165 0 1
si166 33165 33265
sp166 0 1
si167 33366 33466
sp167 0 1
si168 33567 33667
sp168 0 1
si169 33768 33868
sp169 0 1
si170 33969 34069
```

```
sp170 0 1
si171 34170 34270
sp171 0 1
si172 34371 34471
sp172 0 1
si173 34572 34672
sp173 0 1
si174 34773 34873
sp174 0 1
si175 34974 35074
sp175 0 1
si176 35175 35275
sp176 0 1
si177 35376 35476
sp177 0 1
si178 35577 35677
sp178 0 1
si179 35778 35878
sp179 0 1
si180 35979 36079
sp180 0 1
si181 36180 36380
sp181 0 1
si182 36381 36581
sp182 0 1
si183 36582 36782
sp183 0 1
si184 36783 36983
sp184 0 1
si185 36984 37184
sp185 0 1
si186 37185 37385
sp186 0 1
si187 37386 37586
sp187 0 1
si188 37587 37787
sp188 0 1
si189 37788 37988
sp189 0 1
si190 37989 38189
sp190 0 1
si191 38190 38390
sp191 0 1
si192 38391 38591
sp192 0 1
si193 38592 38792
sp193 0 1
si194 38793 38993
sp194 0 1
si195 38994 39194
sp195 0 1
si196 39195 39395
sp196 0 1
si197 39396 39596
sp197 0 1
si198 39597 39797
sp198 0 1
si199 39798 39998
sp199 0 1
si200 39999 40199
sp200 0 1
si201 40200 40400
sp201 0 1
si202 40401 40601
sp202 0 1
si203 40602 40802
sp203 0 1
si204 40803 41003
sp204 0 1
si205 41004 41204
sp205 0 1
si206 41205 41405
sp206 0 1
si207 41406 41606
sp207 0 1
si208 41607 41807
```

```
sp208 0 1
si209 41808 42008
sp209 0 1
si210 42009 42209
sp210 0 1
si211 42210 42410
sp211 0 1
si212 42411 42611
sp212 0 1
si213 42612 42812
sp213 0 1
si214 42813 43013
sp214 0 1
si215 43014 43214
sp215 0 1
si216 43215 43415
sp216 0 1
ds999 S       217       218       219       220       221
      222       223       224       225       226       227
      228       229       230       231       232       233
      234       235       236       237       238       239
      240       241       242       243       244       245
      246       247       248       249       250       251
      252       253       254       255       256       257
      258       259       260       261       262       263
      264       265       266       267       268       269
      270       271       272       273       274       275
      276       277       278       279       280       281
      282       283       284       285       286       287
      288       289       290       291       292       293
      294       295       296       297       298       299
      300       301       302       303       304       305
      306       307       308       309       310       311
      312       313       314       315       316       317
      318       319       320       321       322       323
      324       325       326       327       328       329
      330       331       332       333       334       335
      336       337       338       339       340       341
      342       343       344       345       346       347
      348       349       350       351       352       353
      354       355       356       357       358       359
      360       361       362       363       364       365
      366       367       368       369       370       371
      372       373       374       375       376       377
      378       379       380       381       382       383
      384       385       386       387       388       389
      390       391       392       393       394       395
      396       397       398       399       400       401
      402       403       404       405       406       407
      408       409       410       411       412       413
      414       415       416       417       418       419
      420       421       422       423       424       425
      426       427       428       429       430       431
      432
si217 0 1
sp217 0 1
si218 0 5
sp218 0 1
si219 0 20
sp219 0 1
si220 0 50
sp220 0 1
si221 0 100
sp221 0 1
si222 0 200
sp222 0 1
si223 0 1
sp223 0 1
si224 0 5
sp224 0 1
si225 0 20
sp225 0 1
si226 0 50
sp226 0 1
si227 0 100
sp227 0 1
```

```
si228 0 200
sp228 0 1
si229 0 1
sp229 0 1
si230 0 5
sp230 0 1
si231 0 20
sp231 0 1
si232 0 50
sp232 0 1
si233 0 100
sp233 0 1
si234 0 200
sp234 0 1
si235 0 1
sp235 0 1
si236 0 5
sp236 0 1
si237 0 20
sp237 0 1
si238 0 50
sp238 0 1
si239 0 100
sp239 0 1
si240 0 200
sp240 0 1
si241 0 1
sp241 0 1
si242 0 5
sp242 0 1
si243 0 20
sp243 0 1
si244 0 50
sp244 0 1
si245 0 100
sp245 0 1
si246 0 200
sp246 0 1
si247 0 1
sp247 0 1
si248 0 5
sp248 0 1
si249 0 20
sp249 0 1
si250 0 50
sp250 0 1
si251 0 100
sp251 0 1
si252 0 200
sp252 0 1
si253 0 1
sp253 0 1
si254 0 5
sp254 0 1
si255 0 20
sp255 0 1
si256 0 50
sp256 0 1
si257 0 100
sp257 0 1
si258 0 200
sp258 0 1
si259 0 1
sp259 0 1
si260 0 5
sp260 0 1
si261 0 20
sp261 0 1
si262 0 50
sp262 0 1
si263 0 100
sp263 0 1
si264 0 200
sp264 0 1
si265 0 1
sp265 0 1
```

```
si266 0 5
sp266 0 1
si267 0 20
sp267 0 1
si268 0 50
sp268 0 1
si269 0 100
sp269 0 1
si270 0 200
sp270 0 1
si271 0 1
sp271 0 1
si272 0 5
sp272 0 1
si273 0 20
sp273 0 1
si274 0 50
sp274 0 1
si275 0 100
sp275 0 1
si276 0 200
sp276 0 1
si277 0 1
sp277 0 1
si278 0 5
sp278 0 1
si279 0 20
sp279 0 1
si280 0 50
sp280 0 1
si281 0 100
sp281 0 1
si282 0 200
sp282 0 1
si283 0 1
sp283 0 1
si284 0 5
sp284 0 1
si285 0 20
sp285 0 1
si286 0 50
sp286 0 1
si287 0 100
sp287 0 1
si288 0 200
sp288 0 1
si289 0 1
sp289 0 1
si290 0 5
sp290 0 1
si291 0 20
sp291 0 1
si292 0 50
sp292 0 1
si293 0 100
sp293 0 1
si294 0 200
sp294 0 1
si295 0 1
sp295 0 1
si296 0 5
sp296 0 1
si297 0 20
sp297 0 1
si298 0 50
sp298 0 1
si299 0 100
sp299 0 1
si300 0 200
sp300 0 1
si301 0 1
sp301 0 1
si302 0 5
sp302 0 1
si303 0 20
sp303 0 1
```

```
si304 0 50
sp304 0 1
si305 0 100
sp305 0 1
si306 0 200
sp306 0 1
si307 0 1
sp307 0 1
si308 0 5
sp308 0 1
si309 0 20
sp309 0 1
si310 0 50
sp310 0 1
si311 0 100
sp311 0 1
si312 0 200
sp312 0 1
si313 0 1
sp313 0 1
si314 0 5
sp314 0 1
si315 0 20
sp315 0 1
si316 0 50
sp316 0 1
si317 0 100
sp317 0 1
si318 0 200
sp318 0 1
si319 0 1
sp319 0 1
si320 0 5
sp320 0 1
si321 0 20
sp321 0 1
si322 0 50
sp322 0 1
si323 0 100
sp323 0 1
si324 0 200
sp324 0 1
si325 0 1
sp325 0 1
si326 0 5
sp326 0 1
si327 0 20
sp327 0 1
si328 0 50
sp328 0 1
si329 0 100
sp329 0 1
si330 0 200
sp330 0 1
si331 0 1
sp331 0 1
si332 0 5
sp332 0 1
si333 0 20
sp333 0 1
si334 0 50
sp334 0 1
si335 0 100
sp335 0 1
si336 0 200
sp336 0 1
si337 0 1
sp337 0 1
si338 0 5
sp338 0 1
si339 0 20
sp339 0 1
si340 0 50
sp340 0 1
si341 0 100
sp341 0 1
```

```
si342 0 200
sp342 0 1
si343 0 1
sp343 0 1
si344 0 5
sp344 0 1
si345 0 20
sp345 0 1
si346 0 50
sp346 0 1
si347 0 100
sp347 0 1
si348 0 200
sp348 0 1
si349 0 1
sp349 0 1
si350 0 5
sp350 0 1
si351 0 20
sp351 0 1
si352 0 50
sp352 0 1
si353 0 100
sp353 0 1
si354 0 200
sp354 0 1
si355 0 1
sp355 0 1
si356 0 5
sp356 0 1
si357 0 20
sp357 0 1
si358 0 50
sp358 0 1
si359 0 100
sp359 0 1
si360 0 200
sp360 0 1
si361 0 1
sp361 0 1
si362 0 5
sp362 0 1
si363 0 20
sp363 0 1
si364 0 50
sp364 0 1
si365 0 100
sp365 0 1
si366 0 200
sp366 0 1
si367 0 1
sp367 0 1
si368 0 5
sp368 0 1
si369 0 20
sp369 0 1
si370 0 50
sp370 0 1
si371 0 100
sp371 0 1
si372 0 200
sp372 0 1
si373 0 1
sp373 0 1
si374 0 5
sp374 0 1
si375 0 20
sp375 0 1
si376 0 50
sp376 0 1
si377 0 100
sp377 0 1
si378 0 200
sp378 0 1
si379 0 1
sp379 0 1
```

```
si380 0 5
sp380 0 1
si381 0 20
sp381 0 1
si382 0 50
sp382 0 1
si383 0 100
sp383 0 1
si384 0 200
sp384 0 1
si385 0 1
sp385 0 1
si386 0 5
sp386 0 1
si387 0 20
sp387 0 1
si388 0 50
sp388 0 1
si389 0 100
sp389 0 1
si390 0 200
sp390 0 1
si391 0 1
sp391 0 1
si392 0 5
sp392 0 1
si393 0 20
sp393 0 1
si394 0 50
sp394 0 1
si395 0 100
sp395 0 1
si396 0 200
sp396 0 1
si397 0 1
sp397 0 1
si398 0 5
sp398 0 1
si399 0 20
sp399 0 1
si400 0 50
sp400 0 1
si401 0 100
sp401 0 1
si402 0 200
sp402 0 1
si403 0 1
sp403 0 1
si404 0 5
sp404 0 1
si405 0 20
sp405 0 1
si406 0 50
sp406 0 1
si407 0 100
sp407 0 1
si408 0 200
sp408 0 1
si409 0 1
sp409 0 1
si410 0 5
sp410 0 1
si411 0 20
sp411 0 1
si412 0 50
sp412 0 1
si413 0 100
sp413 0 1
si414 0 200
sp414 0 1
si415 0 1
sp415 0 1
si416 0 5
sp416 0 1
si417 0 20
sp417 0 1
```

```
si418 0 50
sp418 0 1
si419 0 100
sp419 0 1
si420 0 200
sp420 0 1
si421 0 1
sp421 0 1
si422 0 5
sp422 0 1
si423 0 20
sp423 0 1
si424 0 50
sp424 0 1
si425 0 100
sp425 0 1
si426 0 200
sp426 0 1
si427 0 1
sp427 0 1
si428 0 5
sp428 0 1
si429 0 20
sp429 0 1
si430 0 50
sp430 0 1
si431 0 100
sp431 0 1
si432 0 200
sp432 0 1
e0
         0
         1e-5
        1.0000000000e-03
        1.9765548681e-03
        2.9531097362e-03
        3.9296646044e-03
        4.9062194725e-03
        5.8827743406e-03
        6.8593292087e-03
        7.8358840768e-03
        8.8124389450e-03
        9.7889938131e-03
        1.0765548681e-02
f8:p
        1         2         3         4         5         6         7         8
        9         10         11         12         13         14         15
        16         17         18         19         20         21         22
        23         24         25         26         27         28         29
        30         31         32         33         34         35         36
        37         38         39         40         41         42         43
        44         45         46         47         48         49         50
        51         52         53         54         55         56         57
        58         59         60         61         62         63         64
        65         66         67         68         69         70         71
        72         73         74         75         76         77         78
        79         80         81         82         83         84         85
        86         87         88         89         90         91         92
        93         94         95         96         97         98         99
        100         101         102         103         104         105
        106         107         108         109         110         111
        112         113         114         115         116         117
        118         119         120         121         122         123
        124         125         126         127         128         129
        130         131         132         133         134         135
        136         137         138         139         140         141
        142         143         144         145         146         147
        148         149         150         151         152         153
        154         155         156         157         158         159
        160         161         162         163         164         165
        166         167         168         169         170         171
        172         173         174         175         176         177
        178         179         180         181         182         183
        184         185         186         187         188         189
        190         191         192         193         194         195
        196         197         198         199         200         201
```

```
      202      203      204      205      206      207
      208      209      210      211      212      213
      214      215      216
prdmp j j 1
nps 1e9
```

Listing A.5: Example Terrestrial Background Input File

```
k40 soil
c -- Cell Cards --
1    1 -2.42   -1 +3 -4  imp:p=3   $ concrete
2    2 -1.6    -1 +2 -3  imp:p=1   $ soil
4    0 (1:-2:4)          imp:p=0   $ void

c -- Surface Cards --
*1    cz  200.0   $ world radius (reflecting)
2     pz  -200.0  $ world bottom (2 meters deep)
3     pz  -30.0   $ soil concrete division
4     pz  0       $ ground and tally surf

c -- Data Cards --
mode  p
c Source Definition for Gammas
sdef pos=0 0 0 axs=0 0 1 ext=d1 rad=d2 erg=1.46083 par=p
si1  -200.0 -30.0
si2  0 200.0
sp2 -21 1
nps 2.1e9
c Concrete
m1     26056  -0.009    $Fe-56
       20040  -0.2375   $Ca-40
       20042  -0.0016   $Ca-42
       20043  -0.0003   $Ca-43
       20044  -0.007    $Ca-44
       20048  -0.0005   $Ca-48
       14028  -0.1337   $Si-28
       14029  -0.0068   $Si-29
       14030  -0.0045   $Si-30
       13027  -0.014    $Al-27
       16032  -0.001    $S-32
       6012   -0.057    $C-12
       8016   -0.499    $O-16
       1001   -0.008    $H-1
c Soil
m2     8016   -0.490    $O-16
       14028  -0.3044   $Si-28
       14029  -0.0154   $Si-29
       14030  -0.0102   $Si-30
       13027  -0.07     $Al-27
       26054  -0.0023   $Fe-54
       26056  -0.0367   $Fe-56
       26057  -0.0008   $Fe-57
       6012   -0.01     $C-12
       20040  -0.01     $Ca-40
       19039  -0.01     $K-39
       11023  -0.007    $Na-23
       12024  -0.006    $Mg-24
       22048  -0.005    $Ti-48
       7014   -0.001    $N-14
c Air
m3     7014   0.7664 $N-14
       8016   0.2336 $O-16
c wwp:p j j j j -1 0 j j
c wwg 11 0 0 j j j j 0
c mesh  geom=cyl
c       ref=0 0 -1e-5
c       origin=-201 0 0
c       axs=0 0 1
c       imesh=200.1
c       iints=1
c       jmesh=50 100 150 175 202
c       kmesh=1
c       kints=1
c f11:p 4
```

```
f1:p 4
c (energy bin structure omitted for brevity)
```

Listing A.6: Example Neutron SNM Input File

```
heu 100 gram neutron spectrum
1 1 −18.95 −1 imp:n,p=1
2 0 1 imp:n,p=0

1 so 1.08003

mode n p
nps 1e9
ctme 240
totnu no
sdef rad=d1 par=−sf cel=1 pos=0 0 0
si1 0 1.08003
sp1 −21 2
f11:p 1
e11 1e−3 100i 10.0
f1:n 1
e1
        1.0000000000e−10 1.3901159058e−10 1.9324222317e−10 2.6862908810e−10
        3.7342556814e−10 5.1910482191e−10 7.2161586974e−10 1.0031296984e−09
        1.3944665494e−09 1.9384701305e−09 2.6946981613e−09 3.7459427755e−09
        5.2072946346e−09 7.2387430979e−09 1.0062691919e−08 1.3988308092e−08
        1.9445369574e−08 2.7031317539e−08 3.7576664467e−08 5.2235918964e−08
        7.2613981808e−08 1.0094185110e−07 1.4032087277e−07 1.9506227716e−07
        2.7115917411e−07 3.7694268095e−07 5.2399401637e−07 7.2841241672e−07
        1.0125776865e−06 1.4076003479e−06 1.9567276326e−06 2.7200782055e−06
        3.7812239786e−06 5.2563395962e−06 7.3069212791e−06 1.0157467493e−05
        1.4120057125e−05 1.9628516000e−05 2.7285912300e−05 3.7930580693e−05
        5.2727903539e−05 7.3297897391e−05 1.0189257303e−04 1.4164248645e−04
        1.9689947336e−04 2.7371308977e−04 3.8049291972e−04 5.2892925976e−04
        7.3527297705e−04 1.0221146605e−03 1.4208578472e−03 1.9751570933e−03
        2.7456972919e−03 3.8168374781e−03 5.3058464883e−03 7.3757415973e−03
        1.0253135712e−02 1.4253047038e−02 1.9813387394e−02 2.7542904964e−02
        3.8287830284e−02 5.3224521877e−02 7.3988254442e−02 1.0285224934e−01
        1.4297654776e−01 1.9875397321e−01 2.7629105951e−01 3.8407659646e−01
        5.3391098579e−01 7.4219815365e−01 1.0317414587e+00 1.4342402124e+00
        1.9937601321e+00 2.7715576720e+00 3.8527864038e+00 5.3558196617e+00
        7.4452101004e+00 1.0349704983e+01 1.4387289517e+01 2.0000000000e+01
t1
        1.0000000000e−04 1.3219411485e−04 1.7475284000e−04 2.3101297001e−04
        3.0538555088e−04 4.0370172586e−04 5.3366992312e−04 7.0548023107e−04
        9.3260334688e−04 1.2328467394e−03 1.6297508346e−03 2.1544346900e−03
        2.8480358684e−03 3.7649358068e−03 4.9770235643e−03 6.5793322466e−03
        8.6974900262e−03 1.1497569954e−02 1.5199110830e−02 2.0092330026e−02
        2.6560877829e−02 3.5111917342e−02 4.6415888336e−02 6.1359072734e−02
        8.1113083079e−02 1.0722672220e−01 1.4174741629e−01 1.8738174229e−01
        2.4770763560e−01 3.2745491629e−01 4.3287612811e−01 5.7223676594e−01
        7.5646332755e−01 1.0000000000e+00 1.3219411485e+00 1.7475284000e+00
        2.3101297001e+00 3.0538555088e+00 4.0370172586e+00 5.3366992312e+00
        7.0548023107e+00 9.3260334688e+00 1.2328467394e+01 1.6297508346e+01
        2.1544346900e+01 2.8480358684e+01 3.7649358068e+01 4.9770235643e+01
        6.5793322466e+01 8.6974900262e+01 1.1497569954e+02 1.5199110830e+02
        2.0092330026e+02 2.6560877829e+02 3.5111917342e+02 4.6415888336e+02
        6.1359072734e+02 8.1113083079e+02 1.0722672220e+03 1.4174741629e+03
        1.8738174229e+03 2.4770763560e+03 3.2745491629e+03 4.3287612811e+03
        5.7223676594e+03 7.5646332755e+03 1.0000000000e+04 1.3219411485e+04
        1.7475284000e+04 2.3101297001e+04 3.0538555088e+04 4.0370172586e+04
        5.3366992312e+04 7.0548023107e+04 9.3260334688e+04 1.2328467394e+05
        1.6297508346e+05 2.1544346900e+05 2.8480358684e+05 3.7649358068e+05
        4.9770235643e+05 6.5793322466e+05 8.6974900262e+05 1.1497569954e+06
        1.5199110830e+06 2.0092330026e+06 2.6560877829e+06 3.5111917342e+06
        4.6415888336e+06 6.1359072734e+06 8.1113083079e+06 1.0722672220e+07
        1.4174741629e+07 1.8738174229e+07 2.4770763560e+07 3.2745491629e+07
        4.3287612811e+07 5.7223676594e+07 7.5646332755e+07 1.0000000000e+08
m1
        92232  −3.0000000000e−08
        92234  −7.0000000000e−01
        92235  −9.0300000000e+01
        92236  −3.0000000000e−01
        92238  −8.7000000000e+00
```

```
esplt:n 2 0.1   2 0.05   2 0.025   3 0.01   4 0.005   5 0.025
prdmp j j -1
```

Listing A.7: Example Neutron Shield Input File

```
Title
1 1 -1.03 901 -1  903 -904 905 -906
2 1 -1.03 1 -2 903 -904 905 -906
3 1 -1.03 -3 2 903 -904 905 -906
4 1 -1.03 -4 3 903 -904 905 -906
5 1 -1.03 -5 4 903 -904 905 -906
6 1 -1.03 -6 5 903 -904 905 -906
7 1 -1.03 -7 6 903 -904 905 -906
8 1 -1.03 -8 7 903 -904 905 -906
9 1 -1.03 -9 8 903 -904 905 -906
10 1 -1.03 -10 9 903 -904 905 -906
11 1 -1.03 -11 10 903 -904 905 -906
12 1 -1.03 -12 11 903 -904 905 -906
13 1 -1.03 -13 12 903 -904 905 -906
14 1 -1.03 -14 13 903 -904 905 -906
15 1 -1.03 -15 14 903 -904 905 -906
16 1 -1.03 -16 15 903 -904 905 -906
17 1 -1.03 -17 16 903 -904 905 -906
18 1 -1.03 -18 17 903 -904 905 -906
19 1 -1.03 -19 18 903 -904 905 -906
20 1 -1.03 -20 19 903 -904 905 -906
21 1 -1.03 -21 20 903 -904 905 -906
22 1 -1.03 -22 21 903 -904 905 -906
23 1 -1.03 -23 22 903 -904 905 -906
24 1 -1.03 -24 23 903 -904 905 -906
25 1 -1.03 -25 24 903 -904 905 -906
26 1 -1.03 -26 25 903 -904 905 -906
27 1 -1.03 -27 26 903 -904 905 -906
28 1 -1.03 -28 27 903 -904 905 -906
29 1 -1.03 -29 28 903 -904 905 -906
30 1 -1.03 -30 29 903 -904 905 -906
31 1 -1.03 -31 30 903 -904 905 -906
32 1 -1.03 -32 31 903 -904 905 -906
33 1 -1.03 -33 32 903 -904 905 -906
34 1 -1.03 -34 33 903 -904 905 -906
35 1 -1.03 -35 34 903 -904 905 -906
36 1 -1.03 -36 35 903 -904 905 -906
37 1 -1.03 -37 36 903 -904 905 -906
38 1 -1.03 -38 37 903 -904 905 -906
39 1 -1.03 -39 38 903 -904 905 -906
40 1 -1.03 -40 39 903 -904 905 -906
41 1 -1.03 -41 40 903 -904 905 -906
42 1 -1.03 -42 41 903 -904 905 -906
43 1 -1.03 -43 42 903 -904 905 -906
44 1 -1.03 -44 43 903 -904 905 -906
45 1 -1.03 -45 44 903 -904 905 -906
46 1 -1.03 -46 45 903 -904 905 -906
47 1 -1.03 -47 46 903 -904 905 -906
48 1 -1.03 -48 47 903 -904 905 -906
49 1 -1.03 -49 48 903 -904 905 -906
50 1 -1.03 -50 49 903 -904 905 -906
51 1 -1.03 -51 50 903 -904 905 -906
52 1 -1.03 -52 51 903 -904 905 -906
53 1 -1.03 -53 52 903 -904 905 -906
54 1 -1.03 -54 53 903 -904 905 -906
55 1 -1.03 -55 54 903 -904 905 -906
56 1 -1.03 -56 55 903 -904 905 -906
57 1 -1.03 -57 56 903 -904 905 -906
58 1 -1.03 -58 57 903 -904 905 -906
59 1 -1.03 -59 58 903 -904 905 -906
60 1 -1.03 -60 59 903 -904 905 -906
61 1 -1.03 -61 60 903 -904 905 -906
62 1 -1.03 -62 61 903 -904 905 -906
63 1 -1.03 -63 62 903 -904 905 -906
64 1 -1.03 -64 63 903 -904 905 -906
65 1 -1.03 -65 64 903 -904 905 -906
66 1 -1.03 -66 65 903 -904 905 -906
67 1 -1.03 -67 66 903 -904 905 -906
68 1 -1.03 -68 67 903 -904 905 -906
```

69	1	-1.03	-69	68	903	-904	905	-906
70	1	-1.03	-70	69	903	-904	905	-906
71	1	-1.03	-71	70	903	-904	905	-906
72	1	-1.03	-72	71	903	-904	905	-906
73	1	-1.03	-73	72	903	-904	905	-906
74	1	-1.03	-74	73	903	-904	905	-906
75	1	-1.03	-75	74	903	-904	905	-906
76	1	-1.03	-76	75	903	-904	905	-906
77	1	-1.03	-77	76	903	-904	905	-906
78	1	-1.03	-78	77	903	-904	905	-906
79	1	-1.03	-79	78	903	-904	905	-906
80	1	-1.03	-80	79	903	-904	905	-906
81	1	-1.03	-81	80	903	-904	905	-906
82	1	-1.03	-82	81	903	-904	905	-906
83	1	-1.03	-83	82	903	-904	905	-906
84	1	-1.03	-84	83	903	-904	905	-906
85	1	-1.03	-85	84	903	-904	905	-906
86	1	-1.03	-86	85	903	-904	905	-906
87	1	-1.03	-87	86	903	-904	905	-906
88	1	-1.03	-88	87	903	-904	905	-906
89	1	-1.03	-89	88	903	-904	905	-906
90	1	-1.03	-90	89	903	-904	905	-906
91	1	-1.03	-91	90	903	-904	905	-906
92	1	-1.03	-92	91	903	-904	905	-906
93	1	-1.03	-93	92	903	-904	905	-906
94	1	-1.03	-94	93	903	-904	905	-906
95	1	-1.03	-95	94	903	-904	905	-906
96	1	-1.03	-96	95	903	-904	905	-906
97	1	-1.03	-97	96	903	-904	905	-906
98	1	-1.03	-98	97	903	-904	905	-906
99	1	-1.03	-99	98	903	-904	905	-906
100	1	-1.03	-100	99	903	-904	905	-906
101	1	-1.03	-101	100	903	-904	905	-906
102	1	-1.03	-102	101	903	-904	905	-906
103	1	-1.03	-103	102	903	-904	905	-906
104	1	-1.03	-104	103	903	-904	905	-906
105	1	-1.03	-105	104	903	-904	905	-906
106	1	-1.03	-106	105	903	-904	905	-906
107	1	-1.03	-107	106	903	-904	905	-906
108	1	-1.03	-108	107	903	-904	905	-906
109	1	-1.03	-109	108	903	-904	905	-906
110	1	-1.03	-110	109	903	-904	905	-906
111	1	-1.03	-111	110	903	-904	905	-906
112	1	-1.03	-112	111	903	-904	905	-906
113	1	-1.03	-113	112	903	-904	905	-906
114	1	-1.03	-114	113	903	-904	905	-906
115	1	-1.03	-115	114	903	-904	905	-906
116	1	-1.03	-116	115	903	-904	905	-906
117	1	-1.03	-117	116	903	-904	905	-906
118	1	-1.03	-118	117	903	-904	905	-906
119	1	-1.03	-119	118	903	-904	905	-906
120	1	-1.03	-120	119	903	-904	905	-906
121	1	-1.03	-121	120	903	-904	905	-906
122	1	-1.03	-122	121	903	-904	905	-906
123	1	-1.03	-123	122	903	-904	905	-906
124	1	-1.03	-124	123	903	-904	905	-906
125	1	-1.03	-125	124	903	-904	905	-906
126	1	-1.03	-126	125	903	-904	905	-906
127	1	-1.03	-127	126	903	-904	905	-906
128	1	-1.03	-128	127	903	-904	905	-906
129	1	-1.03	-129	128	903	-904	905	-906
130	1	-1.03	-130	129	903	-904	905	-906
131	1	-1.03	-131	130	903	-904	905	-906
132	1	-1.03	-132	131	903	-904	905	-906
133	1	-1.03	-133	132	903	-904	905	-906
134	1	-1.03	-134	133	903	-904	905	-906
135	1	-1.03	-135	134	903	-904	905	-906
136	1	-1.03	-136	135	903	-904	905	-906
137	1	-1.03	-137	136	903	-904	905	-906
138	1	-1.03	-138	137	903	-904	905	-906
139	1	-1.03	-139	138	903	-904	905	-906
140	1	-1.03	-140	139	903	-904	905	-906
141	1	-1.03	-141	140	903	-904	905	-906
142	1	-1.03	-142	141	903	-904	905	-906
143	1	-1.03	-143	142	903	-904	905	-906
144	1	-1.03	-144	143	903	-904	905	-906

```
145 1 −1.03 −145 144 903 −904 905 −906
146 1 −1.03 −146 145 903 −904 905 −906
147 1 −1.03 −147 146 903 −904 905 −906
148 1 −1.03 −148 147 903 −904 905 −906
149 1 −1.03 −149 148 903 −904 905 −906
150 1 −1.03 −150 149 903 −904 905 −906
151 1 −1.03 −151 150 903 −904 905 −906
152 1 −1.03 −152 151 903 −904 905 −906
153 1 −1.03 −153 152 903 −904 905 −906
154 1 −1.03 −154 153 903 −904 905 −906
155 1 −1.03 −155 154 903 −904 905 −906
156 1 −1.03 −156 155 903 −904 905 −906
157 1 −1.03 −157 156 903 −904 905 −906
158 1 −1.03 −158 157 903 −904 905 −906
159 1 −1.03 −159 158 903 −904 905 −906
160 1 −1.03 −160 159 903 −904 905 −906
161 1 −1.03 −161 160 903 −904 905 −906
162 1 −1.03 −162 161 903 −904 905 −906
163 1 −1.03 −163 162 903 −904 905 −906
164 1 −1.03 −164 163 903 −904 905 −906
165 1 −1.03 −165 164 903 −904 905 −906
166 1 −1.03 −166 165 903 −904 905 −906
167 1 −1.03 −167 166 903 −904 905 −906
168 1 −1.03 −168 167 903 −904 905 −906
169 1 −1.03 −169 168 903 −904 905 −906
170 1 −1.03 −170 169 903 −904 905 −906
171 1 −1.03 −171 170 903 −904 905 −906
172 1 −1.03 −172 171 903 −904 905 −906
173 1 −1.03 −173 172 903 −904 905 −906
174 1 −1.03 −174 173 903 −904 905 −906
175 1 −1.03 −175 174 903 −904 905 −906
176 1 −1.03 −176 175 903 −904 905 −906
177 1 −1.03 −177 176 903 −904 905 −906
178 1 −1.03 −178 177 903 −904 905 −906
179 1 −1.03 −179 178 903 −904 905 −906
180 1 −1.03 −180 179 903 −904 905 −906
181 1 −1.03 −181 180 903 −904 905 −906
182 1 −1.03 −182 181 903 −904 905 −906
183 1 −1.03 −183 182 903 −904 905 −906
184 1 −1.03 −184 183 903 −904 905 −906
185 1 −1.03 −185 184 903 −904 905 −906
186 1 −1.03 −186 185 903 −904 905 −906
187 1 −1.03 −187 186 903 −904 905 −906
188 1 −1.03 −188 187 903 −904 905 −906
189 1 −1.03 −189 188 903 −904 905 −906
190 1 −1.03 −190 189 903 −904 905 −906
191 1 −1.03 −191 190 903 −904 905 −906
192 1 −1.03 −192 191 903 −904 905 −906
193 1 −1.03 −193 192 903 −904 905 −906
194 1 −1.03 −194 193 903 −904 905 −906
195 1 −1.03 −195 194 903 −904 905 −906
196 1 −1.03 −196 195 903 −904 905 −906
197 1 −1.03 −197 196 903 −904 905 −906
198 1 −1.03 −198 197 903 −904 905 −906
199 1 −1.03 −199 198 903 −904 905 −906
201 0  (−901:199:−903:904:−905:906)

901 px 0
*903 py −1
*904 py 1
*905 pz −1
*906 pz 1
1 px 1.0000000000e−04
2 px 1.1000000000e−04
3 px 5.0514949495e−01
4 px 5.0515949495e−01
5 px 1.0101989899e+00
6 px 1.0102089899e+00
7 px 1.5152484848e+00
8 px 1.5152584848e+00
9 px 2.0202979798e+00
10 px 2.0203079798e+00
11 px 2.5253474747e+00
12 px 2.5253574747e+00
13 px 3.0303969697e+00
14 px 3.0304069697e+00
```

```
15  px  3.5354464646e+00
16  px  3.5354564646e+00
17  px  4.0404959596e+00
18  px  4.0405059596e+00
19  px  4.5455454545e+00
20  px  4.5455554545e+00
21  px  5.0505949495e+00
22  px  5.0506049495e+00
23  px  5.5556444444e+00
24  px  5.5556544444e+00
25  px  6.0606939394e+00
26  px  6.0607039394e+00
27  px  6.5657434343e+00
28  px  6.5657534343e+00
29  px  7.0707929293e+00
30  px  7.0708029293e+00
31  px  7.5758424242e+00
32  px  7.5758524242e+00
33  px  8.0808919192e+00
34  px  8.0809019192e+00
35  px  8.5859414141e+00
36  px  8.5859514141e+00
37  px  9.0909909091e+00
38  px  9.0910009091e+00
39  px  9.5960404040e+00
40  px  9.5960504040e+00
41  px  1.0101089899e+01
42  px  1.0101099899e+01
43  px  1.0606139394e+01
44  px  1.0606149394e+01
45  px  1.1111188889e+01
46  px  1.1111198889e+01
47  px  1.1616238384e+01
48  px  1.1616248384e+01
49  px  1.2121287879e+01
50  px  1.2121297879e+01
51  px  1.2626337374e+01
52  px  1.2626347374e+01
53  px  1.3131386869e+01
54  px  1.3131396869e+01
55  px  1.3636436364e+01
56  px  1.3636446364e+01
57  px  1.4141485859e+01
58  px  1.4141495859e+01
59  px  1.4646535354e+01
60  px  1.4646545354e+01
61  px  1.5151584848e+01
62  px  1.5151594848e+01
63  px  1.5656634343e+01
64  px  1.5656644343e+01
65  px  1.6161683838e+01
66  px  1.6161693838e+01
67  px  1.6666733333e+01
68  px  1.6666743333e+01
69  px  1.7171782828e+01
70  px  1.7171792828e+01
71  px  1.7676832323e+01
72  px  1.7676842323e+01
73  px  1.8181881818e+01
74  px  1.8181891818e+01
75  px  1.8686931313e+01
76  px  1.8686941313e+01
77  px  1.9191980808e+01
78  px  1.9191990808e+01
79  px  1.9697030303e+01
80  px  1.9697040303e+01
81  px  2.0202079798e+01
82  px  2.0202089798e+01
83  px  2.0707129293e+01
84  px  2.0707139293e+01
85  px  2.1212178788e+01
86  px  2.1212188788e+01
87  px  2.1717228283e+01
88  px  2.1717238283e+01
89  px  2.2222277778e+01
90  px  2.2222287778e+01
```

```
91  px  2.2727327273e+01
92  px  2.2727337273e+01
93  px  2.3232376768e+01
94  px  2.3232386768e+01
95  px  2.3737426263e+01
96  px  2.3737436263e+01
97  px  2.4242475758e+01
98  px  2.4242485758e+01
99  px  2.4747525253e+01
100  px  2.4747535253e+01
101  px  2.5252574747e+01
102  px  2.5252584747e+01
103  px  2.5757624242e+01
104  px  2.5757634242e+01
105  px  2.6262673737e+01
106  px  2.6262683737e+01
107  px  2.6767723232e+01
108  px  2.6767733232e+01
109  px  2.7272772727e+01
110  px  2.7272782727e+01
111  px  2.7777822222e+01
112  px  2.7777832222e+01
113  px  2.8282871717e+01
114  px  2.8282881717e+01
115  px  2.8787921212e+01
116  px  2.8787931212e+01
117  px  2.9292970707e+01
118  px  2.9292980707e+01
119  px  2.9798020202e+01
120  px  2.9798030202e+01
121  px  3.0303069697e+01
122  px  3.0303079697e+01
123  px  3.0808119192e+01
124  px  3.0808129192e+01
125  px  3.1313168687e+01
126  px  3.1313178687e+01
127  px  3.1818218182e+01
128  px  3.1818228182e+01
129  px  3.2323267677e+01
130  px  3.2323277677e+01
131  px  3.2828317172e+01
132  px  3.2828327172e+01
133  px  3.3333366667e+01
134  px  3.3333376667e+01
135  px  3.3838416162e+01
136  px  3.3838426162e+01
137  px  3.4343465657e+01
138  px  3.4343475657e+01
139  px  3.4848515152e+01
140  px  3.4848525152e+01
141  px  3.5353564646e+01
142  px  3.5353574646e+01
143  px  3.5858614141e+01
144  px  3.5858624141e+01
145  px  3.6363663636e+01
146  px  3.6363673636e+01
147  px  3.6868713131e+01
148  px  3.6868723131e+01
149  px  3.7373762626e+01
150  px  3.7373772626e+01
151  px  3.7878812121e+01
152  px  3.7878822121e+01
153  px  3.8383861616e+01
154  px  3.8383871616e+01
155  px  3.8888911111e+01
156  px  3.8888921111e+01
157  px  3.9393960606e+01
158  px  3.9393970606e+01
159  px  3.9899010101e+01
160  px  3.9899020101e+01
161  px  4.0404059596e+01
162  px  4.0404069596e+01
163  px  4.0909109091e+01
164  px  4.0909119091e+01
165  px  4.1414158586e+01
166  px  4.1414168586e+01
```

```
167 px 4.1919208081e+01
168 px 4.1919218081e+01
169 px 4.2424257576e+01
170 px 4.2424267576e+01
171 px 4.2929307071e+01
172 px 4.2929317071e+01
173 px 4.3434356566e+01
174 px 4.3434366566e+01
175 px 4.3939406061e+01
176 px 4.3939416061e+01
177 px 4.4444455556e+01
178 px 4.4444465556e+01
179 px 4.4949505051e+01
180 px 4.4949515051e+01
181 px 4.5454554545e+01
182 px 4.5454564545e+01
183 px 4.5959604040e+01
184 px 4.5959614040e+01
185 px 4.6464653535e+01
186 px 4.6464663535e+01
187 px 4.6969703030e+01
188 px 4.6969713030e+01
189 px 4.7474752525e+01
190 px 4.7474762525e+01
191 px 4.7979802020e+01
192 px 4.7979812020e+01
193 px 4.8484851515e+01
194 px 4.8484861515e+01
195 px 4.8989901010e+01
196 px 4.8989911010e+01
197 px 4.9494950505e+01
198 px 4.9494960505e+01
199 px 5.0000000000e+01

phys:n 20.1
mode n
sdef sur=901 par=n x=0 y=d2 z=d3 vec=1 0 0 erg=d1 dir=d4
si1 1.0000000000e-10 1.3901159058e-10
sp1 0 1
si2 -1 1
sp2  0 1
si3 -1 1
sp3  0 1
si4
       0.0000000000e+00 5.2631578947e-02 1.0526315789e-01 1.5789473684e-01
       2.1052631579e-01 2.6315789474e-01 3.1578947368e-01 3.6842105263e-01
       4.2105263158e-01 4.7368421053e-01 5.2631578947e-01 5.7894736842e-01
       6.3157894737e-01 6.8421052632e-01 7.3684210526e-01 7.8947368421e-01
       8.4210526316e-01 8.9473684211e-01 9.4736842105e-01 1.0000000000e+00
sp4
       0 5.2631578947e-02 5.2631578947e-02 5.2631578947e-02
       5.2631578947e-02 5.2631578947e-02 5.2631578947e-02 5.2631578947e-02
       5.2631578947e-02 5.2631578947e-02 5.2631578947e-02 5.2631578947e-02
       5.2631578947e-02 5.2631578947e-02 5.2631578947e-02 5.2631578947e-02
       5.2631578947e-02 5.2631578947e-02 5.2631578947e-02 5.2631578947e-02
f1:n 1
sf1 2
f11:n 3
sf11 4
f21:n 5
sf21 6
f31:n 7
sf31 8
f41:n 9
sf41 10
f51:n 11
sf51 12
f61:n 13
sf61 14
f71:n 15
sf71 16
f81:n 17
sf81 18
f91:n 19
sf91 20
f101:n 21
```

```
sf101 22
f111:n 23
sf111 24
f121:n 25
sf121 26
f131:n 27
sf131 28
f141:n 29
sf141 30
f151:n 31
sf151 32
f161:n 33
sf161 34
f171:n 35
sf171 36
f181:n 37
sf181 38
f191:n 39
sf191 40
f201:n 41
sf201 42
f211:n 43
sf211 44
f221:n 45
sf221 46
f231:n 47
sf231 48
f241:n 49
sf241 50
f251:n 51
sf251 52
f261:n 53
sf261 54
f271:n 55
sf271 56
f281:n 57
sf281 58
f291:n 59
sf291 60
f301:n 61
sf301 62
f311:n 63
sf311 64
f321:n 65
sf321 66
f331:n 67
sf331 68
f341:n 69
sf341 70
f351:n 71
sf351 72
f361:n 73
sf361 74
f371:n 75
sf371 76
f381:n 77
sf381 78
f391:n 79
sf391 80
f401:n 81
sf401 82
f411:n 83
sf411 84
f421:n 85
sf421 86
f431:n 87
sf431 88
f441:n 89
sf441 90
f451:n 91
sf451 92
f461:n 93
sf461 94
f471:n 95
sf471 96
f481:n 97
```

```
sf481 98
f491:n 99
sf491 100
f501:n 101
sf501 102
f511:n 103
sf511 104
f521:n 105
sf521 106
f531:n 107
sf531 108
f541:n 109
sf541 110
f551:n 111
sf551 112
f561:n 113
sf561 114
f571:n 115
sf571 116
f581:n 117
sf581 118
f591:n 119
sf591 120
f601:n 121
sf601 122
f611:n 123
sf611 124
f621:n 125
sf621 126
f631:n 127
sf631 128
f641:n 129
sf641 130
f651:n 131
sf651 132
f661:n 133
sf661 134
f671:n 135
sf671 136
f681:n 137
sf681 138
f691:n 139
sf691 140
f701:n 141
sf701 142
f711:n 143
sf711 144
f721:n 145
sf721 146
f731:n 147
sf731 148
f741:n 149
sf741 150
f751:n 151
sf751 152
f761:n 153
sf761 154
f771:n 155
sf771 156
f781:n 157
sf781 158
f791:n 159
sf791 160
f801:n 161
sf801 162
f811:n 163
sf811 164
f821:n 165
sf821 166
f831:n 167
sf831 168
f841:n 169
sf841 170
f851:n 171
sf851 172
f861:n 173
```

```
sf861 174
f871:n 175
sf871 176
f881:n 177
sf881 178
f891:n 179
sf891 180
f901:n 181
sf901 182
f911:n 183
sf911 184
f921:n 185
sf921 186
f931:n 187
sf931 188
f941:n 189
sf941 190
f951:n 191
sf951 192
f961:n 193
sf961 194
f971:n 195
sf971 196
f981:n 197
sf981 198
f991:n 199
tf991 j j j j j j j j
t0
     0.0001 0.000132194 0.000174753 0.000231013 0.000305386
     0.000403702 0.00053367 0.00070548 0.000932603 0.00123285
     0.00162975 0.00215443 0.00284804 0.00376494 0.00497702
     0.00657933 0.00869749 0.0114976 0.0151991 0.0200923
     0.0265609 0.0351119 0.0464159 0.0613591 0.0811131
     0.107227 0.141747 0.187382 0.247708 0.327455 0.432876
     0.572237 0.756463 1 1.32194 1.74753 2.31013 3.05386
     4.03702 5.3367 7.0548 9.32603 12.3285 16.2975 21.5443
     28.4804 37.6494 49.7702 65.7933 86.9749 114.976 151.991
     200.923 265.609 351.119 464.159 613.591 811.131 1072.27
     1417.47 1873.82 2477.08 3274.55 4328.76 5722.37 7564.63
     10000 13219.4 17475.3 23101.3 30538.6 40370.2 53367 70548
     93260.3 123285 162975 215443 284804 376494 497702 657933
     869749 1.14976e+06 1.51991e+06 2.00923e+06 2.65609e+06
     3.51119e+06 4.64159e+06 6.13591e+06 8.11131e+06 1.07227e+07
     1.41747e+07 1.87382e+07 2.47708e+07 3.27455e+07 4.32876e+07
     5.72237e+07 7.56463e+07 1e+08
e0
      1.0000000000e-10  1.3901159058e-10  1.9324222317e-10  2.6862908810e-10
      3.7342556814e-10  5.1910482191e-10  7.2161586974e-10  1.0031296984e-09
      1.3944665494e-09  1.9384701305e-09  2.6946981613e-09  3.7459427755e-09
      5.2072946346e-09  7.2387430979e-09  1.0062691919e-08  1.3988308092e-08
      1.9445369574e-08  2.7031317539e-08  3.7576664467e-08  5.2235918964e-08
      7.2613981808e-08  1.0094185110e-07  1.4032087277e-07  1.9506227716e-07
      2.7115917411e-07  3.7694268095e-07  5.2399401637e-07  7.2841241672e-07
      1.0125776865e-06  1.4076003479e-06  1.9567276326e-06  2.7200782055e-06
      3.7812239786e-06  5.2563395962e-06  7.3069212791e-06  1.0157467493e-05
      1.4120057125e-05  1.9628516000e-05  2.7285912300e-05  3.7930580693e-05
      5.2727903539e-05  7.3297897391e-05  1.0189257303e-04  1.4164248645e-04
      1.9689947336e-04  2.7371308977e-04  3.8049291972e-04  5.2892925976e-04
      7.3527297705e-04  1.0221146605e-03  1.4208578472e-03  1.9751570933e-03
      2.7456972919e-03  3.8168374781e-03  5.3058464883e-03  7.3757415973e-03
      1.0253135712e-02  1.4253047038e-02  1.9813387394e-02  2.7542904964e-02
      3.8287830284e-02  5.3224521877e-02  7.3988254442e-02  1.0285224934e-01
      1.4297654776e-01  1.9875397321e-01  2.7629105951e-01  3.8407659646e-01
      5.3391098579e-01  7.4219815365e-01  1.0317414587e+00  1.4342402124e+00
      1.9937601321e+00  2.7715576720e+00  3.8527864038e+00  5.3558196617e+00
      7.4452101004e+00  1.0349704983e+01  1.4387289517e+01  2.0000000000e+01
m1
      001001 -0.12934
      006012 -0.77066
      005010 -0.0199
      005011 -0.0801
mt1 poly.60t
prdmp j j 1 3 0
imp:n 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1   0
ctme 120
nps 2.1e9
dbcn j j j j j j j 10 j j j j 1529175
wwg 991 0 0 j j j j 0
wwge:n 1e-10 10log 20
mesh geom=rec ref=0 0 0 origin=0 -1 -1
      imesh=0.0001 100i 50
      jmesh=2
      kmesh=2
```

Listing A.8: Example Neutron Detector Input File

```
helium 3 detector energy 1e-10 to 1.39012e-10
1 1 -0.0007425 -1 imp:n=1
2 2 -0.93 -2 1 imp:n=1
3 3 -7.9 -3 2 1001 imp:n=1
4 0  (-3 2 -1001):(-4 3) imp:n=1
11 1 -0.0007425 -11 imp:n=1
12 2 -0.93 -12 11 imp:n=1
13 3 -7.9 -13 12 1001 imp:n=1
14 0  (-13 12 -1001):(-14 13) imp:n=1
21 1 -0.0007425 -21 imp:n=1
22 2 -0.93 -22 21 imp:n=1
23 3 -7.9 -23 22 1001 imp:n=1
24 0  (-23 22 -1001):(-24 23) imp:n=1
31 1 -0.0007425 -31 imp:n=1
32 2 -0.93 -32 31 imp:n=1
33 3 -7.9 -33 32 1001 imp:n=1
34 0  (-33 32 -1001):(-34 33) imp:n=1
41 1 -0.0007425 -41 imp:n=1
42 2 -0.93 -42 41 imp:n=1
43 3 -7.9 -43 42 1001 imp:n=1
44 0  (-43 42 -1001):(-44 43) imp:n=1
51 1 -0.0007425 -51 imp:n=1
52 2 -0.93 -52 51 imp:n=1
53 3 -7.9 -53 52 1001 imp:n=1
54 0  (-53 52 -1001):(-54 53) imp:n=1
61 1 -0.0007425 -61 imp:n=1
62 2 -0.93 -62 61 imp:n=1
63 3 -7.9 -63 62 1001 imp:n=1
64 0  (-63 62 -1001):(-64 63) imp:n=1
71 1 -0.0007425 -71 imp:n=1
72 2 -0.93 -72 71 imp:n=1
73 3 -7.9 -73 72 1001 imp:n=1
74 0  (-73 72 -1001):(-74 73) imp:n=1
81 1 -0.0007425 -81 imp:n=1
82 2 -0.93 -82 81 imp:n=1
83 3 -7.9 -83 82 1001 imp:n=1
84 0  (-83 82 -1001):(-84 83) imp:n=1
91 1 -0.0007425 -91 imp:n=1
92 2 -0.93 -92 91 imp:n=1
93 3 -7.9 -93 92 1001 imp:n=1
94 0  (-93 92 -1001):(-94 93) imp:n=1
101 1 -0.0007425 -101 imp:n=1
102 2 -0.93 -102 101 imp:n=1
103 3 -7.9 -103 102 1001 imp:n=1
104 0  (-103 102 -1001):(-104 103) imp:n=1
111 1 -0.0007425 -111 imp:n=1
112 2 -0.93 -112 111 imp:n=1
113 3 -7.9 -113 112 1001 imp:n=1
114 0  (-113 112 -1001):(-114 113) imp:n=1
121 1 -0.0007425 -121 imp:n=1
122 2 -0.93 -122 121 imp:n=1
123 3 -7.9 -123 122 1001 imp:n=1
124 0  (-123 122 -1001):(-124 123) imp:n=1
131 1 -0.0007425 -131 imp:n=1
132 2 -0.93 -132 131 imp:n=1
133 3 -7.9 -133 132 1001 imp:n=1
134 0  (-133 132 -1001):(-134 133) imp:n=1
```

```
141 1 -0.0007425 -141 imp:n=1
142 2 -0.93 -142 141 imp:n=1
143 3 -7.9 -143 142 1001 imp:n=1
144 0 (-143 142 -1001):(-144 143) imp:n=1
151 1 -0.0007425 -151 imp:n=1
152 2 -0.93 -152 151 imp:n=1
153 3 -7.9 -153 152 1001 imp:n=1
154 0 (-153 152 -1001):(-154 153) imp:n=1
161 1 -0.0007425 -161 imp:n=1
162 2 -0.93 -162 161 imp:n=1
163 3 -7.9 -163 162 1001 imp:n=1
164 0 (-163 162 -1001):(-164 163) imp:n=1
171 1 -0.0007425 -171 imp:n=1
172 2 -0.93 -172 171 imp:n=1
173 3 -7.9 -173 172 1001 imp:n=1
174 0 (-173 172 -1001):(-174 173) imp:n=1
181 1 -0.0007425 -181 imp:n=1
182 2 -0.93 -182 181 imp:n=1
183 3 -7.9 -183 182 1001 imp:n=1
184 0 (-183 182 -1001):(-184 183) imp:n=1
191 1 -0.0007425 -191 imp:n=1
192 2 -0.93 -192 191 imp:n=1
193 3 -7.9 -193 192 1001 imp:n=1
194 0 (-193 192 -1001):(-194 193) imp:n=1
201 1 -0.0007425 -201 imp:n=1
202 2 -0.93 -202 201 imp:n=1
203 3 -7.9 -203 202 1001 imp:n=1
204 0 (-203 202 -1001):(-204 203) imp:n=1
211 1 -0.0007425 -211 imp:n=1
212 2 -0.93 -212 211 imp:n=1
213 3 -7.9 -213 212 1001 imp:n=1
214 0 (-213 212 -1001):(-214 213) imp:n=1
221 1 -0.0007425 -221 imp:n=1
222 2 -0.93 -222 221 imp:n=1
223 3 -7.9 -223 222 1001 imp:n=1
224 0 (-223 222 -1001):(-224 223) imp:n=1
231 1 -0.0007425 -231 imp:n=1
232 2 -0.93 -232 231 imp:n=1
233 3 -7.9 -233 232 1001 imp:n=1
234 0 (-233 232 -1001):(-234 233) imp:n=1
241 1 -0.0007425 -241 imp:n=1
242 2 -0.93 -242 241 imp:n=1
243 3 -7.9 -243 242 1001 imp:n=1
244 0 (-243 242 -1001):(-244 243) imp:n=1
251 1 -0.0007425 -251 imp:n=1
252 2 -0.93 -252 251 imp:n=1
253 3 -7.9 -253 252 1001 imp:n=1
254 0 (-253 252 -1001):(-254 253) imp:n=1
261 1 -0.0007425 -261 imp:n=1
262 2 -0.93 -262 261 imp:n=1
263 3 -7.9 -263 262 1001 imp:n=1
264 0 (-263 262 -1001):(-264 263) imp:n=1
271 1 -0.0007425 -271 imp:n=1
272 2 -0.93 -272 271 imp:n=1
273 3 -7.9 -273 272 1001 imp:n=1
274 0 (-273 272 -1001):(-274 273) imp:n=1
281 1 -0.0007425 -281 imp:n=1
282 2 -0.93 -282 281 imp:n=1
283 3 -7.9 -283 282 1001 imp:n=1
284 0 (-283 282 -1001):(-284 283) imp:n=1
291 1 -0.0007425 -291 imp:n=1
292 2 -0.93 -292 291 imp:n=1
293 3 -7.9 -293 292 1001 imp:n=1
294 0 (-293 292 -1001):(-294 293) imp:n=1
301 1 -0.0007425 -301 imp:n=1
302 2 -0.93 -302 301 imp:n=1
303 3 -7.9 -303 302 1001 imp:n=1
304 0 (-303 302 -1001):(-304 303) imp:n=1
311 1 -0.0007425 -311 imp:n=1
312 2 -0.93 -312 311 imp:n=1
313 3 -7.9 -313 312 1001 imp:n=1
314 0 (-313 312 -1001):(-314 313) imp:n=1
321 1 -0.0007425 -321 imp:n=1
322 2 -0.93 -322 321 imp:n=1
323 3 -7.9 -323 322 1001 imp:n=1
324 0 (-323 322 -1001):(-324 323) imp:n=1
```

```
331 1 -0.0007425 -331 imp:n=1
332 2 -0.93 -332 331 imp:n=1
333 3 -7.9 -333 332 1001 imp:n=1
334 0 (-333 332 -1001):(-334 333) imp:n=1
341 1 -0.0007425 -341 imp:n=1
342 2 -0.93 -342 341 imp:n=1
343 3 -7.9 -343 342 1001 imp:n=1
344 0 (-343 342 -1001):(-344 343) imp:n=1
351 1 -0.0007425 -351 imp:n=1
352 2 -0.93 -352 351 imp:n=1
353 3 -7.9 -353 352 1001 imp:n=1
354 0 (-353 352 -1001):(-354 353) imp:n=1
361 1 -0.0007425 -361 imp:n=1
362 2 -0.93 -362 361 imp:n=1
363 3 -7.9 -363 362 1001 imp:n=1
364 0 (-363 362 -1001):(-364 363) imp:n=1
371 1 -0.0007425 -371 imp:n=1
372 2 -0.93 -372 371 imp:n=1
373 3 -7.9 -373 372 1001 imp:n=1
374 0 (-373 372 -1001):(-374 373) imp:n=1
381 1 -0.0007425 -381 imp:n=1
382 2 -0.93 -382 381 imp:n=1
383 3 -7.9 -383 382 1001 imp:n=1
384 0 (-383 382 -1001):(-384 383) imp:n=1
391 1 -0.0007425 -391 imp:n=1
392 2 -0.93 -392 391 imp:n=1
393 3 -7.9 -393 392 1001 imp:n=1
394 0 (-393 392 -1001):(-394 393) imp:n=1
401 1 -0.0007425 -401 imp:n=1
402 2 -0.93 -402 401 imp:n=1
403 3 -7.9 -403 402 1001 imp:n=1
404 0 (-403 402 -1001):(-404 403) imp:n=1
411 1 -0.0007425 -411 imp:n=1
412 2 -0.93 -412 411 imp:n=1
413 3 -7.9 -413 412 1001 imp:n=1
414 0 (-413 412 -1001):(-414 413) imp:n=1
421 1 -0.0007425 -421 imp:n=1
422 2 -0.93 -422 421 imp:n=1
423 3 -7.9 -423 422 1001 imp:n=1
424 0 (-423 422 -1001):(-424 423) imp:n=1
431 1 -0.0007425 -431 imp:n=1
432 2 -0.93 -432 431 imp:n=1
433 3 -7.9 -433 432 1001 imp:n=1
434 0 (-433 432 -1001):(-434 433) imp:n=1
441 1 -0.0007425 -441 imp:n=1
442 2 -0.93 -442 441 imp:n=1
443 3 -7.9 -443 442 1001 imp:n=1
444 0 (-443 442 -1001):(-444 443) imp:n=1
451 1 -0.0007425 -451 imp:n=1
452 2 -0.93 -452 451 imp:n=1
453 3 -7.9 -453 452 1001 imp:n=1
454 0 (-453 452 -1001):(-454 453) imp:n=1
461 1 -0.0007425 -461 imp:n=1
462 2 -0.93 -462 461 imp:n=1
463 3 -7.9 -463 462 1001 imp:n=1
464 0 (-463 462 -1001):(-464 463) imp:n=1
471 1 -0.0007425 -471 imp:n=1
472 2 -0.93 -472 471 imp:n=1
473 3 -7.9 -473 472 1001 imp:n=1
474 0 (-473 472 -1001):(-474 473) imp:n=1
481 1 -0.0007425 -481 imp:n=1
482 2 -0.93 -482 481 imp:n=1
483 3 -7.9 -483 482 1001 imp:n=1
484 0 (-483 482 -1001):(-484 483) imp:n=1
491 1 -0.0007425 -491 imp:n=1
492 2 -0.93 -492 491 imp:n=1
493 3 -7.9 -493 492 1001 imp:n=1
494 0 (-493 492 -1001):(-494 493) imp:n=1
501 1 -0.0007425 -501 imp:n=1
502 2 -0.93 -502 501 imp:n=1
503 3 -7.9 -503 502 1001 imp:n=1
504 0 (-503 502 -1001):(-504 503) imp:n=1
511 1 -0.0007425 -511 imp:n=1
512 2 -0.93 -512 511 imp:n=1
513 3 -7.9 -513 512 1001 imp:n=1
514 0 (-513 512 -1001):(-514 513) imp:n=1
```

```
521 1 −0.0007425 −521 imp:n=1
522 2 −0.93 −522 521 imp:n=1
523 3 −7.9 −523 522 1001 imp:n=1
524 0  (−523 522 −1001):(−524 523) imp:n=1
531 1 −0.0007425 −531 imp:n=1
532 2 −0.93 −532 531 imp:n=1
533 3 −7.9 −533 532 1001 imp:n=1
534 0  (−533 532 −1001):(−534 533) imp:n=1
541 1 −0.0007425 −541 imp:n=1
542 2 −0.93 −542 541 imp:n=1
543 3 −7.9 −543 542 1001 imp:n=1
544 0  (−543 542 −1001):(−544 543) imp:n=1
551 1 −0.0007425 −551 imp:n=1
552 2 −0.93 −552 551 imp:n=1
553 3 −7.9 −553 552 1001 imp:n=1
554 0  (−553 552 −1001):(−554 553) imp:n=1
561 1 −0.0007425 −561 imp:n=1
562 2 −0.93 −562 561 imp:n=1
563 3 −7.9 −563 562 1001 imp:n=1
564 0  (−563 562 −1001):(−564 563) imp:n=1
571 1 −0.0007425 −571 imp:n=1
572 2 −0.93 −572 571 imp:n=1
573 3 −7.9 −573 572 1001 imp:n=1
574 0  (−573 572 −1001):(−574 573) imp:n=1
581 1 −0.0007425 −581 imp:n=1
582 2 −0.93 −582 581 imp:n=1
583 3 −7.9 −583 582 1001 imp:n=1
584 0  (−583 582 −1001):(−584 583) imp:n=1
591 1 −0.0007425 −591 imp:n=1
592 2 −0.93 −592 591 imp:n=1
593 3 −7.9 −593 592 1001 imp:n=1
594 0  (−593 592 −1001):(−594 593) imp:n=1
601 1 −0.0007425 −601 imp:n=1
602 2 −0.93 −602 601 imp:n=1
603 3 −7.9 −603 602 1001 imp:n=1
604 0  (−603 602 −1001):(−604 603) imp:n=1
611 1 −0.0007425 −611 imp:n=1
612 2 −0.93 −612 611 imp:n=1
613 3 −7.9 −613 612 1001 imp:n=1
614 0  (−613 612 −1001):(−614 613) imp:n=1
621 1 −0.0007425 −621 imp:n=1
622 2 −0.93 −622 621 imp:n=1
623 3 −7.9 −623 622 1001 imp:n=1
624 0  (−623 622 −1001):(−624 623) imp:n=1
631 1 −0.0007425 −631 imp:n=1
632 2 −0.93 −632 631 imp:n=1
633 3 −7.9 −633 632 1001 imp:n=1
634 0  (−633 632 −1001):(−634 633) imp:n=1
9999 0 −9999 4 14 24 34 44 54 64 74 84 94 104 114 124
        134 144 154 164 174 184 194 204 214 224 234 244
        254 264 274 284 294 304 314 324 334 344 354 364
        374 384 394 404 414 424 434 444 454 464 474 484
        494 504 514 524 534 544 554 564 574 584 594 604
        614 624 634
         imp:n=0
10000 0 9999 imp:n=0

1001 py 0
1 rcc 20 0 0 0 0 10 2.495
2 rcc 20 0 0 0 0 10 2.495
3 rcc 20 0 0 0 0 10 2.495
4 rpp 1 39 −20 20 0 200
11 rcc 60 0 0 0 0 10 2.495
12 rcc 60 0 0 0 0 10 2.495
13 rcc 60 0 0 0 0 10 3.495
14 rpp 41 79 −20 20 0 200
21 rcc 100 0 0 0 0 10 2.495
22 rcc 100 0 0 0 0 10 2.495
23 rcc 100 0 0 0 0 10 5.495
24 rpp 81 119 −20 20 0 200
31 rcc 140 0 0 0 0 10 2.495
32 rcc 140 0 0 0 0 10 2.495
33 rcc 140 0 0 0 0 10 7.495
34 rpp 121 159 −20 20 0 200
41 rcc 180 0 0 0 0 10 2.495
42 rcc 180 0 0 0 0 10 3.495
```

```
43 rcc 180 0 0 0 0 10 3.495
44 rpp 161 199 -20 20 0 200
51 rcc 220 0 0 0 0 10 2.495
52 rcc 220 0 0 0 0 10 3.495
53 rcc 220 0 0 0 0 10 4.495
54 rpp 201 239 -20 20 0 200
61 rcc 260 0 0 0 0 10 2.495
62 rcc 260 0 0 0 0 10 3.495
63 rcc 260 0 0 0 0 10 6.495
64 rpp 241 279 -20 20 0 200
71 rcc 300 0 0 0 0 10 2.495
72 rcc 300 0 0 0 0 10 3.495
73 rcc 300 0 0 0 0 10 8.495
74 rpp 281 319 -20 20 0 200
81 rcc 340 0 0 0 0 10 2.495
82 rcc 340 0 0 0 0 10 7.495
83 rcc 340 0 0 0 0 10 7.495
84 rpp 321 359 -20 20 0 200
91 rcc 380 0 0 0 0 10 2.495
92 rcc 380 0 0 0 0 10 7.495
93 rcc 380 0 0 0 0 10 8.495
94 rpp 361 399 -20 20 0 200
101 rcc 420 0 0 0 0 10 2.495
102 rcc 420 0 0 0 0 10 7.495
103 rcc 420 0 0 0 0 10 10.495
104 rpp 401 439 -20 20 0 200
111 rcc 460 0 0 0 0 10 2.495
112 rcc 460 0 0 0 0 10 7.495
113 rcc 460 0 0 0 0 10 12.495
114 rpp 441 479 -20 20 0 200
121 rcc 500 0 0 0 0 10 2.495
122 rcc 500 0 0 0 0 10 12.495
123 rcc 500 0 0 0 0 10 12.495
124 rpp 481 519 -20 20 0 200
131 rcc 540 0 0 0 0 10 2.495
132 rcc 540 0 0 0 0 10 12.495
133 rcc 540 0 0 0 0 10 13.495
134 rpp 521 559 -20 20 0 200
141 rcc 580 0 0 0 0 10 2.495
142 rcc 580 0 0 0 0 10 12.495
143 rcc 580 0 0 0 0 10 15.495
144 rpp 561 599 -20 20 0 200
151 rcc 620 0 0 0 0 10 2.495
152 rcc 620 0 0 0 0 10 12.495
153 rcc 620 0 0 0 0 10 17.495
154 rpp 601 639 -20 20 0 200
161 rcc 660 0 0 0 0 50 2.495
162 rcc 660 0 0 0 0 50 2.495
163 rcc 660 0 0 0 0 50 2.495
164 rpp 641 679 -20 20 0 200
171 rcc 700 0 0 0 0 50 2.495
172 rcc 700 0 0 0 0 50 2.495
173 rcc 700 0 0 0 0 50 3.495
174 rpp 681 719 -20 20 0 200
181 rcc 740 0 0 0 0 50 2.495
182 rcc 740 0 0 0 0 50 2.495
183 rcc 740 0 0 0 0 50 5.495
184 rpp 721 759 -20 20 0 200
191 rcc 780 0 0 0 0 50 2.495
192 rcc 780 0 0 0 0 50 2.495
193 rcc 780 0 0 0 0 50 7.495
194 rpp 761 799 -20 20 0 200
201 rcc 820 0 0 0 0 50 2.495
202 rcc 820 0 0 0 0 50 3.495
203 rcc 820 0 0 0 0 50 3.495
204 rpp 801 839 -20 20 0 200
211 rcc 860 0 0 0 0 50 2.495
212 rcc 860 0 0 0 0 50 3.495
213 rcc 860 0 0 0 0 50 4.495
214 rpp 841 879 -20 20 0 200
221 rcc 900 0 0 0 0 50 2.495
222 rcc 900 0 0 0 0 50 3.495
223 rcc 900 0 0 0 0 50 6.495
224 rpp 881 919 -20 20 0 200
231 rcc 940 0 0 0 0 50 2.495
232 rcc 940 0 0 0 0 50 3.495
```

```
233  rcc  940  0  0  0  0  50  8.495
234  rpp  921  959  −20  20  0  200
241  rcc  980  0  0  0  0  50  2.495
242  rcc  980  0  0  0  0  50  7.495
243  rcc  980  0  0  0  0  50  7.495
244  rpp  961  999  −20  20  0  200
251  rcc  1020  0  0  0  0  50  2.495
252  rcc  1020  0  0  0  0  50  7.495
253  rcc  1020  0  0  0  0  50  8.495
254  rpp  1001  1039  −20  20  0  200
261  rcc  1060  0  0  0  0  50  2.495
262  rcc  1060  0  0  0  0  50  7.495
263  rcc  1060  0  0  0  0  50  10.495
264  rpp  1041  1079  −20  20  0  200
271  rcc  1100  0  0  0  0  50  2.495
272  rcc  1100  0  0  0  0  50  7.495
273  rcc  1100  0  0  0  0  50  12.495
274  rpp  1081  1119  −20  20  0  200
281  rcc  1140  0  0  0  0  50  2.495
282  rcc  1140  0  0  0  0  50  12.495
283  rcc  1140  0  0  0  0  50  12.495
284  rpp  1121  1159  −20  20  0  200
291  rcc  1180  0  0  0  0  50  2.495
292  rcc  1180  0  0  0  0  50  12.495
293  rcc  1180  0  0  0  0  50  13.495
294  rpp  1161  1199  −20  20  0  200
301  rcc  1220  0  0  0  0  50  2.495
302  rcc  1220  0  0  0  0  50  12.495
303  rcc  1220  0  0  0  0  50  15.495
304  rpp  1201  1239  −20  20  0  200
311  rcc  1260  0  0  0  0  50  2.495
312  rcc  1260  0  0  0  0  50  12.495
313  rcc  1260  0  0  0  0  50  17.495
314  rpp  1241  1279  −20  20  0  200
321  rcc  1300  0  0  0  0  100  2.495
322  rcc  1300  0  0  0  0  100  2.495
323  rcc  1300  0  0  0  0  100  2.495
324  rpp  1281  1319  −20  20  0  200
331  rcc  1340  0  0  0  0  100  2.495
332  rcc  1340  0  0  0  0  100  2.495
333  rcc  1340  0  0  0  0  100  3.495
334  rpp  1321  1359  −20  20  0  200
341  rcc  1380  0  0  0  0  100  2.495
342  rcc  1380  0  0  0  0  100  2.495
343  rcc  1380  0  0  0  0  100  5.495
344  rpp  1361  1399  −20  20  0  200
351  rcc  1420  0  0  0  0  100  2.495
352  rcc  1420  0  0  0  0  100  2.495
353  rcc  1420  0  0  0  0  100  7.495
354  rpp  1401  1439  −20  20  0  200
361  rcc  1460  0  0  0  0  100  2.495
362  rcc  1460  0  0  0  0  100  3.495
363  rcc  1460  0  0  0  0  100  3.495
364  rpp  1441  1479  −20  20  0  200
371  rcc  1500  0  0  0  0  100  2.495
372  rcc  1500  0  0  0  0  100  3.495
373  rcc  1500  0  0  0  0  100  4.495
374  rpp  1481  1519  −20  20  0  200
381  rcc  1540  0  0  0  0  100  2.495
382  rcc  1540  0  0  0  0  100  3.495
383  rcc  1540  0  0  0  0  100  6.495
384  rpp  1521  1559  −20  20  0  200
391  rcc  1580  0  0  0  0  100  2.495
392  rcc  1580  0  0  0  0  100  3.495
393  rcc  1580  0  0  0  0  100  8.495
394  rpp  1561  1599  −20  20  0  200
401  rcc  1620  0  0  0  0  100  2.495
402  rcc  1620  0  0  0  0  100  7.495
403  rcc  1620  0  0  0  0  100  7.495
404  rpp  1601  1639  −20  20  0  200
411  rcc  1660  0  0  0  0  100  2.495
412  rcc  1660  0  0  0  0  100  7.495
413  rcc  1660  0  0  0  0  100  8.495
414  rpp  1641  1679  −20  20  0  200
421  rcc  1700  0  0  0  0  100  2.495
422  rcc  1700  0  0  0  0  100  7.495
```

```
423  rcc  1700  0  0  0  0  100  10.495
424  rpp  1681  1719  −20  20  0  200
431  rcc  1740  0  0  0  0  100  2.495
432  rcc  1740  0  0  0  0  100  7.495
433  rcc  1740  0  0  0  0  100  12.495
434  rpp  1721  1759  −20  20  0  200
441  rcc  1780  0  0  0  0  100  2.495
442  rcc  1780  0  0  0  0  100  12.495
443  rcc  1780  0  0  0  0  100  12.495
444  rpp  1761  1799  −20  20  0  200
451  rcc  1820  0  0  0  0  100  2.495
452  rcc  1820  0  0  0  0  100  12.495
453  rcc  1820  0  0  0  0  100  13.495
454  rpp  1801  1839  −20  20  0  200
461  rcc  1860  0  0  0  0  100  2.495
462  rcc  1860  0  0  0  0  100  12.495
463  rcc  1860  0  0  0  0  100  15.495
464  rpp  1841  1879  −20  20  0  200
471  rcc  1900  0  0  0  0  100  2.495
472  rcc  1900  0  0  0  0  100  12.495
473  rcc  1900  0  0  0  0  100  17.495
474  rpp  1881  1919  −20  20  0  200
481  rcc  1940  0  0  0  0  200  2.495
482  rcc  1940  0  0  0  0  200  2.495
483  rcc  1940  0  0  0  0  200  2.495
484  rpp  1921  1959  −20  20  0  200
491  rcc  1980  0  0  0  0  200  2.495
492  rcc  1980  0  0  0  0  200  2.495
493  rcc  1980  0  0  0  0  200  3.495
494  rpp  1961  1999  −20  20  0  200
501  rcc  2020  0  0  0  0  200  2.495
502  rcc  2020  0  0  0  0  200  2.495
503  rcc  2020  0  0  0  0  200  5.495
504  rpp  2001  2039  −20  20  0  200
511  rcc  2060  0  0  0  0  200  2.495
512  rcc  2060  0  0  0  0  200  2.495
513  rcc  2060  0  0  0  0  200  7.495
514  rpp  2041  2079  −20  20  0  200
521  rcc  2100  0  0  0  0  200  2.495
522  rcc  2100  0  0  0  0  200  3.495
523  rcc  2100  0  0  0  0  200  3.495
524  rpp  2081  2119  −20  20  0  200
531  rcc  2140  0  0  0  0  200  2.495
532  rcc  2140  0  0  0  0  200  3.495
533  rcc  2140  0  0  0  0  200  4.495
534  rpp  2121  2159  −20  20  0  200
541  rcc  2180  0  0  0  0  200  2.495
542  rcc  2180  0  0  0  0  200  3.495
543  rcc  2180  0  0  0  0  200  6.495
544  rpp  2161  2199  −20  20  0  200
551  rcc  2220  0  0  0  0  200  2.495
552  rcc  2220  0  0  0  0  200  3.495
553  rcc  2220  0  0  0  0  200  8.495
554  rpp  2201  2239  −20  20  0  200
561  rcc  2260  0  0  0  0  200  2.495
562  rcc  2260  0  0  0  0  200  7.495
563  rcc  2260  0  0  0  0  200  7.495
564  rpp  2241  2279  −20  20  0  200
571  rcc  2300  0  0  0  0  200  2.495
572  rcc  2300  0  0  0  0  200  7.495
573  rcc  2300  0  0  0  0  200  8.495
574  rpp  2281  2319  −20  20  0  200
581  rcc  2340  0  0  0  0  200  2.495
582  rcc  2340  0  0  0  0  200  7.495
583  rcc  2340  0  0  0  0  200  10.495
584  rpp  2321  2359  −20  20  0  200
591  rcc  2380  0  0  0  0  200  2.495
592  rcc  2380  0  0  0  0  200  7.495
593  rcc  2380  0  0  0  0  200  12.495
594  rpp  2361  2399  −20  20  0  200
601  rcc  2420  0  0  0  0  200  2.495
602  rcc  2420  0  0  0  0  200  12.495
603  rcc  2420  0  0  0  0  200  12.495
604  rpp  2401  2439  −20  20  0  200
611  rcc  2460  0  0  0  0  200  2.495
612  rcc  2460  0  0  0  0  200  12.495
```

```
613 rcc 2460 0 0 0 0 200 13.495
614 rpp 2441 2479 −20 20 0 200
621 rcc 2500 0 0 0 0 200 2.495
622 rcc 2500 0 0 0 0 200 12.495
623 rcc 2500 0 0 0 0 200 15.495
624 rpp 2481 2519 −20 20 0 200
631 rcc 2540 0 0 0 0 200 2.495
632 rcc 2540 0 0 0 0 200 12.495
633 rcc 2540 0 0 0 0 200 17.495
634 rpp 2521 2559 −20 20 0 200
9999 rpp −1 2561 −40 40 −10 210

sdef par=n x=d1 y=−10 z=fx=d2 vec=0 1 0 dir=1 erg=d3
si3 1e−10 1.39012e−10
sp3 0 1
si4 17.505 22.495
sp4 0 1
si5 0 10
sp5 0 1
si14 56.505 63.495
sp14 0 1
si15 0 10
sp15 0 1
si24 94.505 105.495
sp24 0 1
si25 0 10
sp25 0 1
si34 132.505 147.495
sp34 0 1
si35 0 10
sp35 0 1
si44 176.505 183.495
sp44 0 1
si45 0 10
sp45 0 1
si54 215.505 224.495
sp54 0 1
si55 0 10
sp55 0 1
si64 253.505 266.495
sp64 0 1
si65 0 10
sp65 0 1
si74 291.505 308.495
sp74 0 1
si75 0 10
sp75 0 1
si84 332.505 347.495
sp84 0 1
si85 0 10
sp85 0 1
si94 371.505 388.495
sp94 0 1
si95 0 10
sp95 0 1
si104 409.505 430.495
sp104 0 1
si105 0 10
sp105 0 1
si114 447.505 472.495
sp114 0 1
si115 0 10
sp115 0 1
si124 487.505 512.495
sp124 0 1
si125 0 10
sp125 0 1
si134 526.505 553.495
sp134 0 1
si135 0 10
sp135 0 1
si144 564.505 595.495
sp144 0 1
si145 0 10
sp145 0 1
si154 602.505 637.495
```

```
sp154 0 1
si155 0 10
sp155 0 1
si164 657.505 662.495
sp164 0 1
si165 0 50
sp165 0 1
si174 696.505 703.495
sp174 0 1
si175 0 50
sp175 0 1
si184 734.505 745.495
sp184 0 1
si185 0 50
sp185 0 1
si194 772.505 787.495
sp194 0 1
si195 0 50
sp195 0 1
si204 816.505 823.495
sp204 0 1
si205 0 50
sp205 0 1
si214 855.505 864.495
sp214 0 1
si215 0 50
sp215 0 1
si224 893.505 906.495
sp224 0 1
si225 0 50
sp225 0 1
si234 931.505 948.495
sp234 0 1
si235 0 50
sp235 0 1
si244 972.505 987.495
sp244 0 1
si245 0 50
sp245 0 1
si254 1011.5 1028.49
sp254 0 1
si255 0 50
sp255 0 1
si264 1049.51 1070.49
sp264 0 1
si265 0 50
sp265 0 1
si274 1087.51 1112.49
sp274 0 1
si275 0 50
sp275 0 1
si284 1127.51 1152.49
sp284 0 1
si285 0 50
sp285 0 1
si294 1166.51 1193.49
sp294 0 1
si295 0 50
sp295 0 1
si304 1204.51 1235.49
sp304 0 1
si305 0 50
sp305 0 1
si314 1242.51 1277.49
sp314 0 1
si315 0 50
sp315 0 1
si324 1297.51 1302.49
sp324 0 1
si325 0 100
sp325 0 1
si334 1336.51 1343.49
sp334 0 1
si335 0 100
sp335 0 1
si344 1374.51 1385.49
```

```
sp344 0 1
si345 0 100
sp345 0 1
si354 1412.51 1427.49
sp354 0 1
si355 0 100
sp355 0 1
si364 1456.51 1463.49
sp364 0 1
si365 0 100
sp365 0 1
si374 1495.51 1504.49
sp374 0 1
si375 0 100
sp375 0 1
si384 1533.51 1546.49
sp384 0 1
si385 0 100
sp385 0 1
si394 1571.51 1588.49
sp394 0 1
si395 0 100
sp395 0 1
si404 1612.51 1627.49
sp404 0 1
si405 0 100
sp405 0 1
si414 1651.51 1668.49
sp414 0 1
si415 0 100
sp415 0 1
si424 1689.51 1710.49
sp424 0 1
si425 0 100
sp425 0 1
si434 1727.51 1752.49
sp434 0 1
si435 0 100
sp435 0 1
si444 1767.51 1792.49
sp444 0 1
si445 0 100
sp445 0 1
si454 1806.51 1833.49
sp454 0 1
si455 0 100
sp455 0 1
si464 1844.51 1875.49
sp464 0 1
si465 0 100
sp465 0 1
si474 1882.51 1917.49
sp474 0 1
si475 0 100
sp475 0 1
si484 1937.51 1942.49
sp484 0 1
si485 0 200
sp485 0 1
si494 1976.51 1983.49
sp494 0 1
si495 0 200
sp495 0 1
si504 2014.51 2025.49
sp504 0 1
si505 0 200
sp505 0 1
si514 2052.51 2067.49
sp514 0 1
si515 0 200
sp515 0 1
si524 2096.51 2103.49
sp524 0 1
si525 0 200
sp525 0 1
si534 2135.51 2144.49
```

```
sp534 0 1
si535 0 200
sp535 0 1
si544 2173.51 2186.49
sp544 0 1
si545 0 200
sp545 0 1
si554 2211.51 2228.49
sp554 0 1
si555 0 200
sp555 0 1
si564 2252.51 2267.49
sp564 0 1
si565 0 200
sp565 0 1
si574 2291.51 2308.49
sp574 0 1
si575 0 200
sp575 0 1
si584 2329.51 2350.49
sp584 0 1
si585 0 200
sp585 0 1
si594 2367.51 2392.49
sp594 0 1
si595 0 200
sp595 0 1
si604 2407.51 2432.49
sp604 0 1
si605 0 200
sp605 0 1
si614 2446.51 2473.49
sp614 0 1
si615 0 200
sp615 0 1
si624 2484.51 2515.49
sp624 0 1
si625 0 200
sp625 0 1
si634 2522.51 2557.49
sp634 0 1
si635 0 200
sp635 0 1
si1 S
       4 14 24 34 44 54 64 74 84 94 104 114 124 134 144 154
       164 174 184 194 204 214 224 234 244 254 264 274 284
       294 304 314 324 334 344 354 364 374 384 394 404 414
       424 434 444 454 464 474 484 494 504 514 524 534 544 554
       564 574 584 594 604 614 624 634
sp1
       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       1 1 1 1 1 1 1 1 1
ds2 S
       5 15 25 35 45 55 65 75 85 95 105 115 125 135 145 155 165
       175 185 195 205 215 225 235 245 255 265 275 285 295 305
       315 325 335 345 355 365 375 385 395 405 415 425 435 445
       455 465 475 485 495 505 515 525 535 545 555 565 575 585
       595 605 615 625 635
m1    2003 -1.0 gas=1 cond=0
m2    1001 -0.143716
      6012 -0.856284
      gas=0 cond=0
m3    26000 -1.0 gas=0 cond=1
f4:n 1
fm4 -195.565 1 103
f14:n 11
fm14 -195.565 1 103
f24:n 21
fm24 -195.565 1 103
f34:n 31
fm34 -195.565 1 103
f44:n 41
fm44 -195.565 1 103
f54:n 51
fm54 -195.565 1 103
```

```
f64:n 61
fm64 −195.565 1 103
f74:n 71
fm74 −195.565 1 103
f84:n 81
fm84 −195.565 1 103
f94:n 91
fm94 −195.565 1 103
f104:n 101
fm104 −195.565 1 103
f114:n 111
fm114 −195.565 1 103
f124:n 121
fm124 −195.565 1 103
f134:n 131
fm134 −195.565 1 103
f144:n 141
fm144 −195.565 1 103
f154:n 151
fm154 −195.565 1 103
f164:n 161
fm164 −977.824 1 103
f174:n 171
fm174 −977.824 1 103
f184:n 181
fm184 −977.824 1 103
f194:n 191
fm194 −977.824 1 103
f204:n 201
fm204 −977.824 1 103
f214:n 211
fm214 −977.824 1 103
f224:n 221
fm224 −977.824 1 103
f234:n 231
fm234 −977.824 1 103
f244:n 241
fm244 −977.824 1 103
f254:n 251
fm254 −977.824 1 103
f264:n 261
fm264 −977.824 1 103
f274:n 271
fm274 −977.824 1 103
f284:n 281
fm284 −977.824 1 103
f294:n 291
fm294 −977.824 1 103
f304:n 301
fm304 −977.824 1 103
f314:n 311
fm314 −977.824 1 103
f324:n 321
fm324 −1955.65 1 103
f334:n 331
fm334 −1955.65 1 103
f344:n 341
fm344 −1955.65 1 103
f354:n 351
fm354 −1955.65 1 103
f364:n 361
fm364 −1955.65 1 103
f374:n 371
fm374 −1955.65 1 103
f384:n 381
fm384 −1955.65 1 103
f394:n 391
fm394 −1955.65 1 103
f404:n 401
fm404 −1955.65 1 103
f414:n 411
fm414 −1955.65 1 103
f424:n 421
fm424 −1955.65 1 103
f434:n 431
fm434 −1955.65 1 103
```

```
f444:n 441
fm444 −1955.65 1 103
f454:n 451
fm454 −1955.65 1 103
f464:n 461
fm464 −1955.65 1 103
f474:n 471
fm474 −1955.65 1 103
f484:n 481
fm484 −3911.3 1 103
f494:n 491
fm494 −3911.3 1 103
f504:n 501
fm504 −3911.3 1 103
f514:n 511
fm514 −3911.3 1 103
f524:n 521
fm524 −3911.3 1 103
f534:n 531
fm534 −3911.3 1 103
f544:n 541
fm544 −3911.3 1 103
f554:n 551
fm554 −3911.3 1 103
f564:n 561
fm564 −3911.3 1 103
f574:n 571
fm574 −3911.3 1 103
f584:n 581
fm584 −3911.3 1 103
f594:n 591
fm594 −3911.3 1 103
f604:n 601
fm604 −3911.3 1 103
f614:n 611
fm614 −3911.3 1 103
f624:n 621
fm624 −3911.3 1 103
f634:n 631
fm634 −3911.3 1 103
e0
      1e−10 1.39012e−10 1.93242e−10 2.68629e−10 3.73426e−10 5.19105e−10
      7.21616e−10 1.00313e−09 1.39447e−09 1.93847e−09 2.6947e−09
      3.74594e−09 5.20729e−09 7.23874e−09 1.00627e−08 1.39883e−08
      1.94454e−08 2.70313e−08 3.75767e−08 5.22359e−08 7.2614e−08
      1.00942e−07 1.40321e−07 1.95062e−07 2.71159e−07 3.76943e−07
      5.23994e−07 7.28412e−07 1.01258e−06 1.4076e−06 1.95673e−06
      2.72008e−06 3.78122e−06 5.25634e−06 7.30692e−06 1.01575e−05
      1.41201e−05 1.96285e−05 2.72859e−05 3.79306e−05 5.27279e−05
      7.32979e−05 0.000101893 0.000141642 0.000196899 0.000273713
      0.000380493 0.000528929 0.000735273 0.00102211 0.00142086
      0.00197516 0.0027457 0.00381684 0.00530585 0.00737574
      0.0102531 0.014253 0.0198134 0.0275429 0.0382878 0.0532245
      0.0739883 0.102852 0.142977 0.198754 0.276291 0.384077
      0.533911 0.742198 1.03174 1.43424 1.99376 2.77156 3.85279
      5.35582 7.44521 10.3497 14.3873 20
t0
      0.0001 0.000132194 0.000174753 0.000231013 0.000305386 0.000403702
      0.00053367 0.00070548 0.000932603 0.00123285 0.00162975 0.00215443
      0.00284804 0.00376494 0.00497702 0.00657933 0.00869749 0.0114976
      0.0151991 0.0200923 0.0265609 0.0351119 0.0464159 0.0613591
      0.0811131 0.107227 0.141747 0.187382 0.247708 0.327455 0.432876
      0.572237 0.756463 1 1.32194 1.74753 2.31013 3.05386 4.03702
      5.3367 7.0548 9.32603 12.3285 16.2975 21.5443 28.4804 37.6494
      49.7702 65.7933 86.9749 114.976 151.991 200.923 265.609 351.119
      464.159 613.591 811.131 1072.27 1417.47 1873.82 2477.08 3274.55
      4328.76 5722.37 7564.63 10000 13219.4 17475.3 23101.3 30538.6 40370.2
      53367 70548 93260.3 123285 162975 215443 284804 376494 497702
      657933 869749 1.14976e+06 1.51991e+06 2.00923e+06 2.65609e+06
      3.51119e+06 4.64159e+06 6.13591e+06 8.11131e+06 1.07227e+07
      1.41747e+07 1.87382e+07 2.47708e+07 3.27455e+07 4.32876e+07
      5.72237e+07 7.56463e+07 1e+08
mode n
prdmp j j −1
ctme 240
nps 2.1e9
```

A.4 Truck Cargo Materials

Material	**Isotope**	**Weight Fraction**
Low-Z / Oak Wood	H	0.05789
	C	0.48000
	O	0.46000
Mid-Z / Tile	O	0.524858
	Al	0.005227
	Si	0.449011
	Ca	0.014419
	Fe	0.007213
High-Z / Structural Steel	Si	0.019755
	Cr	0.181400
	Mn	0.020198
	Fe	0.650753
	Ni	0.113436
	Mo	0.014458

Table A.5: Truck Cargo Materials

NORM	**K-40** [Bq/kg]	**Ra-226** [Bq/kg]	**U-238** [Bq/kg]	**Th-232** [Bq/kg]
adobe	1150	0	55	111.5
alum_shales	155	0	2200	6.1
banana	130	0	0	0
basalt	3900	0	37	37
beer	14	0	0	0
brazil_nuts	210	111	0	0
carrot	130	0.03	0	0
cat_litter	250	0	78.5	30.5
coal_ash	552	0	248	162
concrete	325	0	40	40
diorite	5200	0	51	24
feldspar	3000	0	70	135
fertilizer	4020	0	1265	25
gabbro	2100	0	33	20
granite	5300	0	265	55
granodiorite	7800	0	99	73
light_salt	8060	0	0	0
lima_beans	170	0.06	0	0
limestone	780	0	15	5.3
marble	120	0	25	20
marble_tile	965	0	63	220
monazite_sand	55	0	515	1525
peridotite	2300	0	18	12
phosphates	0	0	1225	87.5
red_meat	110	0	0	0
sandstone	520	0	70	70
scotch_brite	210	0	350	310
shales	7800	0	40	4.5
slate	1000	0	70	70
us_soil	10300	0	140	130
white_potatoes	130	0	0	0
none	0	0	0	0

Table A.6: Cargo NORM Sources

Appendix B

XPASS

B.1 Input File Format

An XPASS input file is composed of "blocks" of input. A block is started by a case-sensitive keyword. The contents of a block must appear directly beneath this keyword, indented by one more tab than the block keyword. The main blocks in the input file are *physics*, *source*, *vehicle*, *background*, *detection*, and *save*. Each of these blocks may have additional blocks embedded within them. The following tables summarize the input for each block.

Keyword	**Req'd**	**Values**	**Description**
photon	yes	[on|off]	toggle photon transport
neutron	yes	[on|off]	toggle neutron transport
fissgamma	yes	[on|off]	toggle prompt fission gamma source
background	yes	[on|off]	toggle natural background sources
mactime	yes	[on|off]	toggle macroscopic time steps, if this is turned off, only one time step in which the average source position is in front of the average detector position will be computed
interval	yes	>0 (float)	macroscopic time step in [seconds]
ergfac	yes	>0 (integer)	factor by which to reduce photon resolution (e.g. ergfac 2 will reduce 1024 bins to 512)
refeps	yes	>0 (float)	neutron reflection convergence criteria

Table B.1: XPASS Input: *physics* block

Keyword	**Req'd**	**Values**	**Description**
snm	no	n/a	initiates *snm* block of input, may have as many of these blocks as desired
norm	no	n/a	initiates *norm* block of input, may have as many of these blocks as desired

Table B.2: XPASS Input: *source* block

Keyword	**Req'd**	**Values**	**Description**
type	yes	[wgpu, rgpu, heu, du]	general type of SNM desired
mass	yes	>0 (float)	mass of the SNM in [grams]
iso	yes	n/a	initiate *iso* block of input
age	yes	$\geq$ 0 (float)	age of the SNM in [years]
shield	no	n/a	initiate *shield* block of input
posx	yes	-129 to 129 (float)	x-position (width) in the cargo [cm]
posy	yes	-807.72 to 807.72 (float)	y-position (length) in the cargo [cm]
posz	yes	0 to 258.95 (float)	z-position (height) in the cargo [cm]

Table B.3: XPASS Input: *source*→*snm* block

Keyword	Req'd	Values	Description
[u232, u234, u235, u236, u238, pu236, pu238, pu239, pu240, pu241, pu242]	yes	>0, ≤ 100 (float)	weight percentage of isotope in [%], may list as many isotopes as desired

Table B.4: XPASS Input: *source*→*snm*→*iso* block

Keyword	Req'd	Values	Description
layer	no	n/a	initiate *layer* block of input

Table B.5: XPASS Input: *source*→*snm*→*shield* block

Keyword	Req'd	Values	Description
type	yes	[lead,bopoly]	type of shielding, lead or borated polyethylene
thick	yes	≥ 0 (float)	radial thickness of shielding in [cm]
stream	yes	≥ 0, ≤ 1 (float)	fraction of shielding that is streaming pathways

Table B.6: XPASS Input:*source*→*snm*→*shield*→*layer* block

Keyword	**Req'd**	**Values**	**Description**
type	yes	[adobe, alum_shales, banana, basalt, beer, brazil_nuts, carrot, cat_litter, coal_ash, concrete, diorite, feldspar, fertilizer, gabbro, granite, granodiorite, light_salt, lima_beans, limestone, marble, marble_tile, monazite_sand, peridotite, phosphates, red_meat, sandstone, scotch_brite, shales, slate, us_soil, white_potatoes, none]	type of NORM
frac	yes	$\geq 0, \leq 1$ (float)	fraction of the cargo that is NORM

Note: NORM does not affect cargo transport

Table B.7: XPASS Input: *source*→*norm* block

Keyword	**Req'd**	**Values**	**Description**
truck	no	n/a	initiates *truck* block of input

Table B.8: XPASS Input: *vehicle* block

Keyword	**Req'd**	**Values**	**Description**
cargo	yes	n/a	initiates *cargo* block of input
velocity	yes	>0 (float)	speed of the truck in [km/hour]

Table B.9: XPASS Input: *vehicle*→*truck* block

Keyword	**Req'd**	**Values**	**Description**
[void, lowz, midz, highz]	yes	$\geq 0, \leq 1$ (float)	fraction (or probability) of cargo type, may use one or all four types

Table B.10: XPASS Input: *vehicle*→*truck*→*cargo* block

Keyword	Req'd	Values	Description
photon	yes	n/a	initiates *photon* block of input
neutron	yes	n/a	initiates *neutron* block of input

Table B.11: XPASS Input: *background* block

Keyword	Req'd	Values	Description
usoil	yes	>0 (float)	concentration of uranium series in soil [Bq/kg]
uconc	yes	>0 (float)	concentration of uranium series in concrete [Bq/kg]
thsoil	yes	>0 (float)	concentration of thorium series in soil [Bq/kg]
thconc	yes	>0 (float)	concentration of thorium series in concrete [Bq/kg]
ksoil	yes	>0 (float)	concentration of potassium-40 in soil [Bq/kg]
kconc	yes	>0 (float)	concentration of potassium-40 in concrete [Bq/kg]

Table B.12: XPASS Input: *background→photon* block

Keyword	Req'd	Values	Description
lat	yes	-90 to 90 (float)	latitude in degrees
long	yes	0 to 345 (float)	longitude in degrees
smod	yes	≥0,≤1 (float)	solar modulation factor, 0 is a solar activity minimum, 1 is a maximum
elev	yes	≥ 0 (float)	elevation in [m]

Table B.13: XPASS Input: *background→neutron* block

Keyword	Req'd	Values	Description
pvt	no	n/a	initiates *pvt* block, may have as many as desired
nai	no	n/a	initiates *nai* block, may have as many as desired
hpge	no	n/a	initiates *hpge* block, may have as many as desired
he3	no	n/a	initiates *he3* block, may have as many as desired
ss	no	n/a	initiates *ss* block, may have as many as desired

Table B.14: XPASS Input: *detection* block

Keyword	Req'd	Values	Description
posx	yes	(float)	x-position of center detector face [cm]
posy	yes	(float)	relative y-position of center detector face [cm], relative because truck moves in this dimension. If you have an array of detectors, you can place them all at posy=0, but physically that is not possible
posz	yes	$\geq$ -132 (float)	z-position (elevation) of center detector face [cm]
dimx	yes	$\geq$1,$\leq$200 (float)	x-dimension detector [cm]
dimy	yes	$\geq$1,$\leq$200 (float)	y-dimension detector [cm]
dimz	yes	$\geq$1,$\leq$200 (float)	z-dimension detector [cm]
eff	yes	$\geq$ 0 (float)	light/charge collection efficiency
gebA	yes	$\geq$ 0 (float)	GEB Parameter A
gebB	yes	$\geq$ 0 (float)	GEB Parameter B
gebC	yes	$\geq$ 0 (float)	GEB Parameter C
fanofac	yes	$\geq$0,$\leq$1 (float)	Fano factor
alarm	no	n/a	initiates *alarm* block to apply alarm algorithm(s) to this detector only

Table B.15: XPASS Input: *detection→pvt/nai/hpge* block

Keyword	Req'd	Values	Description
gc	no	n/a	initiates *gc* block to apply gross count to this detector only, may have multiple of this algorithm
ew	no	n/a	initiates *gc* block to apply energy window to this detector only, may have multiple of this algorithm
template	no	n/a	initiates *gc* block to apply template matching to this detector only, may have multiple of this algorithm

Table B.16: XPASS Input: *detection*→*pvt/nai/hpge*→*alarm* block

Keyword	Req'd	Values	Description
nint	yes	>0 (integer)	number of macroscopic time steps to sum over
nuisance	no	n/a	initiates *nuisance* block to read a nuisance spectrum

Table B.17: XPASS Input: *detection*→*pvt/nai/hpge*→*alarm*→*gc* block

Keyword	Req'd	Values	Description
nint	yes	>0 (integer)	number of macroscopic time steps to sum over
nwindow spacing	no no	>0 (integer) [lin,log]	number of windows to space over the spectrum linear or logarithmic spacing
nbound bounds	no no	>0 (integer) (float list)	number of user-specified window bounds energy window boundaries
nuisance	no	n/a	initiates *nuisance* block to read a nuisance spectrum

Table B.18: XPASS Input: *detection*→*pvt/nai/hpge*→*alarm*→*ew* block

Keyword	Req'd	Values	Description
nint	yes	>0 (integer)	number of macroscopic time steps to sum over
types	yes	n/a	initiate *types* block to list templates to attempt matching
nuisance	no	n/a	initiates *nuisance* block to read a nuisance spectrum

Table B.19: XPASS Input: *detection*→*pvt/nai/hpge*→*alarm*→*template* block

Keyword	Req'd	Values	Description
[du, heu, vheu, rgpu, wgpu, ...]	yes		must list at least one SNM type to match

Table B.20: XPASS Input: *detection*→*pvt/nai/hpge*→*alarm*→*template*→*types* block

Keyword	Req'd	Values	Description
name	yes	(string no spaces)	name of job to save
[signal, nuisance]	no		list spectral information to save to file

Table B.21: XPASS Input: *save* block

Keyword	**Req'd**	**Values**	**Description**
posx	yes	(float)	x-position of center detector face [cm]
posy	yes	(float)	relative y-position of center detector face [cm], relative because truck moves in this dimension. If you have an array of detectors, you can place them all at posy=0, but physically that is not possible
posz	yes	$\geq$ -132 (float)	z-position (elevation) of center detector face [cm]
height	yes	$\geq$10,$\leq$200 (float)	helium-3 tube height [cm]
modrad	yes	$\geq$0,$\leq$10 (float)	polyethylene radial thickness [cm]
refrad	yes	$\geq$0,$\leq$5 (float)	iron reflector/shield thickness [cm]
actwidth	yes	>0 (float)	actual width of moderator [cm]
eff	yes	$\geq$ 0 (float)	charge collection efficiency
fanofac	yes	$\geq$0,$\leq$1 (float)	Fano factor
alarm	no	n/a	initiates *alarm* block to apply alarm algorithm(s) to this detector only

Table B.22: XPASS Input: *detection*→*he3* block

Keyword	**Req'd**	**Values**	**Description**
posx	yes	(float)	x-position of center detector face [cm]
posy	yes	(float)	relative y-position of center detector face [cm], relative because truck moves in this dimension. If you have an array of detectors, you can place them all at posy=0, but physically that is not possible
posz	yes	$\geq$ -132 (float)	z-position (elevation) of center detector face [cm]
modt	yes	$\geq$0,$\leq$10 (float)	polyethylene slab thickness [cm]
area	yes	>0 (float)	area of detector face [cm]
eff	yes	$\geq$ 0 (float)	charge collection efficiency
fanofac	yes	$\geq$0,$\leq$1 (float)	Fano factor
alarm	no	n/a	initiates *alarm* block to apply alarm algorithm(s) to this detector only

Table B.23: XPASS Input: *detection*→*ss* block

B.2 Source Code

Listing B.1: alarm.cpp

```
#include "alarm.hpp"
#include <math.h>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "extras.hpp"
#include "errh.hpp"

grosscount::grosscount ( double fanofac )
{
    my_fanofac = fanofac;
}

double grosscount::threshold ( double nuisanceMean,
    double nuisanceStdDev )
{
    const double mu_n = nuisanceMean;
    const double sig_n = nuisanceStdDev;
    return ( mu_n + sig_n*sqrt ( 2.0 ) *erfinv ( 1.0-2.0*my_FAP ) );
}

double grosscount::q ( double threshold,
    double totSig,double totSigStdDev )
{
    const double t = threshold;
    const double mu_s = totSig;
    const double sig_s = totSigStdDev;
    return ( ( 1.0/2.0 ) + ( 1.0/2.0 )
            * erf ( ( t-mu_s ) / ( sqrt ( 2.0 ) *sig_s ) ) );
}


double grosscount::calculateEvasionProb ( double time,double FAP )
{
    my_FAP = FAP;

    const double totSig = my_signal.sum()*time;
    const double totNuis = my_nuisance.sum()*time;

    //std::cout << "time = " << time << std::endl;

    //std::cout << "totSig = " << totSig << std::endl;
    //std::cout << "totNuis = " << totNuis << std::endl;

    const double t = threshold( totNuis,sqrt(my_fanofac*totNuis) );

    //std::cout << "threshold = " << t << std::endl;

    return q( t,totSig,sqrt(my_fanofac*totSig) );
}



//double grosscountpoisson::F( double lambda,int x )
//{
    //double sum = 0.0;
    //for ( int i=0;i<x;++i )
    //{
        //sum += exp(-lambda)*pow(lambda,1.0*i)/factorial(i);
    //}
    //return sum;
//}

//double grosscountpoisson::threshold ( double nuisanceMean )
//{
    //const double lambda = nuisanceMean;
    //int x = 0;
    //while ( true )
    //{
        //const double p = 1.0 - F(lambda,x);
        //if ( p < my_FAP )
```

```
        //{
            //return static_cast<double>(x);
        //}
        //x++;
    //}
//}

//double grosscountpoisson::calculateThreshold ( )
//{
    //return threshold ( my_bg );
//}


//double grosscountpoisson::q ( double threshold,double sourceMean )
//{
    //const double t = threshold;
    //const double lambda = sourceMean;

    //return 1.0-F(lambda,t);
//}



//templatematch::templatematch ( double fanofac,
    //double gebA,double gebB,double gebC,
    //const std::string& dir,
    //const std::vector<std::string>& tempname )
//{
    //my_fanofac = fanofac;
    //my_gebA = gebA;
    //my_gebB = gebB;
    //my_gebC = gebC;
    //for ( unsigned int i=0;i<tempname.size();++i )
    //{
        //double dummy;
        //std::ifstream in((dir+tempname[i]+".dat").c_str());
        //while ( in.peek() != EOF )
        //{
            //in >> dummy;
            //if ( ! in.eof() )
            //{
                ////std::cout << dummy/1000.0 << " ";
                //my_energy.push_back(dummy/1000.0);
                //in >> dummy;
                ////std::cout << dummy << std::endl;
                //my_intensity.push_back(dummy);
            //}
        //}
    //}
//}

//std::vector<double> templatematch::convertToWindow(
    //const spectrum& spec,double erg )
//{
    //// calculate FWHM
    //const double a = my_gebA;
    //const double b = my_gebB;
    //const double c = my_gebC;
    //const double FWHM = a+b*sqrt(erg+c*pow(erg,2.0));

    //// calculate energy boundaries
    //std::vector<double> energy(4);
    //energy[0] = spec.firsterg();
    //energy[1] = erg - FWHM/2.0;
    //energy[2] = erg + FWHM/2.0;
    //energy[3] = spec.lasterg();

    ////std::cout << "energy = " << energy << std::endl;

    //matrix<double> T = genTransform ( spec.erg(),energy );

    //return T*(spec.get());
//}

//double templatematch::calculateEvasionProb ( const spectrum& signal,
    //const spectrum& noise,const spectrum& nuisance,double time,
```

```
    //double FAP )
//{
    //my_FAP = FAP;
    //my_numWindow = 3;
    //double minq = std::numeric_limits<double>::max();
    //for ( unsigned int e=0;e<my_energy.size();++e )
    //{
        ////std::cout << "calculating windows for energy "
        //  << my_energy[e] << std::endl;
        //my_noise = convertToWindow(noise,my_energy[e])*time;
        //my_signal = convertToWindow(signal,my_energy[e])*time;
        //my_nuisance = convertToWindow(nuisance,my_energy[e])*time;

        ////std::cout << "my_noise = " << my_noise << std::endl;
        ////std::cout << "my_signal = " << my_signal << std::endl;
        ////std::cout << "my_nuisance = " << my_nuisance << std::endl;

        ////std::vector<double> R_noise = ratio ( my_noise );
        ////std::vector<double> sig_Rnoise = error ( my_noise );

        //std::vector<double> R_totSig = ratio ( my_noise+my_signal );
        //std::vector<double> sig_RtotSig = error ( my_noise+my_signal );

        //std::vector<double> R_nuisance = ratio ( my_nuisance );
        //std::vector<double> sig_Rnuisance = error ( my_nuisance );

        ////std::cout << "R_totSig = " << R_totSig << std::endl;
        ////std::cout << "R_nuisance = " << R_nuisance << std::endl;
        ////std::cout << "sig_Rnuisance = " << sig_Rnuisance << std::endl;

        //// calculate alarm threshold based on nuisance
        //std::vector<double> t = threshold ( R_nuisance,sig_Rnuisance );

        //// calculate evasion probability for middle window
        //const double evaprob = q ( t[1],R_totSig[1],sig_RtotSig[1] );

        //// get minimum evasion probability
        //if ( evaprob < minq )
        //{
            //minq = evaprob;
            //my_alarmWindowIdx = e;
        //}
    //}
    //std::cout << "energy " << my_energy[my_alarmWindowIdx]
    //  << " alarmed " << std::endl;
    //return minq;
//}


templatematch::templatematch ( double fanofac,
    double gebA,double gebB,double gebC,
    const std::string& dir,
    const std::vector<std::string>& tempname )
{
    my_fanofac = fanofac;
    my_gebA = gebA;
    my_gebB = gebB;
    my_gebC = gebC;
    my_numTemplate = static_cast<int>(tempname.size());
    my_energy.resize(my_numTemplate);
    my_intensity.resize(my_numTemplate);
    for ( unsigned int i=0;i<tempname.size();++i )
    {
        double dummy;
        std::ifstream in((dir+tempname[i]+".dat").c_str());
        if ( ! in.good( ) )
        {
            throw fatal_error("could not find file "
                +tempname[i]+".dat");
        }
        while ( in.peek() != EOF )
        {
            in >> dummy;
            if ( ! in.eof() )
            {
```

```
                //std::cout << dummy/1000.0 << " ";
                my_energy[i].push_back(dummy/1000.0);
                in >> dummy;
                //std::cout << dummy << std::endl;
                my_intensity[i].push_back(dummy);
            }
        }
    }
}

//void templatematch::ratio( const spectrum& spec,int tempNum,
    //std::vector<double>& R,std::vector<double>& sig )
//{
    //datapoint sumbin(0.0,0.0);
    //datapoint sumtot;
    //const double specsum = spec.sum();
    //sumtot.set(specsum);
    //sumtot.setAbsErr(sqrt(my_fanofac*specsum));

    //R.resize(my_energy[tempNum].size()+1);
    //sig.resize(my_energy[tempNum].size()+1);
    ////R.resize(1);
    ////sig.resize(1);
    //for ( unsigned int e=0;e<my_energy[tempNum].size();++e )
    //{
        //const double erg = my_energy[tempNum][e];
        //// calculate FWHM
        //const double a = my_gebA;
        //const double b = my_gebB;
        //const double c = my_gebC;
        //const double FWHM = a+b*sqrt(erg+c*pow(erg,2.0));
        //// calculate energy boundaries
        //std::vector<double> energy(6);
        //energy[0] = spec.firsterg();
        //energy[1] = erg - FWHM/2.0-FWHM;
        //energy[2] = erg - FWHM/2.0;
        //energy[3] = erg + FWHM/2.0;
        //energy[4] = erg + FWHM/2.0+FWHM;
        //energy[5] = spec.lasterg();
        //// get intensity in that bin
        //matrix<double> T = genTransform ( spec.erg(),energy );
        //std::vector<double> newspec = T*(spec.get());

        //datapoint pt0;
        //pt0.set(newspec[1]);
        //pt0.setAbsErr(sqrt(my_fanofac*newspec[1]));

        //datapoint pt1;
        //pt1.set(newspec[2]);
        //pt1.setAbsErr(sqrt(my_fanofac*newspec[2]));

        //datapoint pt2;
        //pt2.set(newspec[3]);
        //pt2.setAbsErr(sqrt(my_fanofac*newspec[3]));

        //// enable this line for ratio window
        //sumbin = sumbin + pt1/(pt0+pt2);
        //datapoint pt = pt1/(pt0+pt2);

        //R[e] = pt.get();
        //sig[e] = pt.getAbsErr();
    //}
    //R.back() = sumbin.get();
    //sig.back() = sumbin.getAbsErr();
//}

void templatematch::ratio( const spectrum& spec,int tempNum,
    std::vector<double>& R,std::vector<double>& sig )
{
    datapoint sumbin(0.0,0.0);
    datapoint sumtot;
    const double specsum = spec.sum();
    sumtot.set(specsum);
    sumtot.setAbsErr(sqrt(my_fanofac*specsum));

    // last bin is for total
```

```
    R.resize(my_energy[tempNum].size()+1);
    sig.resize(my_energy[tempNum].size()+1);
    //R.resize(1);
    //sig.resize(1);
    for ( unsigned int e=0;e<my_energy[tempNum].size();++e )
    {
        const double erg = my_energy[tempNum][e];
        // calculate FWHM
        const double a = my_gebA;
        const double b = my_gebB;
        const double c = my_gebC;
        const double FWHM = a+b*sqrt(erg+c*pow(erg,2.0));
        // calculate energy boundaries
        std::vector<double> energy(4);
        energy[0] = spec.firsterg();
        energy[1] = erg - FWHM/2.0;
        energy[2] = erg + FWHM/2.0;
        energy[3] = spec.lasterg();
        // get intensity in that bin
        matrix<double> T = genTransform ( spec.erg(),energy );
        std::vector<double> newspec = T*(spec.get());

        //sumbin += newspec[1];
        //sumerr += my_fanofac*newspec[1];

        datapoint pt;
        pt.set(newspec[1]);
        pt.setAbsErr(sqrt(my_fanofac*newspec[1]));

        // enable this line for ratio window
        //pt = pt/sumtot;

        R[e] = pt.get();
        sig[e] = pt.getAbsErr();
        sumbin = sumbin + pt;
    }
    R.back() = sumbin.get();
    sig.back() = sumbin.getAbsErr();
    //sumbin = sumbin/sumtot;
    //std::pair<double,double> result;
    //result.first = sumbin.get();
    //result.second = sumbin.getAbsErr();
    //return result;
}

double templatematch::calculateEvasionProb ( double time,double FAP )
{
    my_FAP = FAP;
    double minq = std::numeric_limits<double>::max();
    int templatenum;
    int energynum;
    for ( int i=0;i<my_numTemplate;++i )
    {
        ratio ( my_signal*time,i,my_R_tot,my_sigR_tot );
        ratio ( my_nuisance*time,i,my_R_nuis,my_sigR_nuis );

        for ( unsigned int e=0;e<my_R_tot.size();++e )
        {
            const double t = threshold ( my_R_nuis[e],my_sigR_nuis[e] );

            const double evaprob = q ( t,my_R_tot[e],my_sigR_tot[e] );

            // get minimum evasion probability
            if ( evaprob < minq )
            {
                minq = evaprob;
                templatenum = i;
                energynum = e;
            }
        }
    }
    return minq;
}
```

```
double energywindow::threshold ( double nuisanceMean,
    double nuisanceStdDev )
{
    const double mu_n = nuisanceMean;
    const double sig_n = nuisanceStdDev;

    // double barrel
    //return ( sqrt(2.0) * erfinv( 1.0-my_FAP ) );

    // single barrel
    return ( mu_n + sig_n*sqrt ( 2.0 ) *erfinv ( 1.0-2.0*my_FAP ) );
}

double energywindow::q ( double threshold,
    double totSig,double totSigStdDev )
{
    const double t = threshold;
    const double mu_s = totSig;
    const double sig_s = totSigStdDev;

    // double barrel
    //return (1.0/2.0) * erf ( ( t-mu_s ) / ( sqrt ( 2.0 ) *sig_s ) )
    // - (1.0/2.0) * erf ( ( t-mu_s ) / ( sqrt ( 2.0 ) *sig_s ) );

    // single barrel
    return ( ( 1.0/2.0 ) + ( 1.0/2.0 )
            * erf ( ( t-mu_s ) / ( sqrt ( 2.0 ) *sig_s ) ) );
}

energywindow::energywindow ( double fanofac,int numWindow,
    const std::string& spaceType )
    : grosscount( fanofac )
{
    if ( spaceType.compare("lin") == 0 ) my_spacing = lin;
    if ( spaceType.compare("log") == 0 ) my_spacing = log;
    my_numWindow = numWindow;
}

energywindow::energywindow ( double fanofac,
    const std::vector<double>& window )
    : grosscount( fanofac )
{
    my_numWindow = static_cast<int> ( window.size() )-1;
    my_window = window;
}

std::vector<double> energywindow::convertToWindow(
    const spectrum& spec )
{
    // calculate energy windows if not specified
    std::vector< std::vector<int> > idx ( my_numWindow );
    if ( static_cast<int> ( my_window.size() ) == 0 )
    {
        if ( my_spacing == log )
        {
            my_window = logspace( spec.firsterg(),
                spec.lasterg(),my_numWindow+1 );
        }
        else /* if ( my_spacing == lin ) */
        {
            my_window = linspace( spec.firsterg(),
                spec.lasterg(),my_numWindow+1 );
        }
    }

    matrix<double> T = genTransform ( spec.erg(),my_window );

    return T*(spec.get());
}
```

```
double energywindow::calculateEvasionProb ( double time,double FAP )
{

    my_FAP = FAP;
    my_signalW = convertToWindow(my_signal)*time;
    my_nuisanceW = convertToWindow(my_nuisance)*time;

    std::vector<double> R_totSig = ratio ( my_signalW );
    std::vector<double> sig_RtotSig = error ( my_signalW );

    std::vector<double> R_nuisance = ratio ( my_nuisanceW );
    std::vector<double> sig_Rnuisance = error ( my_nuisanceW );


    // calculate alarm threshold based on nuisance
    std::vector<double> t = threshold ( R_nuisance,sig_Rnuisance );

    // calculate evasion probability for each window
    std::vector< double > evaprob( my_numWindow );
    for ( int k=0;k<my_numWindow;++k )
    {
        evaprob[k] = q ( t[k],R_totSig[k],sig_RtotSig[k] );
    }
    // get minimum evasion probability
    double minq = std::numeric_limits<double>::max();
    for ( int k=0;k<my_numWindow;++k )
    {
        if ( evaprob[k] < minq )
        {
            minq = evaprob[k];
            my_alarmWindowIdx = k;
        }
    }
    return minq;
}


std::vector<double> energywindow::threshold (
    const std::vector<double>& nuisanceMean,
    const std::vector<double>& nuisanceStdDev )
{
    std::vector<double> t ( my_numWindow );

    //const double factor = sqrt(2)*boost::math::erf_inv(1-2*FAP);
    const double factor = sqrt ( 2 ) *erfinv ( 1-2*my_FAP );

    for ( int i=0;i<my_numWindow;++i )
    {
        t[i] = nuisanceMean[i] + nuisanceStdDev[i]*factor;
    }
    return t;
}


double sum ( std::vector<double> N,int j,int J )
{
    double sum = 0.0;
    for ( int i=j;i<J;++i )
    {
        sum += N[i];
    }
    return sum;
}

double sumroot ( std::vector<double> N,int j,int J )
{
    double sum = 0.0;
    for ( int i=j;i<J;++i )
    {
        sum += sqrt ( N[i] );
    }
    return sum;
}

std::vector<double> energywindow::ratio ( const std::vector<double>& N )
```

```
{
    const double I = my_numWindow;
    std::vector<double> ratio ( I );

    const double sum_N = sum ( N,0,I );
    //std::cout << "N = " << N << std::endl;
    //std::cout << "sum = " << sum_N << std::endl;
    for ( int i=0;i<I;++i )
    {
        ratio[i] = N[i]/sum_N;
    }
    return ratio;

}

std::vector<double> energywindow::error ( const std::vector<double>& N )
{
    const double I = my_numWindow;
    std::vector<double> error( I );

    const double F = my_fanofac;
    // calculate error for summation
    double sig = sqrt(F*N[0]+F*N[1]+2*sqrt(F*N[0])*sqrt(F*N[1]));
    for ( int i=2;i<I;++i )
    {
        sig = sqrt(sig*sig+F*N[i]+2*sig*sqrt(F*N[i]));
    }

    //double sig = F*N[0];
    //for ( int i=1;i<I;++i )
    //{
        //sig += F*N[i];
    //}
    //sig = sqrt(sig);

    // sum denominator
    const double sumDenom = sum(N,0,I);

    for ( int i=0;i<I;++i )
    {
        const double f = N[i]/sumDenom;
        const double siga = sqrt(F*N[i]);
        const double A = N[i];
        const double sigb = sig;
        const double B = sumDenom;
        error[i] = f*sqrt(siga*siga/(A*A)+sigb*sigb/(B*B)-2*siga*sigb/(A*B));
        //std::cout << f << " +/- " << error[i]/f*100.0 << "%" << std::endl;
    }
    return error;

}
```

Listing B.2: alarm.hpp

```
#ifndef _alarm_hpp_included_
#define _alarm_hpp_included_

#include <vector>
#include <string>
#include <fstream>

#include "model.hpp"

class model;

class alarma
{
    public:

    virtual double calculateEvasionProb ( double time,double FAP ) = 0;

    bool useNuisanceFile() { return (!my_nuisanceFile.empty()); };
    void setNuisance( const std::string& filename )
    {
        my_nuisanceFile = filename;
    }

    void setSignal( const spectrum& signal )
    {
        my_signal = signal;
    }
    void setNuisance( const spectrum& nuisance )
    {
        my_nuisance = nuisance;
    }
    void getNuisanceFromFile( int timeIdx )
    {
        if ( useNuisanceFile() )
        {
            my_nuisance.read( my_nuisanceFile+"_time"+str(timeIdx) );
        }
        else
        {
            throw fatal_error("no nuisance file specified \
for alarm algorithm");
        }
    }

    protected:

    spectrum my_signal;
    spectrum my_nuisance;

    double my_fanofac;

    std::string my_nuisanceFile;

};

class grosscount : public alarma
{
    public:


    double calculateEvasionProb ( double time,double FAP );

    grosscount( double fanofac );

    grosscount() {};

    protected:

    virtual double threshold(double,double);
    virtual double q(double,double,double);

    double my_FAP;
};

//class grosscountpoisson : public grosscount
```

```
//{
    //public:

    //protected:

    //virtual double q(double,double);
    //virtual double calculateThreshold ( );

    //private:

    //double threshold(double);

//};




class energywindow : public grosscount
{
    public:

    enum spacing { lin,log };

    std::vector<double> error ( const std::vector<double>&  );
    std::vector<double> ratio ( const std::vector<double>&  );
    std::vector<double> threshold ( const std::vector<double>&,
                                  const std::vector<double>& );

    double calculateEvasionProb ( double time,double FAP );

    energywindow(double,int,const std::string&);
    energywindow(double,const std::vector<double>&);
    energywindow() {};

    protected:

    virtual double threshold(double,double);
    virtual double q(double,double,double);

    std::vector<double> my_signalW;
    std::vector<double> my_nuisanceW;

    int my_alarmWindowIdx;
    std::vector<double> convertToWindow( const spectrum& spec );

    spacing my_spacing;
    int my_numWindow;
    std::vector<double> my_window;
};



//class templatematch : public energywindow
//{
    //public:

    //double calculateEvasionProb ( const spectrum& signal,
        //const spectrum& noise,const spectrum& nuisance,
        //double time,double FAP );

    //templatematch(double,double,double,double,
    //  const std::string&,const std::vector<std::string>&);

    //private:

    //std::vector<double> convertToWindow(
    //  const spectrum& spec,double );

    //std::vector<double> my_energy;
    //std::vector<double> my_intensity;
    //double my_gebA;
    //double my_gebB;
    //double my_gebC;

//};
```

```
class templatematch : public energywindow
{
    public:

    double calculateEvasionProb ( double time,double FAP );

    templatematch(double,double,double,double,
        const std::string&,const std::vector<std::string>&);

    private:

    void ratio(
        const spectrum& spec,int tempNum,
        std::vector<double>& R,
        std::vector<double>& sig );

    std::vector< std::vector<double> > my_energy;
    std::vector< std::vector<double> > my_intensity;
    double my_gebA;
    double my_gebB;
    double my_gebC;
    int my_numTemplate;

    std::vector<double> my_R_tot;
    std::vector<double> my_sigR_tot;
    std::vector<double> my_R_nuis;
    std::vector<double> my_sigR_nuis;

};

#endif
```

Listing B.3: background.cpp

```
#include "background.hpp"
#include "extras.hpp"
#include "interpolation.hpp"

#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>
#include "fileio.hpp"


int background::numInErgs ( )
{
    return my_R.numergin();
}

int background::numInErgs ( ) const
{
    return my_R.numergin();
}

int background::numOutErgs ( )
{
    return my_R.numergout();
}

int background::numOutErgs ( ) const
{
    return my_R.numergout();
}



void background::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector positions/areas
    my_detSidePtX = readbin( my_datapath+"erg0/side/detposx.dat" );
    my_detSidePtY = readbin( my_datapath+"erg0/side/detposy.dat" );
    my_detSidePtZ = readbin( my_datapath+"erg0/side/detposz.dat" );
    my_detSideArea = readbin( my_datapath+"erg0/side/detarea.dat" );
    my_detTopPtX = readbin( my_datapath+"erg0/top/detposx.dat" );
    my_detTopPtY = readbin( my_datapath+"erg0/top/detposy.dat" );
    my_detTopPtZ = readbin( my_datapath+"erg0/top/detposz.dat" );
    my_detTopArea = readbin( my_datapath+"erg0/top/detarea.dat" );

    return;
}

background::background ( const std::string& path )
{
    my_datapath = path + sep();
    readDataFile();
}

void background::initialize( const std::string& path )
{
    my_datapath = path + sep();
    readDataFile();
}


std::string background::getTallyEnergyPath( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
}

std::string background::getTallyPath(
    int ergIdx ,int rowIdx ,int colIdx )
{
    int detRow,detCol;
    if ( rowIdx == 1 )
    {
```

```
        detRow = my_detIdxRow1;
    }
    else /*if ( rowIdx == 2 )*/
    {
        detRow = my_detIdxRow2;
    }
    if ( colIdx == 1 )
    {
        detCol = my_detIdxCol1;
    }
    else /*if ( colIdx == 2 )*/
    {
        detCol = my_detIdxCol2;
    }
    if ( my_detPosType == side )
    {
        return my_datapath + "erg" + str(ergIdx)
            + sep() + "side" + sep() + "dpos"
            + str(detRow) + "_" + str(detCol) + sep();
    }
    else /*if ( my_detPosType == top )*/
    {
        return my_datapath + "erg" + str(ergIdx)
            + sep() + "top" + sep() + "dpos"
            + str(detRow) + "_" + str(detCol) + sep();
    }
}

std::vector<datapoint> background::interpolateTallies(
    double newpeak, const std::vector< double >& newerg,
    int ergIdx1, int ergIdx3 )
{
    // get indices
    const int newSrcIdx1 = ergIdx1;
    const int newSrcIdx3 = ergIdx3;

    // parse tallies and interpolate
    if ( newSrcIdx1 != srcIdx1 )
    {
        srcIdx1 = newSrcIdx1;

        std::string talErgPath = getTallyEnergyPath(ergIdx1);
        std::string talPath;

        talPath = getTallyPath(ergIdx1,1,1);
        tal1_Row1Col1->parse( talErgPath, talPath );
        tal1_Row1Col1->setSourceEnergy( my_srcErg[ergIdx1] );


        talPath = getTallyPath(ergIdx1,1,2);
        tal1_Row1Col2->parse( talErgPath, talPath );
        tal1_Row1Col2->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1,2,1);
        tal1_Row2Col1->parse( talErgPath, talPath );
        tal1_Row2Col1->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1,2,2);
        tal1_Row2Col2->parse( talErgPath, talPath );
        tal1_Row2Col2->setSourceEnergy( my_srcErg[ergIdx1] );

        if ( my_detPosType == side )
        {
            tal1_Row1->interpolate( tal1_Row1Col1,
                tal1_Row1Col2,
                my_detSidePtY[my_detIdxCol1],
                my_detSidePtY[my_detIdxCol2],
                my_detPosY,
                "lin" );
            tal1_Row2->interpolate( tal1_Row2Col1,
                tal1_Row2Col2,
                my_detSidePtY[my_detIdxCol1],
                my_detSidePtY[my_detIdxCol2],
                my_detPosY,
                "lin" );
            tal1->interpolate( tal1_Row1,
```

```
            tal1_Row2,
            my_detSidePtZ[my_detIdxRow1],
            my_detSidePtZ[my_detIdxRow2],
            my_detPosZ,
            "lin" );
    }
    else if ( my_detPosType == top )
    {
        tal1_Row1->interpolate( tal1_Row1Col1,
            tal1_Row1Col2,
            my_detTopPtY[my_detIdxCol1],
            my_detTopPtY[my_detIdxCol2],
            my_detPosY,
            "lin" );
        tal1_Row2->interpolate( tal1_Row2Col1,
            tal1_Row2Col2,
            my_detTopPtY[my_detIdxCol1],
            my_detTopPtY[my_detIdxCol2],
            my_detPosY,
            "lin" );
        tal1->interpolate( tal1_Row1,
            tal1_Row2,
            my_detTopPtX[my_detIdxRow1],
            my_detTopPtX[my_detIdxRow2],
            my_detPosX,
            "lin" );
    }
}


if ( newSrcIdx3 != srcIdx3 )
{
    srcIdx3 = newSrcIdx3;

    std::string talErgPath = getTallyEnergyPath(ergIdx3);
    std::string talPath;

    talPath = getTallyPath(ergIdx3,1,1);
    tal3_Row1Col1->parse( talErgPath,talPath );
    tal3_Row1Col1->setSourceEnergy( my_srcErg[ergIdx3] );

    talPath = getTallyPath(ergIdx3,1,2);
    tal3_Row1Col2->parse( talErgPath,talPath );
    tal3_Row1Col2->setSourceEnergy( my_srcErg[ergIdx3] );

    talPath = getTallyPath(ergIdx3,2,1);
    tal3_Row2Col1->parse( talErgPath,talPath );
    tal3_Row2Col1->setSourceEnergy( my_srcErg[ergIdx3] );

    talPath = getTallyPath(ergIdx3,2,2);
    tal3_Row2Col2->parse( talErgPath,talPath );
    tal3_Row2Col2->setSourceEnergy( my_srcErg[ergIdx3] );

    if ( my_detPosType == side )
    {
        tal3_Row1->interpolate( tal3_Row1Col1,
            tal3_Row1Col2,
            my_detSidePtY[my_detIdxCol1],
            my_detSidePtY[my_detIdxCol2],
            my_detPosY,
            "lin" );
        tal3_Row2->interpolate( tal3_Row2Col1,
            tal3_Row2Col2,
            my_detSidePtY[my_detIdxCol1],
            my_detSidePtY[my_detIdxCol2],
            my_detPosY,
            "lin" );
        tal3->interpolate( tal3_Row1,
            tal3_Row2,
            my_detSidePtZ[my_detIdxRow1],
            my_detSidePtZ[my_detIdxRow2],
            my_detPosZ,
            "lin" );
    }
    else if ( my_detPosType == top )
    {
```

```
            tal3_Row1->interpolate( tal3_Row1Col1,
                tal3_Row1Col2,
                my_detTopPtY[my_detIdxCol1],
                my_detTopPtY[my_detIdxCol2],
                my_detPosY,
                "lin" );
            tal3_Row2->interpolate( tal3_Row2Col1,
                tal3_Row2Col2,
                my_detTopPtY[my_detIdxCol1],
                my_detTopPtY[my_detIdxCol2],
                my_detPosY,
                "lin" );
            tal3->interpolate( tal3_Row1,
                tal3_Row2,
                my_detTopPtX[my_detIdxRow1],
                my_detTopPtX[my_detIdxRow2],
                my_detPosX,
                "lin" );
        }
    }

    // interpolate by source energy
    std::vector<double> peak;
    interpolator K;
    K.setSourceEnergies(
        my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);
    std::vector<datapoint> interpResult
        = K.interpolate( newerg,tal1,tal3,peak );

    return interpResult;
}


void background::initialize( )
{
    // keep tallies in scope outside of loop for efficiency
    // don't have to read files as often
    tal1_Row1Col1 = tallyPtr( new tally(my_redFact) );
    tal1_Row1Col2 = tallyPtr( new tally(my_redFact) );
    tal1_Row2Col1 = tallyPtr( new tally(my_redFact) );
    tal1_Row2Col2 = tallyPtr( new tally(my_redFact) );
    tal1_Row1 = tallyPtr( new tally(my_redFact) );
    tal1_Row2 = tallyPtr( new tally(my_redFact) );
    tal1 = tallyPtr( new tally(my_redFact) );

    tal3_Row1Col1 = tallyPtr( new tally(my_redFact) );
    tal3_Row1Col2 = tallyPtr( new tally(my_redFact) );
    tal3_Row2Col1 = tallyPtr( new tally(my_redFact) );
    tal3_Row2Col2 = tallyPtr( new tally(my_redFact) );
    tal3_Row1 = tallyPtr( new tally(my_redFact) );
    tal3_Row2 = tallyPtr( new tally(my_redFact) );
    tal3 = tallyPtr( new tally(my_redFact) );

    // set source indices to -1
    srcIdx1 = -1;
    srcIdx3 = -1;
}

void background::getDetectorPlane( )
{
    // hard-code in detector planes,
    // should read this in from file eventually
    const double tside = 129.54;
    const double ttop = 259.08;
    const double soff = 68.56;
    my_detXPlane = tside+soff; // side of truck + standoff
    my_detZPlane = ttop+soff; // side of truck + standoff

    // find out if this detector is a "side" or "top" detector
    if ( my_detPosZ < my_detZPlane && my_detPosX > tside )
    {
        my_detPosType = side;
        if ( fabs( my_detXPlane-my_detPosX ) > 0.1 )
        {
            // in future, need to add 1/r^2 correction factor
            throw fatal_error("side detector does not lie in x plane");
```

```
        }
    }
    else if ( my_detPosZ > ttop && my_detPosX < my_detXPlane )
    {
        my_detPosType = top;
        if ( fabs( my_detZPlane-my_detPosZ ) > 0.1 )
        {
            throw fatal_error("top detector does not lie in z plane");
        }
    }
    else
    {
        throw fatal_error("invalid detector position");
    }
}


void background::getDetectorIndices( )
{
    my_detIdxRow1 = 0;
    my_detIdxRow2 = 0;
    my_detIdxCol1 = 0;
    my_detIdxCol2 = 0;
    if ( my_detPosType == side )
    {
        const int Y = static_cast<int>( my_detSidePtY.size() );
        for ( int i=0;i<Y-1;++i )
        {
            if ( my_detPosY > my_detSidePtY[i]
                && my_detPosY < my_detSidePtY[i+1] )
            {
                my_detIdxCol1 = i;
                my_detIdxCol2 = i+1;
                break;
            }
        }
        const int Z = static_cast<int>( my_detSidePtZ.size() );
        for ( int i=0;i<Z-1;++i )
        {
            if ( my_detPosZ > my_detSidePtZ[i]
                && my_detPosZ < my_detSidePtZ[i+1] )
            {
                my_detIdxRow1 = i;
                my_detIdxRow2 = i+1;
                break;
            }
        }
    }
    else if ( my_detPosType == top )
    {
        const int Y = static_cast<int>( my_detTopPtY.size() );
        for ( int i=0;i<Y-1;++i )
        {
            if ( my_detPosY > my_detTopPtY[i]
                && my_detPosY < my_detTopPtY[i+1] )
            {
                my_detIdxCol1 = i;
                my_detIdxCol2 = i+1;
                break;
            }
        }
        const int X = static_cast<int>( my_detTopPtX.size() );
        for ( int i=0;i<X-1;++i )
        {
            if ( my_detPosX > my_detTopPtX[i]
                && my_detPosX < my_detTopPtX[i+1] )
            {
                my_detIdxRow1 = i;
                my_detIdxRow2 = i+1;
                break;
            }
        }
    }
}
```

```
void background::buildResponse ( double detPosX,
                                 double detPosY,
                                 double detPosZ,
                                 double maxErg,
                                 int redFact )
{
    //std::cout << "building background response" << std::endl;
    my_redFact = redFact;
    initialize( );

    // record detector position
    my_detPosX = detPosX;
    my_detPosY = detPosY;
    my_detPosZ = detPosZ;

    getDetectorPlane( );
    getDetectorIndices( );

    // build response function matrix
    computeResponse( maxErg );

    // make identity matrix
    my_I.resize( my_R.numergout(),my_R.numergin() );
    my_I.ergout() = my_R.ergout();
    my_I.ergin() = my_R.ergin();
    my_I.identity();

    // tallies are already divided by
    // detector area so don't need to do it again
    //my_R = my_R * ( 1.0/my_detArea[my_detAreaIdx] );

    //std::cout << "finished building background response" << std::endl;

    return;
}

spectrum background::operator() ( const spectrum& S )
{
    return my_R * S;
}

spectrum background::operator() ( const dspectrum& S )
{
    return my_R * S;
}
```

Listing B.4: background.hpp

```
#ifndef _background_hpp_included_
#define _background_hpp_included_
#include <string>
#include <vector>

#include "spectrum.hpp"
#include "dspectrum.hpp"
#include "response.hpp"
#include "submodel.hpp"



class background : public submodel
{
public:
    enum dpos { side,top };

    background ( const std::string& );
    background ( ) { };



    void buildResponse ( double detPosX,
                         double detPosY,
                         double detPosZ,
                         double maxErg,
                         int redFact );

    spectrum operator() ( const spectrum& );
    spectrum operator() ( const dspectrum& );

    virtual void initialize( const std::string& );

    void addResponse( const response& R,double frac )
    {
        my_R = my_R + R*frac;
    }

protected:


    virtual void readDataFile();
    std::string getTallyEnergyPath( int );
    virtual std::vector<datapoint>
        interpolateTallies(
        double,const std::vector< double >&,int,int );
    virtual void initialize ( );
    void getDetectorPlane( );
    void getDetectorIndices( );

    int numInErgs ( );
    int numInErgs ( ) const;
    int numOutErgs ( );
    int numOutErgs ( ) const;



    response my_normR;

    response my_I;

    double my_detPosX;
    double my_detPosY;
    double my_detPosZ;

    double my_detXPlane;
    double my_detZPlane;

    std::vector< double > my_detSidePtX;
    std::vector< double > my_detSidePtY;
    std::vector< double > my_detSidePtZ;
    std::vector< double > my_detSideArea;
    std::vector< double > my_detTopPtX;
    std::vector< double > my_detTopPtY;
    std::vector< double > my_detTopPtZ;
```

```
    std::vector< double > my_detTopArea;
    int my_detIdxRow1;
    int my_detIdxRow2;
    int my_detIdxCol1;
    int my_detIdxCol2;

    std::vector<int> my_detPosIdx;
    std::vector<double> my_detDistance;

    dpos my_detPosType;

    int srcIdx1;
    int srcIdx3;

private:

    tallyPtr tal1_Row1Col1;
    tallyPtr tal1_Row2Col1;
    tallyPtr tal1_Row1Col2;
    tallyPtr tal1_Row2Col2;
    tallyPtr tal1_Row1;
    tallyPtr tal1_Row2;
    tallyPtr tal1;

    tallyPtr tal3_Row1Col1;
    tallyPtr tal3_Row2Col1;
    tallyPtr tal3_Row1Col2;
    tallyPtr tal3_Row2Col2;
    tallyPtr tal3_Row1;
    tallyPtr tal3_Row2;
    tallyPtr tal3;

    std::string getTallyPath( int,int,int );

};

#endif
```

Listing B.5: cargo.cpp

```
#include "cargo.hpp"
#include "extras.hpp"
#include "interpolation.hpp"

#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>
#include "fileio.hpp"

#include <phys.hpp>

void cargo::readDataFile( )
{
    my_srcPtX = readbin( my_datapath+"srcposx.dat" );
    my_srcPtY = readbin( my_datapath+"srcposy.dat" );
    my_srcPtZ = readbin( my_datapath+"srcposz.dat" );
    return;
}

cargo::cargo ( const std::string& path )
{
    initialize(path);
}

void cargo::initialize( const std::string& path )
{
    my_datapath = path + sep();
    background::readDataFile( );
    cargo::readDataFile( );
}

std::string cargo::getTallyPath( int ergIdx,int detRowIdx,
    int detColIdx,int srcIdxX,int srcIdxY,int srcIdxZ )
{
    int detRow,detCol;
    if ( detRowIdx == 1 )
        detRow = my_detIdxRow1;
    else /*if ( detRowIdx == 2 )*/
        detRow = my_detIdxRow2;
    if ( detColIdx == 1 )
        detCol = my_detIdxCol1;
    else /*if ( detColIdx == 2 )*/
        detCol = my_detIdxCol2;
    int srcX,srcY,srcZ;
    if ( srcIdxX == 1 )
        srcX = my_srcIdxX1;
    else /*if ( srcIdxX == 2 )*/
        srcX = my_srcIdxX2;
    if ( srcIdxY == 1 )
        srcY = my_srcIdxY1;
    else /*if ( srcIdxY == 2 )*/
        srcY = my_srcIdxY2;
    if ( srcIdxZ == 1 )
        srcZ = my_srcIdxZ1;
    else /*if ( srcIdxZ == 2 )*/
        srcZ = my_srcIdxZ2;


    if ( my_detPosType == side )
    {
        return my_datapath + "erg" + str(ergIdx)
            + sep() + "side" + sep() + "dpos"
            + str(detRow) + "_" + str(detCol)
            + sep() + "spos" + str(srcX) + "_"
            + str(srcY) + "_" + str(srcZ) + sep();
    }
    else /*if ( my_detPosType == top )*/
    {
        return my_datapath + "erg" + str(ergIdx)
            + sep() + "top" + sep() + "dpos"
            + str(detRow) + "_" + str(detCol)
            + sep() + "spos" + str(srcX) + "_"
            + str(srcY) + "_" + str(srcZ) + sep();
```

```
    }
}

cargo::tallyPtr cargo::interpolateSourceXZ(
    tallyPtr talx1z1,tallyPtr talx1z2,
    tallyPtr talx2z1,tallyPtr talx2z2 )
{
    tallyPtr talx1 = tallyPtr( new tally(my_redFact) );
    tallyPtr talx2 = tallyPtr( new tally(my_redFact) );
    tallyPtr result = tallyPtr( new tally(my_redFact) );



    talx1->interpolate( talx1z1,
        talx1z2,
        my_srcPtZ[my_srcIdxZ1],
        my_srcPtZ[my_srcIdxZ2],
        my_srcPosZ,
        "lin" );


    talx2->interpolate( talx2z1,
        talx2z2,
        my_srcPtZ[my_srcIdxZ1],
        my_srcPtZ[my_srcIdxZ2],
        my_srcPosZ,
        "lin" );

    result->interpolate( talx1,
        talx2,
        my_srcPtX[my_srcIdxX1],
        my_srcPtX[my_srcIdxX2],
        my_srcPosX,
        "lin" );
    return result;
}

cargo::tallyPtr cargo::interpolateDetectorRow(
    tallyPtr talRow1,tallyPtr talRow2 )
{
    tallyPtr result = tallyPtr( new tally(my_redFact) );

    if ( my_detPosType == side )
    {
        result->interpolate( talRow1,
            talRow2,
            my_detSidePtZ[my_detIdxRow1],
            my_detSidePtZ[my_detIdxRow2],
            my_detPosZ,
            "lin" );
    }
    else /*if ( my_detPosType == top )*/
    {
        result->interpolate( talRow1,
            talRow2,
            my_detTopPtX[my_detIdxRow1],
            my_detTopPtX[my_detIdxRow2],
            my_detPosX,
            "lin" );
    }
    return result;
}

cargo::tallyPtr cargo::interpolateDetectorsAndSources( int ergIdx )
{
    std::string talErgPath = getTallyEnergyPath(ergIdx);
    std::string talPath;

    tallyPtr tal_r1c1_x1y1z1 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,1,1,1,1,1 );
    tal_r1c1_x1y1z1->parse( talErgPath,talPath );
    tal_r1c1_x1y1z1->setSourceEnergy( my_srcErg[ergIdx] );

    tallyPtr tal_r1c1_x1y1z2 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,1,1,1,1,2 );
    tal_r1c1_x1y1z2->parse( talErgPath,talPath );
```

```
tal_r1c1_x1y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x1y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,1,2,1 );
tal_r1c1_x1y2z1->parse( talErgPath,talPath );
tal_r1c1_x1y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x1y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,1,2,2 );
tal_r1c1_x1y2z2->parse( talErgPath,talPath );
tal_r1c1_x1y2z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x2y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,2,1,1 );
tal_r1c1_x2y1z1->parse( talErgPath,talPath );
tal_r1c1_x2y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x2y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,2,1,2 );
tal_r1c1_x2y1z2->parse( talErgPath,talPath );
tal_r1c1_x2y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x2y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,2,2,1 );
tal_r1c1_x2y2z1->parse( talErgPath,talPath );
tal_r1c1_x2y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_x2y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,1,2,2,2 );
tal_r1c1_x2y2z2->parse( talErgPath,talPath );
tal_r1c1_x2y2z2->setSourceEnergy( my_srcErg[ergIdx] );


tallyPtr tal_r1c2_x1y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,1,1,1 );
tal_r1c2_x1y1z1->parse( talErgPath,talPath );
tal_r1c2_x1y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x1y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,1,1,2 );
tal_r1c2_x1y1z2->parse( talErgPath,talPath );
tal_r1c2_x1y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x1y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,1,2,1 );
tal_r1c2_x1y2z1->parse( talErgPath,talPath );
tal_r1c2_x1y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x1y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,1,2,2 );
tal_r1c2_x1y2z2->parse( talErgPath,talPath );
tal_r1c2_x1y2z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x2y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,2,1,1 );
tal_r1c2_x2y1z1->parse( talErgPath,talPath );
tal_r1c2_x2y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x2y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,2,1,2 );
tal_r1c2_x2y1z2->parse( talErgPath,talPath );
tal_r1c2_x2y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x2y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,2,2,1 );
tal_r1c2_x2y2z1->parse( talErgPath,talPath );
tal_r1c2_x2y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c2_x2y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,1,2,2,2,2 );
tal_r1c2_x2y2z2->parse( talErgPath,talPath );
tal_r1c2_x2y2z2->setSourceEnergy( my_srcErg[ergIdx] );
```

```
tallyPtr tal_r2c1_x1y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,1,1,1 );
tal_r2c1_x1y1z1->parse( talErgPath,talPath );
tal_r2c1_x1y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x1y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,1,1,2 );
tal_r2c1_x1y1z2->parse( talErgPath,talPath );
tal_r2c1_x1y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x1y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,1,2,1 );
tal_r2c1_x1y2z1->parse( talErgPath,talPath );
tal_r2c1_x1y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x1y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,1,2,2 );
tal_r2c1_x1y2z2->parse( talErgPath,talPath );
tal_r2c1_x1y2z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x2y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,2,1,1 );
tal_r2c1_x2y1z1->parse( talErgPath,talPath );
tal_r2c1_x2y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x2y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,2,1,2 );
tal_r2c1_x2y1z2->parse( talErgPath,talPath );
tal_r2c1_x2y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x2y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,2,2,1 );
tal_r2c1_x2y2z1->parse( talErgPath,talPath );
tal_r2c1_x2y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c1_x2y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,1,2,2,2 );
tal_r2c1_x2y2z2->parse( talErgPath,talPath );
tal_r2c1_x2y2z2->setSourceEnergy( my_srcErg[ergIdx] );



tallyPtr tal_r2c2_x1y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,1,1,1 );
tal_r2c2_x1y1z1->parse( talErgPath,talPath );
tal_r2c2_x1y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x1y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,1,1,2 );
tal_r2c2_x1y1z2->parse( talErgPath,talPath );
tal_r2c2_x1y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x1y2z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,1,2,1 );
tal_r2c2_x1y2z1->parse( talErgPath,talPath );
tal_r2c2_x1y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x1y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,1,2,2 );
tal_r2c2_x1y2z2->parse( talErgPath,talPath );
tal_r2c2_x1y2z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x2y1z1 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,2,1,1 );
tal_r2c2_x2y1z1->parse( talErgPath,talPath );
tal_r2c2_x2y1z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x2y1z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,2,1,2 );
tal_r2c2_x2y1z2->parse( talErgPath,talPath );
tal_r2c2_x2y1z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x2y2z1 = tallyPtr( new tally(my_redFact) );
```

```
talPath = getTallyPath( ergIdx,2,2,2,2,1 );
tal_r2c2_x2y2z1->parse( talErgPath,talPath );
tal_r2c2_x2y2z1->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r2c2_x2y2z2 = tallyPtr( new tally(my_redFact) );
talPath = getTallyPath( ergIdx,2,2,2,2,2 );
tal_r2c2_x2y2z2->parse( talErgPath,talPath );
tal_r2c2_x2y2z2->setSourceEnergy( my_srcErg[ergIdx] );

tallyPtr tal_r1c1_s1 = tallyPtr( new tally(my_redFact) );
tal_r1c1_s1 = interpolateSourceXZ( tal_r1c1_x1y1z1,
    tal_r1c1_x1y1z2,tal_r1c1_x2y1z1,tal_r1c1_x2y1z2 );

tallyPtr tal_r1c1_s2 = tallyPtr( new tally(my_redFact) );
tal_r1c1_s2 = interpolateSourceXZ( tal_r1c1_x1y2z1,
    tal_r1c1_x1y2z2,tal_r1c1_x2y2z1,tal_r1c1_x2y2z2 );

tallyPtr tal_r1c2_s1 = tallyPtr( new tally(my_redFact) );
tal_r1c2_s1 = interpolateSourceXZ( tal_r1c2_x1y1z1,
    tal_r1c2_x1y1z2,tal_r1c2_x2y1z1,tal_r1c2_x2y1z2 );

tallyPtr tal_r1c2_s2 = tallyPtr( new tally(my_redFact) );
tal_r1c2_s2 = interpolateSourceXZ( tal_r1c2_x1y2z1,
    tal_r1c2_x1y2z2,tal_r1c2_x2y2z1,tal_r1c2_x2y2z2 );

tallyPtr tal_r2c1_s1 = tallyPtr( new tally(my_redFact) );
tal_r2c1_s1 = interpolateSourceXZ( tal_r2c1_x1y1z1,
    tal_r2c1_x1y1z2,tal_r2c1_x2y1z1,tal_r2c1_x2y1z2 );

tallyPtr tal_r2c1_s2 = tallyPtr( new tally(my_redFact) );
tal_r2c1_s2 = interpolateSourceXZ( tal_r2c1_x1y2z1,
    tal_r2c1_x1y2z2,tal_r2c1_x2y2z1,tal_r2c1_x2y2z2 );

tallyPtr tal_r2c2_s1 = tallyPtr( new tally(my_redFact) );
tal_r2c2_s1 = interpolateSourceXZ( tal_r2c2_x1y1z1,
    tal_r2c2_x1y1z2,tal_r2c2_x2y1z1,tal_r2c2_x2y1z2 );

tallyPtr tal_r2c2_s2 = tallyPtr( new tally(my_redFact) );
tal_r2c2_s2 = interpolateSourceXZ( tal_r2c2_x1y2z1,
    tal_r2c2_x1y2z2,tal_r2c2_x2y2z1,tal_r2c2_x2y2z2 );

tallyPtr tal_d1s1 = tallyPtr( new tally(my_redFact) );
tal_d1s1 = interpolateDetectorRow( tal_r1c1_s1,tal_r2c1_s1 );

tallyPtr tal_d1s2 = tallyPtr( new tally(my_redFact) );
tal_d1s2 = interpolateDetectorRow( tal_r1c1_s2,tal_r2c1_s2 );

tallyPtr tal_d2s1 = tallyPtr( new tally(my_redFact) );
tal_d2s1 = interpolateDetectorRow( tal_r1c2_s1,tal_r2c2_s1 );

tallyPtr tal_d2s2 = tallyPtr( new tally(my_redFact) );
tal_d2s2 = interpolateDetectorRow( tal_r1c2_s2,tal_r2c2_s2 );


tallyPtr talds = tallyPtr( new tally(my_redFact) );
if ( my_intrp == normal )
{
    tallyPtr tal_d1s = tallyPtr( new tally(my_redFact) );
    tal_d1s->interpolate( tal_d1s1,
        tal_d1s2,
        my_srcPtY[my_srcIdxY1],
        my_srcPtY[my_srcIdxY2],
        my_srcPosY,
        "lin" );

    tallyPtr tal_d2s = tallyPtr( new tally(my_redFact) );
    tal_d2s->interpolate( tal_d2s1,
        tal_d2s2,
        my_srcPtY[my_srcIdxY1],
        my_srcPtY[my_srcIdxY2],
        my_srcPosY,
        "lin" );
    if ( my_detPosType == side )
    {
        talds->interpolate( tal_d1s,
            tal_d2s,
```

```
            my_detSidePtY[my_detIdxCol1],
            my_detSidePtY[my_detIdxCol2],
            my_detPosY,
            "lin");
    }
    else /*if ( my_detPosType == top )*/
    {
        talds->interpolate( tal_d1s,
            tal_d2s,
            my_detTopPtY[my_detIdxCol1],
            my_detTopPtY[my_detIdxCol2],
            my_detPosY,
            "lin");
    }
}
else /*if ( my_intrp == peak )*/
{
    tallyPtr tal_dsp = tallyPtr( new tally(my_redFact) );
    tal_dsp->interpolate( tal_d1s1,
        tal_d2s2,
        my_srcPtY[my_srcIdxY1],
        my_srcPtY[my_srcIdxY2],
        my_srcPosY,
        "lin" );

    if ( my_srcPosY > my_detPosY )
    {
        if ( my_detPosType == side )
        {
            tallyPtr tal_ds2 = tallyPtr( new tally(my_redFact) );
            tal_ds2->interpolate( tal_d1s2,
                tal_d2s2,
                my_detSidePtY[my_detIdxCol1],
                my_detSidePtY[my_detIdxCol2],
                my_detPosY,
                "lin" );

            talds->interpolate( tal_dsp,
                tal_ds2,
                my_detPosY,
                my_srcPtY[my_srcIdxY2],
                my_srcPosY,
                "lin" );
        }
        else /*if ( my_detPosType == top )*/
        {
            tallyPtr tal_ds2 = tallyPtr( new tally(my_redFact) );
            tal_ds2->interpolate( tal_d1s2,
                tal_d2s2,
                my_detTopPtY[my_detIdxCol1],
                my_detTopPtY[my_detIdxCol2],
                my_detPosY,
                "lin" );

            talds->interpolate( tal_dsp,
                tal_ds2,
                my_detPosY,
                my_srcPtY[my_srcIdxY2],
                my_srcPosY,
                "lin" );
        }
    }
    else /*if ( my_srcPosY < my_detPosY )*/
    {
        if ( my_detPosType == side )
        {
            tallyPtr tal_ds1 = tallyPtr( new tally(my_redFact) );
            tal_ds1->interpolate( tal_d1s1,
                tal_d2s1,
                my_detSidePtY[my_detIdxCol1],
                my_detSidePtY[my_detIdxCol2],
                my_detPosY,
                "lin" );

            talds->interpolate( tal_dsp,
                tal_ds1,
```

```
                    my_detPosY,
                    my_srcPtY[my_srcIdxY1],
                    my_srcPosY,
                    "lin" );
            }
            else /*if ( my_detPosType == top )*/
            {
                tallyPtr tal_ds1 = tallyPtr( new tally(my_redFact) );
                tal_ds1->interpolate( tal_d1s1,
                    tal_d2s1,
                    my_detTopPtY[my_detIdxCol1],
                    my_detTopPtY[my_detIdxCol2],
                    my_detPosY,
                    "lin" );

                talds->interpolate( tal_dsp,
                    tal_ds1,
                    my_detPosY,
                    my_srcPtY[my_srcIdxY1],
                    my_srcPosY,
                    "lin" );
            }
        }
    }


    return talds;
}


std::vector<datapoint> cargo::interpolateTallies(
    double newpeak,const std::vector< double >& newerg,
    int ergIdx1,int ergIdx3 )
{
    // get indices
    const int newSrcIdx1 = ergIdx1;
    const int newSrcIdx3 = ergIdx3;

    // parse tallies and interpolate

    if ( newSrcIdx1 != srcIdx1 )
    {
        srcIdx1 = newSrcIdx1;
        tal1 = interpolateDetectorsAndSources( ergIdx1 );
        //if ( ergIdx1 == 37 )
        //{
        //  tal1->print( );
        //}
    }


    if ( newSrcIdx3 != srcIdx3 )
    {
        srcIdx3 = newSrcIdx3;
        tal3 = interpolateDetectorsAndSources( ergIdx3 );
    }

    // interpolate by source energy
    std::vector<double> peak;
    interpolator K;
    K.setSourceEnergies(
        my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);
    std::vector<datapoint> interpResult
       = K.interpolate( newerg,tal1,tal3,peak );
    //std::cout << "done" << std::endl;
    return interpResult;
}

void cargo::initialize( )
{
    // keep tallies in scope outside of loop for efficiency
    // don't have to read files as often
    tal1 = tallyPtr( new tally(my_redFact) );
    tal3 = tallyPtr( new tally(my_redFact) );

    // initialize source indices to -1
```

```
    srcIdx1 = -1;
    srcIdx3 = -1;
}


void cargo::buildResponse ( double srcXPos,double srcYPos,
    double srcZPos,double detXPos,double detYPos,
    double detZPos,double maxErg,int redFact )
{
    //std::cout << "building cargo response" << std::endl;
    my_redFact = redFact;

    initialize( );

    // store detector positions
    my_detPosX = detXPos;
    my_detPosY = detYPos;
    my_detPosZ = detZPos;


    getDetectorPlane( );

    // record source position
    my_srcPosX = srcXPos;
    my_srcPosY = srcYPos;
    my_srcPosZ = srcZPos;


    getDetectorIndices( );

    getSourceIndices( );

    getDetectorSourceIndices( );

    // determine interpolation type
    //
    // if the detector position is bounded by
    // the closest source positions, use peak method
    if ( my_detPosY < my_srcPtY[my_srcIdxY2]
        && my_detPosY > my_srcPtY[my_srcIdxY1] )
    {
        my_intrp = peak;
    }
    else
    {
        my_intrp = normal;
    }

    // build response function matrix
    computeResponse( maxErg );

    // make identity matrix
    //my_I.resize( my_R.numergout(),my_R.numergin() );
    //my_I.ergout() = my_R.ergout();
    //my_I.ergin() = my_R.ergin();
    //my_I.identity();
    // divide identity by area of detector face
    //std::vector<double> d2s(3);
    //d2s[0] = srcXPos - detXPos;
    //d2s[1] = srcYPos - detYPos;
    //d2s[2] = srcZPos - detZPos;

    // tallies are already divided
    // by the area of the detectors, so don't need to do it again
    //my_R = my_R * (1.0-omegaStream)
    //  +  my_I * ( (omegaX+omegaY+omegaZ)/(4*pi) * omegaStream );
    //my_R = my_R * (1.0-omegaStream)
    //  +  my_I * ( (1.0/(4*phys::pi*pow(mag(d2s),2.0))) * omegaStream );

    //std::cout << "finished building cargo response" << std::endl;
}


void cargo::buildNoVehicle( double srcXPos,
                            double srcYPos,
```

```
                             double srcZPos,
                             double detXPos,
                             double detYPos,
                             double detZPos,
                             const spectrum& S )
{
    //std::cout << "building no vehicle response" << std::endl;

    my_R.initialize( S.erg(),S.erg() );
    my_R.identity();

    std::vector<double> d2s(3);
    d2s[0] = srcXPos - detXPos;
    d2s[1] = srcYPos - detYPos;
    d2s[2] = srcZPos - detZPos;

    my_R = my_R * (1.0/(4*phys::pi*pow(mag(d2s),2.0)));

    //std::cout << "finished building no vehicle response" << std::endl;

    return;
}


void cargo::getSourceIndices( )
{
    my_srcIdxX1 = 0;
    my_srcIdxX2 = 0;
    my_srcIdxY1 = 0;
    my_srcIdxY2 = 0;
    my_srcIdxZ1 = 0;
    my_srcIdxZ2 = 0;

    const int X = static_cast<int>( my_srcPtX.size() );
    for ( int i=0;i<X-1;++i )
    {
        if ( my_srcPosX >= my_srcPtX[i]
            && my_srcPosX < my_srcPtX[i+1] )
        {
            my_srcIdxX1 = i;
            my_srcIdxX2 = i+1;
            break;
        }
    }
    const int Y =
        static_cast<int>( my_srcPtY.size() );
    for ( int i=0;i<Y-1;++i )
    {
        if ( my_srcPosY >= my_srcPtY[i]
            && my_srcPosY < my_srcPtY[i+1] )
        {
            my_srcIdxY1 = i;
            my_srcIdxY2 = i+1;
            break;
        }
    }
    const int Z =
        static_cast<int>( my_srcPtZ.size() );
    for ( int i=0;i<Z-1;++i )
    {
        if ( my_srcPosZ >= my_srcPtZ[i]
            && my_srcPosZ < my_srcPtZ[i+1] )
        {
            my_srcIdxZ1 = i;
            my_srcIdxZ2 = i+1;
            break;
        }
    }



}

void cargo::getDetectorSourceIndices( )
{
    // if we need to worry about the peak interpolation
```

```
    // algorithm, use only detector indices which line
    // up with source
    if ( my_intrp == peak )
    {
        if ( my_detPosType == side )
        {
            const int Y =
                static_cast<int>( my_detSidePtY.size() );
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detSidePtY[i]
                    - my_srcPtY[my_srcIdxY1] ) < 0.1 )
                {
                    my_detIdxCol1 = i;
                    break;
                }
            }
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detSidePtY[i]
                    - my_srcPtY[my_srcIdxY2] ) < 0.1 )
                {
                    my_detIdxCol2 = i;
                    break;
                }
            }
        }
        else if ( my_detPosType == top )
        {
            const int Y =
                static_cast<int>( my_detTopPtY.size() );
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detTopPtY[i]
                    - my_srcPtY[my_srcIdxY1] ) < 0.1 )
                {
                    my_detIdxCol1 = i;
                    break;
                }
            }
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detTopPtY[i]
                    - my_srcPtY[my_srcIdxY2] ) < 0.1 )
                {
                    my_detIdxCol2 = i;
                    break;
                }
            }
        }
    }
}


cargonorm::cargonorm ( const std::string& path )
{
    initialize(path);
}

void cargonorm::initialize( const std::string& path )
{
    my_datapath = path + sep();
    background::readDataFile( );
}

std::string cargonorm::getTallyPath( int ergIdx,
    int detRowIdx,int detColIdx )
{
    int detRow,detCol;
    if ( detRowIdx == 1 )
        detRow = my_detIdxRow1;
    else /*if ( detRowIdx == 2 )*/
        detRow = my_detIdxRow2;
    if ( detColIdx == 1 )
        detCol = my_detIdxCol1;
```

```
    else /*if ( detColIdx == 2 )*/
        detCol = my_detIdxCol2;


    if ( my_detPosType == side )
    {
        return my_datapath + "erg" + str(ergIdx) + sep()
        + "side" + sep() + "dpos" + str(detRow) + "_"
        + str(detCol) + sep();
    }
    else /*if ( my_detPosType == top )*/
    {
        return my_datapath + "erg" + str(ergIdx) + sep()
        + "top" + sep() + "dpos" + str(detRow) + "_"
        + str(detCol) + sep();
    }
}

std::vector<datapoint> cargonorm::interpolateTallies(
    double newpeak,const std::vector< double >& newerg,
    int ergIdx1,int ergIdx3 )
{
    // get indices
    const int newSrcIdx1 = ergIdx1;
    const int newSrcIdx3 = ergIdx3;

    // parse tallies and interpolate
    if ( newSrcIdx1 != srcIdx1 )
    {
        int ergIdx = newSrcIdx1;

        std::string talErgPath = getTallyEnergyPath(ergIdx);
        std::string talPath;

        tallyPtr tal_r1c1 = tallyPtr( new tally(my_redFact) );
        talPath = getTallyPath( ergIdx,1,1 );
        tal_r1c1->parse( talErgPath,talPath );
        tal_r1c1->setSourceEnergy( my_srcErg[ergIdx] );


        tallyPtr tal_r1c2 = tallyPtr( new tally(my_redFact) );
        talPath = getTallyPath( ergIdx,1,2 );
        tal_r1c2->parse( talErgPath,talPath );
        tal_r1c2->setSourceEnergy( my_srcErg[ergIdx] );

        tallyPtr tal_r2c1 = tallyPtr( new tally(my_redFact) );
        talPath = getTallyPath( ergIdx,2,1 );
        tal_r2c1->parse( talErgPath,talPath );
        tal_r2c1->setSourceEnergy( my_srcErg[ergIdx] );


        tallyPtr tal_r2c2 = tallyPtr( new tally(my_redFact) );
        talPath = getTallyPath( ergIdx,2,2 );
        tal_r2c2->parse( talErgPath,talPath );
        tal_r2c2->setSourceEnergy( my_srcErg[ergIdx] );

        tallyPtr tal_c1 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal_c2 = tallyPtr( new tally(my_redFact) );
        if ( my_detPosType == side )
        {
            tal_c1->interpolate( tal_r1c1,tal_r2c1,
                my_detSidePtZ[my_detIdxRow1],
                my_detSidePtZ[my_detIdxRow2],
                my_detPosZ,"lin" );

            tal_c2->interpolate( tal_r1c2,tal_r2c2,
                my_detSidePtZ[my_detIdxRow1],
                my_detSidePtZ[my_detIdxRow2],
                my_detPosZ,"lin" );


        }
        else /*if ( my_detPosType == top )*/
        {
            tal_c1->interpolate( tal_r1c1,tal_r2c1,
                my_detTopPtX[my_detIdxRow1],
```

```
            my_detTopPtX[my_detIdxRow2],
            my_detPosX,"lin" );

        tal_c2->interpolate( tal_r1c2,tal_r2c2,
            my_detTopPtX[my_detIdxRow1],
            my_detTopPtX[my_detIdxRow2],
            my_detPosX,"lin" );
    }
    if ( my_detPosType == side )
    {
        tal1->interpolate( tal_c1,tal_c2,
            my_detSidePtY[my_detIdxCol1],
            my_detSidePtY[my_detIdxCol2],
            my_detPosY,"lin");
    }
    else /*if ( my_detPosType == top )*/
    {
        tal1->interpolate( tal_c1,tal_c2,
            my_detTopPtY[my_detIdxCol1],
            my_detTopPtY[my_detIdxCol2],
            my_detPosY,"lin");
    }
    srcIdx1 = newSrcIdx1;
}


if ( newSrcIdx3 != srcIdx3 )
{
    int ergIdx = newSrcIdx3;

    //std::cout << "ergIdx3 = " << ergIdx << std::endl;


    std::string talErgPath = getTallyEnergyPath(ergIdx);
    std::string talPath;



    tallyPtr tal_r1c1 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,1,1 );
    tal_r1c1->parse( talErgPath,talPath );
    tal_r1c1->setSourceEnergy( my_srcErg[ergIdx] );


    tallyPtr tal_r1c2 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,1,2 );
    tal_r1c2->parse( talErgPath,talPath );
    tal_r1c2->setSourceEnergy( my_srcErg[ergIdx] );


    tallyPtr tal_r2c1 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,2,1 );
    tal_r2c1->parse( talErgPath,talPath );
    tal_r2c1->setSourceEnergy( my_srcErg[ergIdx] );


    tallyPtr tal_r2c2 = tallyPtr( new tally(my_redFact) );
    talPath = getTallyPath( ergIdx,2,2 );
    tal_r2c2->parse( talErgPath,talPath );
    tal_r2c2->setSourceEnergy( my_srcErg[ergIdx] );


    tallyPtr tal_c1 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal_c2 = tallyPtr( new tally(my_redFact) );
    if ( my_detPosType == side )
    {
        tal_c1->interpolate( tal_r1c1,tal_r2c1,
            my_detSidePtZ[my_detIdxRow1],
            my_detSidePtZ[my_detIdxRow2],
            my_detPosZ,"lin" );

        tal_c2->interpolate( tal_r1c2,tal_r2c2,
            my_detSidePtZ[my_detIdxRow1],
            my_detSidePtZ[my_detIdxRow2],
            my_detPosZ,"lin" );
    }
```

```
        else /*if ( my_detPosType == top )*/
        {
            tal_c1->interpolate( tal_r1c1,tal_r2c1,
                my_detTopPtX[my_detIdxRow1],
                my_detTopPtX[my_detIdxRow2],
                my_detPosX,"lin" );

            tal_c2->interpolate( tal_r1c2,tal_r2c2,
                my_detTopPtX[my_detIdxRow1],
                my_detTopPtX[my_detIdxRow2],
                my_detPosX,"lin" );
        }
        if ( my_detPosType == side )
        {
            tal3->interpolate( tal_c1,tal_c2,
                my_detSidePtY[my_detIdxCol1],
                my_detSidePtY[my_detIdxCol2],
                my_detPosY,"lin");
        }
        else /*if ( my_detPosType == top )*/
        {
            tal3->interpolate( tal_c1,tal_c2,
                my_detTopPtY[my_detIdxCol1],
                my_detTopPtY[my_detIdxCol2],
                my_detPosY,"lin");
        }
        srcIdx3 = newSrcIdx3;
    }



    // interpolate by source energy
    std::vector<double> peak;
    interpolator K;
    K.setSourceEnergies(
        my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);
    std::vector<datapoint> interpResult
        = K.interpolate( newerg,tal1,tal3,peak );
    //std::cout << interpResult << std::endl;
    //std::cout << "done" << std::endl;

    return interpResult;
}


void cargonorm::buildResponse ( double detXPos,double detYPos,
    double detZPos,double maxErg,int redFact )
{
    //std::cout << "building norm cargo response" << std::endl;
    my_redFact = redFact;

    cargo::initialize( );

    // store detector positions
    my_detPosX = detXPos;
    my_detPosY = detYPos;
    my_detPosZ = detZPos;


    getDetectorPlane( );

    getDetectorIndices( );

    // build response function matrix
    computeResponse( maxErg );

    //std::cout << "finished building norm cargo response" << std::endl;
}
```

Listing B.6: cargo.hpp

```
#ifndef _cargo_hpp_included_
#define _cargo_hpp_included_

#include "background.hpp"


class cargo : public background
{
    public:

    enum interpolationType { peak,normal };

    cargo ( const std::string& );
    cargo ( ) { };

    void initialize( const std::string& );

    void buildResponse( double srcPosX,double srcPosY,
        double srcPosZ,double detPosX,double detPosY,
        double detPosZ,double maxErg,int redFact );

    void buildNoVehicle( double,double,double,double,double,
        double,const spectrum& );



    protected:

    interpolationType my_intrp;

    void initialize( );
    void readDataFile();
    virtual std::vector<datapoint> interpolateTallies(
        double,const std::vector< double >&,int,int );


    std::string getTallyPath( int,int,int,int,int,int );
    tallyPtr interpolateSourceXZ( tallyPtr talx1z1,
        tallyPtr talx1z2,tallyPtr talx2z1,tallyPtr talx2z2 );
    tallyPtr interpolateDetectorRow( tallyPtr talRow1,
        tallyPtr talRow2 );
    tallyPtr interpolateDetectorsAndSources( int ergIdx );
    void getSourceIndices( );
    void getDetectorSourceIndices( );

    double my_srcPosX;
    double my_srcPosY;
    double my_srcPosZ;

    double my_det2SrcDistance;

    std::vector< double > my_srcPtX;
    std::vector< double > my_srcPtY;
    std::vector< double > my_srcPtZ;

    int my_srcIdxX1;
    int my_srcIdxX2;
    int my_srcIdxY1;
    int my_srcIdxY2;
    int my_srcIdxZ1;
    int my_srcIdxZ2;

    tallyPtr tal1;
    tallyPtr tal3;
};


class cargonorm : public cargo
{
    public:

    cargonorm ( const std::string& path );
```

```
    cargonorm ( ) { };

    void initialize( const std::string& path );

    void buildResponse( double detPosX, double detPosY,
        double detPosZ, double maxErg, int redFact );

    private:

    std::string getTallyPath( int ergIdx,
        int detRowIdx, int detColIdx );

    std::vector<datapoint> interpolateTallies(
        double, const std::vector< double >&, int, int );

};

#endif
```

Listing B.7: data.cpp

```
#include <fstream>
#include <math.h>
#include <algorithm>
#include <iostream>
#include "extras.hpp"
#include "data.hpp"
#include "errh.hpp"
#include "fileio.hpp"

namespace input
{
    using namespace std;

    bool w2b( const string& word )
    {
        if ( word.compare("yes") == 0 || word.compare("on") == 0)
        {
            return true;
        }
        else if ( word.compare("no") == 0 || word.compare("off") == 0)
        {
            return false;
        }
        return true;
    }

    vector<block> block::getBlocks( const string& keyword )
    {
        vector<block> blocks;
        while ( true )
        {
            block tempblock = getBlock(keyword);
            if ( ! tempblock.empty() )
            {
                blocks.push_back(tempblock);
            }
            else
            {
                break;
            }
        }
        return blocks;
    }

    template< class T >
    bool block::getData ( const std::string& keyword,T& val )
    {
        for ( int i=0;i<nLine();++i )
        {
            size_t fpos = line[i].find(keyword);
            if ( fpos != string::npos )
            {
                size_t size = keyword.size();
                stringstream ss;
                ss << line[i];
                ss.seekg( fpos+size );
                ss >> val;
                return true;
            }
        }
        return false;
    }

    template< class T >
    T block::getData ( const std::string& keyword )
    {
        T result;
        for ( int i=0;i<nLine();++i )
        {
            size_t fpos = line[i].find(keyword);
            if ( fpos != string::npos )
            {
                size_t size = keyword.size();
                stringstream ss;
                ss << line[i];
```

```
            ss.seekg( fpos+size );
            ss >> result;
            return result;
        }
    }
    throw fatal_error("did not find keyword \""
        +keyword+"\" in block \""+name+"\"");
}

bool block::canHasKeyword ( const std::string& keyword )
{
    for ( int i=0;i<nLine();++i )
    {
        size_t fpos = line[i].find(keyword);
        if ( fpos != string::npos )
        {
            return true;
        }
    }
    return false;
}

template<class T1,class T2>
pair<T1,T2> getDataPair( const string& line )
{
    stringstream ss;
    ss << line;
    pair<T1,T2> result;
    ss >> result.first;
    ss >> result.second;
    return result;
}

template<class T>
T getData( const string& line )
{
    stringstream ss;
    ss << line;
    T result;
    ss >> result;
    return result;
}

block block::getBlock( const string& keyword )
{
    block myblock;
    myblock.name = keyword;
    bool found = false;
    //cout << "searching for   " << keyword << endl;
    for ( unsigned int l=0;l<line.size();++l )
    {
        // determine how many indents are on this line
        unsigned int thisIndent = 0;
        for ( unsigned int c=0;c<line[l].size();++c )
        {
            if ( line[l][c] == '\t' )
                thisIndent++;
            else
                break;
        }
        //cout << "there are " << thisIndent
        //  << " indents on this line" << endl;

        // search for keyword on this line
        size_t fpos = line[l].find( keyword );
        //cout << "found keyword at position " << fpos << endl;
        if ( fpos == thisIndent )
        {
        //  cout << "found keyword and adding lines" << endl;
            found = true;
            line.erase(line.begin()+l);

            // get lines until we return to the same indent
            for ( unsigned int ll=l;ll<line.size();++ll )
            {
                unsigned int nIndent = 0;
```

```
                for ( unsigned int c=0;c<line[ll].size();++c )
                {
                    if ( line[ll][c] == '\t' )
                        nIndent++;
                    else
                        break;
                }
                if ( nIndent > thisIndent )
                {
                    myblock.line.push_back(line[ll]);
                    line.erase(line.begin()+ll);
                    ll--;
                    continue;
                }
                else
                {
                    break;
                }
            }
            break;
        }

    }
    return myblock;
}


vector<gc> getGCData( block& detblock )
{
    vector<block> gcblock = detblock.getBlocks("gc");
    vector<gc> result( gcblock.size() );
    for ( unsigned int j=0;j<gcblock.size();++j )
    {
        result[j].nint = gcblock[j].getData<int>("nint");
        //result[j].nsigma = gcblock[j].getData<double>("nsigma");
        block nuisblock = gcblock[j].getBlock("nuisance");
        if ( ! nuisblock.empty() )
        {
            result[j].jobName
                = nuisblock.getData<std::string>("name");
            result[j].detName
                = nuisblock.getData<std::string>("det");
            result[j].specName
                = nuisblock.getData<std::string>("spec");
        }
    }
    return result;
}

vector<temp> getTempData( block& detblock )
{
    vector<block> tempblock = detblock.getBlocks("template");
    vector<temp> result( tempblock.size() );
    for ( unsigned int j=0;j<tempblock.size();++j )
    {
        block typeblock = tempblock[j].getBlock("types");
        for ( int k=0;k<typeblock.nLine();++k )
        {
            result[j].tempname.push_back(
                getData<std::string>(typeblock.line[k]) );
        }
        block nuisblock = tempblock[j].getBlock("nuisance");
        if ( ! nuisblock.empty() )
        {
            result[j].jobName
                = nuisblock.getData<std::string>("name");
            result[j].detName
                = nuisblock.getData<std::string>("det");
            result[j].specName
                = nuisblock.getData<std::string>("spec");
        }
    }
    return result;
}

vector<ew> getEWData( block& detblock )
```

```
{
    vector<block> ewblock = detblock.getBlocks("ew");
    vector<ew> result( ewblock.size() );
    for ( unsigned int j=0;j<ewblock.size();++j )
    {
        result[j].nint = ewblock[j].getData<int>("nint");
        //result[j].nsigma = ewblock[j].getData<double>("nsigma");
        if ( ewblock[j].canHasKeyword("nwindow") )
        {
            result[j].nwindow
                = ewblock[j].getData<int>("nwindow");
            result[j].spacing
                = ewblock[j].getData<string>("spacing");
        }
        else
        {
            int numWindow = ewblock[j].getData<int>("nbound")-1;
            block bblock = ewblock[j].getBlock("bounds");
            std::stringstream ss;
            ss << bblock.line[0];
            string dummy;
            ss >> dummy;
            result[j].windowBins.resize(numWindow+1);
            for ( int k=0;k<numWindow+1;++k )
            {
                ss >> result[j].windowBins[k];
            }
        }
        block nuisblock = ewblock[j].getBlock("nuisance");
        if ( ! nuisblock.empty() )
        {
            result[j].jobName
                = nuisblock.getData<std::string>("name");
            result[j].detName
                = nuisblock.getData<std::string>("det");
            result[j].specName
                = nuisblock.getData<std::string>("spec");
        }
    }
    return result;
}

pdet getDetectorData( block& detblock )
{
    pdet result;
    result.posx = detblock.getData<double>("posx");
    result.posy = detblock.getData<double>("posy");
    result.posz = detblock.getData<double>("posz");
    result.dimx = detblock.getData<double>("dimx");
    result.dimy = detblock.getData<double>("dimy");
    result.dimz = detblock.getData<double>("dimz");
    result.eff = detblock.getData<double>("eff");
    result.A = detblock.getData<double>("gebA");
    result.B = detblock.getData<double>("gebB");
    result.C = detblock.getData<double>("gebC");
    result.fanofac = detblock.getData<double>("fanofac");
    result.gcalarm = getGCData( detblock );
    result.ewalarm = getEWData( detblock );
    result.tempalarm = getTempData( detblock );
    return result;
}




void data::parse ( const std::string& filename )
{
    using namespace std;

    // get necessary paths
    //
    mypath.parse ( );

    my_filename = filename;
```

```
my_comment = '#';

// open input file
//
ifstream infile ( my_filename.c_str() );
if ( ! infile.good() )
{
    throw fatal_error (
        "error opening input file "+my_filename );
}
// get lines
block fileblock;
while ( ! infile.eof() )
{
    string line;
    getline(infile,line);
    fileblock.line.push_back(line);
}
// close inputfile
infile.close();

// remove lines with comments
for ( unsigned int i=0;i<fileblock.line.size();++i )
{
    if ( fileblock.line[i][0] == my_comment )
    {
        fileblock.line.erase(fileblock.line.begin()+i);
        i--;
        continue;
    }
}

// get physics block
block physblock = fileblock.getBlock("physics");
if ( physblock.empty() )
{
    throw fatal_error("physics block not found");
}
trackPhoton = w2b( physblock.getData<string>("photon") );
trackNeutron = w2b( physblock.getData<string>("neutron") );
macroTime = w2b( physblock.getData<string>("mactime") );
ergRedFact = physblock.getData<int>("ergfac");
refeps = physblock.getData<double>("refeps");
interval = physblock.getData<double>("interval");
trackFissGam = w2b( physblock.getData<string>("fissgamma") );
toggleBackground = w2b( physblock.getData<string>("background") );


// get source block
block srcblock = fileblock.getBlock("source");
if ( srcblock.empty() )
{
    throw fatal_error("source block not found");
}
// get snm block(s)
vector<block> snmblock = srcblock.getBlocks("snm");
//if ( snmblock.size() == 0 )
//{
    //throw fatal_error("no snm found");
//}
mysnm.resize(snmblock.size());
for ( unsigned int i=0;i<mysnm.size();++i )
{
    block isoblock = snmblock[i].getBlock("iso");
    for ( int j=0;j<isoblock.nLine();++j )
    {
        pair<string,double> isoPair
            = getDataPair<string,double>(isoblock.line[j]);
        mysnm[i].isoVector.push_back(isoPair.first);
        mysnm[i].fraction.push_back(isoPair.second);
    }
    double sum = 0.0;
    for ( unsigned int j=0;j<mysnm[i].fraction.size();++j )
    {
        sum += mysnm[i].fraction[j];
    }
```

```
    for ( unsigned int j=0;j<mysnm[i].fraction.size();++j )
    {
        mysnm[i].fraction[j] = mysnm[i].fraction[j]/sum;
    }
    mysnm[i].type = snmblock[i].getData<string>("type");
    snmblock[i].getData("mass",mysnm[i].mass);
    snmblock[i].getData("age",mysnm[i].age);
    block shieldblock = snmblock[i].getBlock("shield");
    vector<block> layerblock = shieldblock.getBlocks("layer");
    mysnm[i].myshield.resize(layerblock.size());
    for ( unsigned int j=0;j<layerblock.size();++j )
    {
        mysnm[i].myshield[j].type
            = layerblock[j].getData<std::string>("type");
        mysnm[i].myshield[j].thickness
            = layerblock[j].getData<double>("thick");
        mysnm[i].myshield[j].omegaStream
            = layerblock[j].getData<double>("stream");
    }
    snmblock[i].getData("posx",mysnm[i].posx);
    snmblock[i].getData("posy",mysnm[i].posy);
    snmblock[i].getData("posz",mysnm[i].posz);
}
// get NORM block
block normblock = srcblock.getBlock("norm");
if ( normblock.empty() )
{
    normfrac = 0.0;
}
else
{
    norm = normblock.getData<string>("type");
    normfrac = normblock.getData<double>("frac");
}
// get sources from file
vector<block> readblock = srcblock.getBlocks("read");
myreadsrc.resize(readblock.size());
for ( unsigned int i=0;i<myreadsrc.size();++i )
{
    myreadsrc[i].filename
        = readblock[i].getData<std::string>("filename");
    myreadsrc[i].posx = readblock[i].getData<double>("posx");
    myreadsrc[i].posy = readblock[i].getData<double>("posy");
    myreadsrc[i].posz = readblock[i].getData<double>("posz");
}

// get vehicle block
block vehblock = fileblock.getBlock("vehicle");
if ( vehblock.empty() )
{
    throw fatal_error("vehicle block not found");
}
if ( vehblock.canHasKeyword("none") )
{
    hasVehicle = false;
    mycargo.velocity = 8.05;
}
else if ( vehblock.canHasKeyword("truck") )
{
    hasVehicle = true;
    block truckblock = vehblock.getBlock("truck");
    block cargoblock = truckblock.getBlock("cargo");
    for ( int j=0;j<cargoblock.nLine();++j )
    {
        pair<string,double> typePair
            = getDataPair<string,double>(cargoblock.line[j]);
        mycargo.type.push_back( typePair.first );
        mycargo.typeFrac.push_back( typePair.second );
    }
    mycargo.typeFrac = normalize( mycargo.typeFrac );
    mycargo.velocity = truckblock.getData<double>("velocity");
}

// get background radiation block
block bgblock = fileblock.getBlock("background");
if ( bgblock.empty() )
```

```
{
    throw fatal_error("background_block_not_found");
}
block photonblock = bgblock.getBlock("photon");
block neutronblock = bgblock.getBlock("neutron");
photonblock.getData("usoil",usoil);
photonblock.getData("uconc",uconc);
photonblock.getData("thsoil",thsoil);
photonblock.getData("thconc",thconc);
photonblock.getData("ksoil",ksoil);
photonblock.getData("kconc",kconc);
neutronblock.getData("lat",latitude);
neutronblock.getData("long",longitude);
neutronblock.getData("elev",elevation);
neutronblock.getData("smod",solarMod);

// get detection block
block detectblock = fileblock.getBlock("detection");
if ( detectblock.empty() )
{
    throw fatal_error("detection_block_not_found");
}
// get detector block(s)
vector<block> pvtdetblock = detectblock.getBlocks("pvt");
vector<block> naidetblock = detectblock.getBlocks("nai");
vector<block> hpgedetblock = detectblock.getBlocks("nai");
vector<block> he3detblock = detectblock.getBlocks("he3");
vector<block> ssdetblock = detectblock.getBlocks("ss");

mypvtdet.resize( pvtdetblock.size() );
mynaidet.resize( naidetblock.size() );
myhpgedet.resize( hpgedetblock.size() );
myhedet.resize( he3detblock.size() );
myssdet.resize( ssdetblock.size() );
for ( unsigned int i=0;i<pvtdetblock.size();++i )
{
    mypvtdet[i] = getDetectorData( pvtdetblock[i] );
    mypvtdet[i].type = "pvt";
    mypdet.push_back(&mypvtdet[i]);
}
for ( unsigned int i=0;i<naidetblock.size();++i )
{
    mynaidet[i] = getDetectorData( naidetblock[i] );
    mynaidet[i].type = "nai";
    mypdet.push_back(&mynaidet[i]);
}
for ( unsigned int i=0;i<hpgedetblock.size();++i )
{
    myhpgedet[i] = getDetectorData( hpgedetblock[i] );
    myhpgedet[i].type = "hpge";
    mypdet.push_back(&myhpgedet[i]);
}
for ( unsigned int i=0;i<he3detblock.size();++i )
{
    myhedet[i].posx
        = he3detblock[i].getData<double>("posx");
    myhedet[i].posy
        = he3detblock[i].getData<double>("posy");
    myhedet[i].posz
        = he3detblock[i].getData<double>("posz");
    myhedet[i].height
        = he3detblock[i].getData<double>("height");
    myhedet[i].modrad
        = he3detblock[i].getData<double>("modrad");
    myhedet[i].refrad
        = he3detblock[i].getData<double>("refrad");
    myhedet[i].eff
        = he3detblock[i].getData<double>("eff");
    myhedet[i].fanofac
        = he3detblock[i].getData<double>("fanofac");
    myhedet[i].gcalarm
        = getGCData( he3detblock[i] );
    myndet.push_back(&myhedet[i]);
}
for ( unsigned int i=0;i<ssdetblock.size();++i )
{
```

```
            myssdet[i].posx
                = ssdetblock[i].getData<double>("posx");
            myssdet[i].posy
                = ssdetblock[i].getData<double>("posy");
            myssdet[i].posz
                = ssdetblock[i].getData<double>("posz");
            myssdet[i].modt
                = ssdetblock[i].getData<double>("modt");
            myssdet[i].area
                = ssdetblock[i].getData<double>("area");
            myssdet[i].eff
                = ssdetblock[i].getData<double>("eff");
            myssdet[i].fanofac
                = ssdetblock[i].getData<double>("fanofac");
            myssdet[i].gcalarm
                = getGCData( ssdetblock[i] );
            myndet.push_back(&myssdet[i]);
        }


        block saveblock = fileblock.getBlock("save");
        if ( ! saveblock.empty() )
        {
            mysave.doSave = true;
            mysave.jobName
                = saveblock.getData<std::string>("name");
            mysave.saveSuppBackground
                = saveblock.canHasKeyword("suppbackground");
            mysave.saveSignal
                = saveblock.canHasKeyword("signal");
            mysave.saveBackground
                = saveblock.canHasKeyword("background");
        }



        std::ifstream truckdimin ( mypath.truckdim.c_str() );
        Find ( truckdimin,"vehicledim",true );
        Find ( truckdimin,"x0",false );
        truckdimin >> mycargo.vehiclex0;
        Find ( truckdimin,"x1",false );
        truckdimin >> mycargo.vehiclex1;
        Find ( truckdimin,"y0",false );
        truckdimin >> mycargo.vehicley0;
        Find ( truckdimin,"y1",false );
        truckdimin >> mycargo.vehicley1;
        Find ( truckdimin,"z0",false );
        truckdimin >> mycargo.vehiclez0;
        Find ( truckdimin,"z1",false );
        truckdimin >> mycargo.vehiclez1;
        Find ( truckdimin,"cargodim",true );
        Find ( truckdimin,"x0",false );
        truckdimin >> mycargo.cargox0;
        Find ( truckdimin,"x1",false );
        truckdimin >> mycargo.cargox1;
        Find ( truckdimin,"y0",false );
        truckdimin >> mycargo.cargoy0;
        Find ( truckdimin,"y1",false );
        truckdimin >> mycargo.cargoy1;
        Find ( truckdimin,"z0",false );
        truckdimin >> mycargo.cargoz0;
        Find ( truckdimin,"z1",false );
        truckdimin >> mycargo.cargoz1;
        truckdimin.close();
    }

} // end namespace Input
```

Listing B.8: data.hpp

```
#ifndef _data_hpp_included_
#define _data_hpp_included_

#include <vector>
#include <string>
#include <memory>
#include "dspectrum.hpp"
#include "paths.hpp"

namespace input
{


struct algorithmbase
{
    int nint;
    std::string jobName;
    std::string detName;
    std::string specName;
};

struct gc : public algorithmbase
{
    double fap;
};

struct ew : public algorithmbase
{
    double fap;
    int nwindow;
    std::vector< double > windowBins;
    std::string spacing;
};

struct temp: public algorithmbase
{
    std::vector<std::string> tempname;
};




struct shield
{
    // type of shielding
    std::string type;
    // shielding thickness in cm
    double thickness;
    // fraction of streaming pathways out of shield
    double omegaStream;
};

struct snm
{
    // isotopic vector of strings (e.g. "Pu239","U235")
    std::vector< std::string > isoVector;
    // fraction of each isotope
    std::vector< double > fraction;
    // age of mixture
    double age;
    // original, user-specified string for radsrc
    std::string radsrcinput;
    // density is in grams/cc
    //double density; // NOT ANYMORE!!! NOT APPLICABLE
    // mass is in grams
    double mass;
    // (x,y,z) position of sphere in cargo
    double posx;
    double posy;
    double posz;
    std::vector<shield> myshield;
    std::string type;
};
```

```
struct detbase
{
    std::string type;
    // location (x,y,z)
    double posx;
    double posy;
    double posz;
    // fano factor
    double fanofac;
};

struct hedet : public detbase
{
    // dimension
    double height;
    double modrad;
    double refrad;
    // collection efficiency
    double eff;
    // alarms
    std::vector<gc> gcalarm;
    hedet( ) { type="he3"; };
};

struct ssdet : public detbase
{
    // dimension
    double modt;
    double area;
    // collection efficiency
    double eff;

    // alarms
    std::vector<gc> gcalarm;
    ssdet( ) { type="ss"; };
};

struct pdet : public detbase
{
    // dimension (width,height,depth)
    double dimx;
    double dimy;
    double dimz;
    double eff;
    // gaussian energy broadening parameters
    double A;
    double B;
    double C;
    // alarms
    std::vector<gc> gcalarm;
    std::vector<ew> ewalarm;
    std::vector<temp> tempalarm;
};

struct cargo
{
    // options are void,lowz,midz,highz materials
    std::vector<std::string> type;
    std::vector<double> typeFrac;
    // truck velocity through RPM in km/h
    double velocity;
    // vehicle dimensions in cm
    // length (y)
    double vehicley0;
    double vehicley1;
    // width (x)
    double vehiclex0;
    double vehiclex1;
    // height (z)
    double vehiclez0;
    double vehiclez1;
    // cargo dimensions in cm
    // length (y)
    double cargoy0;
    double cargoy1;
    // width (x)
```

```
    double cargox0;
    double cargox1;
    // height (z)
    double cargoz0;
    double cargoz1;
    // fill point -- where the default source location is
    //std::vector< double > fillpt;
    // fraction of streaming pathways out of cargo
};


struct save
{
    bool doSave;
    std::string jobName;

    bool saveSuppBackground;
    bool saveSignal;
    bool saveBackground;


    save( ) { doSave = false; };
};

struct readsrc
{
    double posx;
    double posy;
    double posz;
    std::string filename;
};




struct block
{
    bool empty( ) { return nLine()==0; };
    int nLine( ) { return static_cast<int>(line.size()); };
    std::string name;
    std::vector<std::string> line;

    template< class T >
    T getData ( const std::string& keyword );

    bool canHasKeyword ( const std::string& keyword );

    template< class T >
    bool getData( const std::string& keyword,T& val );

    block getBlock( const std::string& keyword );

    std::vector<block> getBlocks( const std::string& keyword );
};

class data
{
    public:
    // det information
    //
    std::vector< pdet > mypvtdet;
    std::vector< pdet > mynaidet;
    std::vector< pdet > myhpgedet;
    std::vector< hedet > myhedet;
    std::vector< ssdet > myssdet;

    std::vector<pdet*> mypdet;
    std::vector<detbase*> myndet;

    // source information
    //
    std::vector< snm > mysnm;

    // file source information
    //
```

```
    std::vector< readsrc > myreadsrc;

    // cargo information
    //
    cargo mycargo;
    bool hasVehicle;

    // natural background information
    //
    double usoil;
    double uconc;
    double thsoil;
    double thconc;
    double ksoil;
    double kconc;
    double latitude;
    double longitude;
    double elevation;
    double solarMod;

    //void parse ( const std::string&,bool );
    void parse ( const std::string& );

    // physics
    //
    bool trackFissGam; // do prompt fission gammas
    bool trackPhoton; // do photon transport
    bool trackNeutron; // do neutron transport
    bool macroTime; // toggle macro time dependence
    bool microTime; // toggle micro time dependence (neutron only)
    int ergRedFact; // energy reduction factor (photon only)
    double refeps; // convergence for reflection
    double interval;// time step interval in seconds
    bool toggleBackground; // do background calculations

    // NORM type
    std::string norm;
    // fraction of the cargo which is norm
    double normfrac;


    paths mypath;

    save mysave;


    private:

    std::string my_filename;
    std::vector<std::string> my_line;
    std::vector<block> my_block;
    char my_comment;

};

} //end namespace Input

#endif
```

Listing B.9: datapoint.cpp

```
#include "datapoint.hpp"
#include <limits>
#include <math.h>

double datapoint::get ( )
{
    return my_value;
}

double datapoint::get ( ) const
{
    return my_value;
}

void datapoint::set ( double a )
{
    my_value = a;
}

double datapoint::getErr ( )
{
    return my_error;
}

double datapoint::getErr ( ) const
{
    return my_error;
}

double datapoint::getVar ( )
{
    return my_error*my_error;
}

double datapoint::getVar ( ) const
{
    return my_error*my_error;
}

double datapoint::getAbsErr ( )
{
    return my_error*my_value;
}

double datapoint::getAbsErr ( ) const
{
    return my_error*my_value;
}

double datapoint::getAbsVar ( )
{
    const double result = my_error*my_value;
    return ( result*result );
}

double datapoint::getAbsVar ( ) const
{
    const double result = my_error*my_value;
    return ( result*result );
}

void datapoint::setErr ( double a )
{
    my_error = a;
    return;
}

void datapoint::setVar ( double a )
{
    my_error = sqrt ( a );
    return;
}

void datapoint::setAbsErr ( double a )
{
```

```
    if ( my_value < std::numeric_limits<double>::min() )
    {
        my_error = 0;
    }
    else
    {
        my_error = a / my_value;
    }
    return;
}

void datapoint::setAbsVar ( double a )
{
    if ( my_value < std::numeric_limits<double>::min() )
    {
        my_error = 0;
    }
    else
    {
        my_error = sqrt ( a ) / my_value;
    }
    return;
}

bool datapoint::operator< ( const datapoint& a ) const
{
    return (my_value < a.get());
}

bool datapoint::operator> ( const datapoint& a )
{
    return (my_value > a.get());
}

bool datapoint::operator== ( const datapoint& a )
{
    return ( my_value == a.get() && this->getErr() == a.getErr() );
}

void datapoint::operator= ( const datapoint& a )
{
    set ( a.get() );
    setErr ( a.getErr() );
    return;
}

void datapoint::operator+= ( const datapoint& a )
{
    set ( my_value+a.get() );
    setAbsVar ( this->getAbsVar()+a.getAbsVar() );
    return;
}

void datapoint::operator= ( double a )
{
    set ( a );
    return;
}

void datapoint::operator *= ( double a )
{
    set ( my_value*a );
    return;
}

void datapoint::zero( )
{
    set( 0.0 );
    setErr( 0.0 );
    return;
}

datapoint::datapoint( )
{
    set( 0.0 );
    setErr( 0.0 );
```

```
}

datapoint::datapoint( double val,double err )
{
    set( val );
    setErr( err );
}


datapoint logInterpolate ( const datapoint& a,
    const datapoint& b, double td )
{
    datapoint result;

    // if x or y is zero, set to zero
    // to avoid +/- infinity when taking log (y/x)
    if ( fabs(a.get()) < std::numeric_limits<double>::min() ||
            fabs(b.get()) < std::numeric_limits<double>::min() )
    {
        return linearInterpolate(a,b,td);
    }
    else
    {
        result = myexp( (1.0-td)*mylog(a) + td*mylog(b) );
    }

    // check for nan or inf, make zero
    if ( isnan(result.get()) || isinf(result.get()) )
    {
        result.zero( );
    }

    return result;
}

datapoint logInterpolate2 ( double x,double x1,
    const datapoint& y1, double slope )
{
    datapoint result;

    // if x or y is zero, set to zero
    // to avoid +/- infinity when taking log (y/x)
    if ( fabs(y1.get()) < std::numeric_limits<double>::min() )
    {
        return datapoint(0.0,0.0);
    }
    else
    {
        result = myexp( mylog(y1) + (x-x1)*slope );
    }

    // check for nan or inf, make zero
    if ( isnan(result.get()) || isinf(result.get()) )
    {
        result.zero( );
    }

    return result;
}



datapoint logInterpolate ( double val_a, const datapoint& a,
                           double val_b, const datapoint& b,
                           double val )
{
    datapoint result;

    const double td = ( val-val_a ) / ( val_b-val_a );

    // if x or y is zero, default to linear interpolation
    // to avoid +/- infinity when taking log (y/x)
    if ( fabs(a.get()) < std::numeric_limits<double>::min() ||
            fabs(b.get()) < std::numeric_limits<double>::min() )
    {
        result = linearInterpolate ( val_a,a,val_b,b,val );
```

```
        if ( result.get() < 0.0 )
        {
            result.zero( );
        }
        return result;
    }
    else
    {
        //result = a*myexp ( td*mylog ( b/a ) );
        result = myexp( mylog(a) + td*(mylog(b)-mylog(a)) );
        //result = a*myexp ( td*mylog ( b/a ) );
    }

    // check for nan or inf, make zero
    if ( isnan(result.get()) || isinf(result.get()) )
    {
        result.zero( );
    }

    return result;
}

datapoint linearInterpolate ( double val_a , const datapoint& a,
                              double val_b , const datapoint& b,
                              double val )

{
    datapoint result;

    const double td = ( val-val_a ) / ( val_b-val_a );

    result = a + ( b-a ) *td;

    return result;
}

datapoint linearInterpolate ( const datapoint& a,
    const datapoint& b,double td )

{
    return ( a + ( b-a ) *td );
}

double logInterpolate ( double val_a , double a,
                        double val_b , double b,
                        double val )

{
    double result;

    const double td = ( val-val_a ) / ( val_b-val_a );

    const double x = a;
    const double y = b;

    // if x or y is zero, default to linear interpolation
    // to avoid +/- infinity when taking log (y/x)
    if ( fabs(x) < std::numeric_limits<double>::min() ||
            fabs(y) < std::numeric_limits<double>::min() )
    {
        result = linearInterpolate ( val_a ,a,val_b ,b,val );
    }
    else
    {
        result = x*exp ( td*log ( y/x ) );
    }

    return result;
}

double linearInterpolate ( double val_a , double a,
                           double val_b , double b,
                           double val )

{
    double result;
```

```
    const double td = ( val-val_a ) / ( val_b-val_a );

    const double x = a;
    const double y = b;

    result = x + ( y-x ) *td;

    return result;
}


datapoint mylog ( const datapoint& a )
{
    datapoint result;

    // take log of value
    result.set ( log ( a.get() ) );

    // propagate error
    // sig_f = sig_a / a
    result.setAbsErr ( a.getErr() );

    return result;
}

datapoint myexp ( const datapoint& a )
{
    datapoint result;

    // take log of value
    result.set ( exp ( a.get() ) );

    // propagate error
    // sig_f = f * sig_a
    // or, sig_f / f = sig_a
    result.setErr ( a.getAbsErr() );

    return result;
}


datapoint operator+ ( const datapoint& a,const datapoint& b )
{
    datapoint result;

    // add values
    result.set ( a.get() + b.get() );

    // propagate error
    result.setAbsVar ( a.getAbsVar() + b.getAbsVar() );

    return result;
}

datapoint operator+ ( const datapoint& a,double b )
{
    datapoint result;

    // add values
    result.set ( a.get() + b );

    result.setErr ( a.getErr() );

    return result;
}

datapoint operator- ( const datapoint& a,const datapoint& b )
{
    datapoint result;

    // add values
    result.set ( a.get() - b.get() );

    // propagate error
    result.setAbsVar ( a.getAbsVar() + b.getAbsVar() );
```

```
    return result;
}

datapoint operator* ( const datapoint& a,const datapoint& b )
{
    datapoint result;

    // multiply values
    result.set ( a.get() * b.get() );

    // propagate error
    result.setVar ( a.getVar() + b.getVar() );

    return result;
}

datapoint operator/ ( const datapoint& a,const datapoint& b )
{
    datapoint result;

    // multiply values
    result.set ( a.get() / b.get() );

    // propagate error
    result.setVar ( a.getVar() + b.getVar() );

    return result;
}

datapoint operator* ( const datapoint& a,double b )
{
    datapoint result;

    // multiply values
    result.set ( a.get() * b );

    // propagate error -- doesn't change relative error
    result.setErr ( a.getErr() );

    return result;
}

datapoint operator* ( double b,const datapoint& a )
{
    return a*b;
}

datapoint operator/ ( const datapoint& a,double b )
{
    datapoint result;

    // multiply values
    result.set ( a.get() / b );

    // propagate error -- doesn't change relative error
    result.setErr ( a.getErr() );

    return result;
}

bool operator< ( const datapoint& a,double b )
{
    return (a.get()<b);
}

bool operator> ( const datapoint& a,double b )
{
    return (a.get()>b);
}

//datapoint operator/ ( double b,const datapoint&a )
//{
//    return a/b;
//}
```

```
//bool operator> ( const datapoint& a,const datapoint& b )
//{
//   return ( a.get() > b.get() );
//}

std::ostream& operator << ( std::ostream& output,
    const datapoint& a )
{
    output << a.get() << " " << a.getErr();
    return output;
}
```

Listing B.10: datapoint.hpp

```
#ifndef _datapoint_hpp_included_
#define _datapoint_hpp_included_

#include <math.h>
#include "errh.hpp"



class datapoint
{
    public:

    double get();
    double get() const;

    void set(double);

    double getErr();
    double getErr() const;

    double getVar();
    double getVar() const;

    double getAbsErr();
    double getAbsErr() const;

    double getAbsVar();
    double getAbsVar() const;

    void setErr(double);
    void setVar(double);
    void setAbsErr(double);
    void setAbsVar(double);

    void zero( );


    bool operator< ( const datapoint& ) const;
    bool operator> ( const datapoint& );
    bool operator== ( const datapoint& );

    void operator = ( const datapoint& );
    void operator += ( const datapoint& );
    void operator = ( double );
    void operator *= ( double );


    datapoint( );
    datapoint( double,double );


    protected:

    double my_value;

    // RELATIVE error
    double my_error;

};

datapoint logInterpolate ( double,const datapoint&,
    double,const datapoint&,double );

datapoint logInterpolate ( const datapoint&,
    const datapoint&,double );

datapoint logInterpolate2 ( double x,double x1,
    const datapoint& y1, double slope );

datapoint linearInterpolate ( double,const datapoint&,
    double,const datapoint&,double );

datapoint linearInterpolate ( const datapoint&,
    const datapoint&,double );
```

```
double logInterpolate ( double,double,
    double,double,double );

double linearInterpolate ( double,double,
    double,double,double );

datapoint mylog ( const datapoint& );

datapoint myexp ( const datapoint& );

datapoint operator+ ( const datapoint&,const datapoint& );

datapoint operator+ ( const datapoint&,double );

datapoint operator- ( const datapoint&,const datapoint& );

datapoint operator* ( const datapoint&,const datapoint& );

datapoint operator/ ( const datapoint&,const datapoint& );

datapoint operator* ( const datapoint&,double );

datapoint operator* ( double,const datapoint& );

datapoint operator/ ( const datapoint&,double );

bool operator< ( const datapoint&,double );

bool operator> ( const datapoint&,double );

std::ostream& operator << ( std::ostream&, const datapoint& );

//bool operator> ( const datapoint&,const datapoint& ) const;

#endif
```

Listing B.11: detector.cpp

```
#include "detector.hpp"
#include <fstream>
#include <iostream>
#include <math.h>
#include "fileio.hpp"
#include "extras.hpp"
#include "interpolation.hpp"

int detector::numInErgs ( )
{
    return my_R.numergin();
}

int detector::numInErgs ( ) const
{
    return my_R.numergin();
}

int detector::numOutErgs ( )
{
    return my_R.numergout();
}

int detector::numOutErgs ( ) const
{
    return my_R.numergout();
}

void detector::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector dimensions
    std::vector<double> tempDim = readbin( my_datapath+"detdim.dat" );
    my_nDim = static_cast<int>( tempDim.size()/3 );
    my_detDim.resize(my_nDim);
    for ( int i=0;i<my_nDim;++i )
    {
        my_detDim[i].resize(3);
        my_detDim[i][0] = tempDim[3*i];
        my_detDim[i][1] = tempDim[3*i+1];
        my_detDim[i][2] = tempDim[3*i+2];
    }

    return;
}

detector::detector ( const std::string& path,
    double detXPos,double detYPos,double detZPos,
    double detXDim,double detYDim,double detZDim,
    double eff,double A,double B,double C )
{
    initialize( path );
    my_detXPos0 = detXPos;
    my_detYPos0 = detYPos;
    my_detZPos0 = detZPos;
    my_detXDim0 = detXDim;
    my_detYDim0 = detYDim;
    my_detZDim0 = detZDim;
    my_eff = eff;
    my_A = A;
    my_B = B;
    my_C = C;

}

void detector::initialize ( const std::string& path )
{
    my_datapath = path + sep();
    readDataFile();
}

std::string detector::getTallyEnergyPath( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
```

```
}

std::vector<std::string> detector::getTallyPath( int ergIdx )
{
    std::vector<std::string> result(8);
    result[0] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx000) + sep();
    result[1] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx100) + sep();
    result[2] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx010) + sep();
    result[3] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx001) + sep();
    result[4] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx110) + sep();
    result[5] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx011) + sep();
    result[6] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx101) + sep();
    result[7] = my_datapath + "erg" + str(ergIdx)
        + sep() + "dim" + str(my_idx111) + sep();
    return result;
}



std::vector<datapoint> detector::interpolateTallies(
    double newpeak, const std::vector< double >& newerg,
    int ergIdx1, int ergIdx3 )
{
    // get indices
    const int newSrcIdx1 = ergIdx1;
    const int newSrcIdx3 = ergIdx3;

    // parse tallies
    if ( newSrcIdx1 != srcIdx1 )
    {
        tallyPtr tal1_000 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_100 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_010 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_001 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_110 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_011 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_101 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_111 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_00 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_10 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_01 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_11 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_0 = tallyPtr( new tally(my_redFact) );
        tallyPtr tal1_1 = tallyPtr( new tally(my_redFact) );

        srcIdx1 = newSrcIdx1;

        std::string talErgPath = getTallyEnergyPath(ergIdx1);
        std::string talPath;

        talPath = getTallyPath(ergIdx1)[0];
        tal1_000->parse( talErgPath, talPath );
        tal1_000->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1)[1];
        tal1_100->parse( talErgPath, talPath );
        tal1_100->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1)[2];
        tal1_010->parse( talErgPath, talPath );
        tal1_010->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1)[3];
        tal1_001->parse( talErgPath, talPath );
        tal1_001->setSourceEnergy( my_srcErg[ergIdx1] );

        talPath = getTallyPath(ergIdx1)[4];
        tal1_110->parse( talErgPath, talPath );
        tal1_110->setSourceEnergy( my_srcErg[ergIdx1] );
```

```
    talPath = getTallyPath(ergIdx1)[5];
    tal1_011->parse( talErgPath,talPath );
    tal1_011->setSourceEnergy( my_srcErg[ergIdx1] );

    talPath = getTallyPath(ergIdx1)[6];
    tal1_101->parse( talErgPath,talPath );
    tal1_101->setSourceEnergy( my_srcErg[ergIdx1] );

    talPath = getTallyPath(ergIdx1)[7];
    tal1_111->parse( talErgPath,talPath );
    tal1_111->setSourceEnergy( my_srcErg[ergIdx1] );

    // interpolate between dimensions
    // interpolate out z dimension
    tal1_00->interpolate( tal1_000,tal1_001,my_z0,
        my_z1,my_detZDim,"lin" );
    //tal1_00->checkData();
    tal1_10->interpolate( tal1_100,tal1_101,my_z0,
        my_z1,my_detZDim,"lin" );
    //tal1_10->checkData();
    tal1_01->interpolate( tal1_010,tal1_011,my_z0,
        my_z1,my_detZDim,"lin" );
    //tal1_01->checkData();
    tal1_11->interpolate( tal1_110,tal1_111,my_z0,
        my_z1,my_detZDim,"lin" );
    //tal1_11->checkData();

    // interpolate out y dimension
    tal1_0->interpolate( tal1_00,tal1_01,my_y0,
        my_y1,my_detYDim,"lin" );
    //tal1_0->checkData();
    tal1_1->interpolate( tal1_10,tal1_11,my_y0,
        my_y1,my_detYDim,"lin" );
    //tal1_1->checkData();

    // interpolate out x dimension
    tal1->interpolate ( tal1_0,tal1_1,my_x0,
        my_x1,my_detXDim,"lin" );

    //tal1 = tal1_111;
    //tal1->checkData();
}


if ( newSrcIdx3 != srcIdx3 )
{
    tallyPtr tal3_000 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_100 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_010 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_001 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_110 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_011 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_101 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_111 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_00 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_10 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_01 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_11 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_0 = tallyPtr( new tally(my_redFact) );
    tallyPtr tal3_1 = tallyPtr( new tally(my_redFact) );

    srcIdx3 = newSrcIdx3;

    std::string talErgPath = getTallyEnergyPath(ergIdx3);
    std::string talPath;

    talPath = getTallyPath(ergIdx3)[0];
    tal3_000->parse( talErgPath,talPath );
    tal3_000->setSourceEnergy( my_srcErg[ergIdx3] );

    talPath = getTallyPath(ergIdx3)[1];
    tal3_100->parse( talErgPath,talPath );
    tal3_100->setSourceEnergy( my_srcErg[ergIdx3] );
```

```
        talPath = getTallyPath(ergIdx3)[2];
        tal3_010->parse( talErgPath,talPath );
        tal3_010->setSourceEnergy( my_srcErg[ergIdx3] );

        talPath = getTallyPath(ergIdx3)[3];
        tal3_001->parse( talErgPath,talPath );
        tal3_001->setSourceEnergy( my_srcErg[ergIdx3] );

        talPath = getTallyPath(ergIdx3)[4];
        tal3_110->parse( talErgPath,talPath );
        tal3_110->setSourceEnergy( my_srcErg[ergIdx3] );

        talPath = getTallyPath(ergIdx3)[5];
        tal3_011->parse( talErgPath,talPath );
        tal3_011->setSourceEnergy( my_srcErg[ergIdx3] );

        talPath = getTallyPath(ergIdx3)[6];
        tal3_101->parse( talErgPath,talPath );
        tal3_101->setSourceEnergy( my_srcErg[ergIdx3] );

        talPath = getTallyPath(ergIdx3)[7];
        tal3_111->parse( talErgPath,talPath );
        tal3_111->setSourceEnergy( my_srcErg[ergIdx3] );

        // interpolate between dimensions
        // interpolate out z dimension
        tal3_00->interpolate( tal3_000,tal3_001,
            my_z0,my_z1,my_detZDim,"lin" );
        tal3_10->interpolate( tal3_100,tal3_101,
            my_z0,my_z1,my_detZDim,"lin" );
        tal3_01->interpolate( tal3_010,tal3_011,
            my_z0,my_z1,my_detZDim,"lin" );
        tal3_11->interpolate( tal3_110,tal3_111,
            my_z0,my_z1,my_detZDim,"lin" );
        // interpolate out y dimension
        tal3_0->interpolate( tal3_00,tal3_01,
            my_y0,my_y1,my_detYDim,"lin" );
        tal3_1->interpolate( tal3_10,tal3_11,
            my_y0,my_y1,my_detYDim,"lin" );
        // interpolate out x dimension
        tal3->interpolate ( tal3_0,tal3_1,
            my_x0,my_x1,my_detXDim,"lin" );

        //tal3 = tal3_111;
    }

    // interpolate by source energy
    std::vector<double> peak;
    detinterpolator K;
    K.setSourceEnergies(
        my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);
    std::vector<datapoint> interpResult
        = K.interpolate( newerg,tal1,tal3,peak );

    return interpResult;
}


void detector::buildResponse ( double srcXPos,
    double srcYPos,double srcZPos,double detYPos,
    double maxErg,int redFact )
{
    //std::cout << "building detector response" << std::endl;

    double detXPos = my_detXPos0;
    double detZPos = my_detZPos0;

    // set all indices to -1
    srcIdx1 = -1;
    srcIdx3 = -1;

    my_redFact = redFact;
    // store dimensions/position of detector
    my_xPos = detXPos;
    my_yPos = detYPos;
    my_zPos = detZPos;
```

```
// hard-code in detector planes,
// should read this in from file eventually
const double tside = 129.54;
const double ttop = 259.08;
const double soff = 68.56;
double my_detXPlane = tside+soff; // side of truck + standoff
double my_detZPlane = ttop+soff; // side of truck + standoff

// find out if this detector is a "side" or "top" detector
if ( my_zPos < my_detZPlane && my_xPos > tside )
{
    my_detPosType = side;
}
else if ( my_zPos > ttop && my_xPos < my_detXPlane )
{
    my_detPosType = top;
}
else
{
    throw fatal_error("invalid detector position");
}

// detector face's normals
std::vector<double> nx(3,0.0);
std::vector<double> ny(3,0.0);
std::vector<double> nz(3,0.0);


// detector dimension min/maxes
// are different because detector
// is always specified by front center face
double xMin,xMax,yMin,yMax,zMin,zMax;
if ( my_detPosType == side )
{
    xMin = detXPos;
    xMax = detXPos + my_detXDim0;
    yMin = detYPos - my_detYDim0/2.0;
    yMax = detYPos + my_detYDim0/2.0;
    zMin = detZPos - my_detZDim0/2.0;
    zMax = detZPos + my_detZDim0/2.0;
}
else /* if ( my_detPosType == top ) */
{
    xMin = detXPos - my_detXDim0/2.0;
    xMax = detXPos + my_detXDim0/2.0;
    yMin = detYPos - my_detYDim0/2.0;
    yMax = detYPos + my_detYDim0/2.0;
    zMin = detZPos;
    zMax = detZPos + my_detZDim0;
}

// find detector side with largest solid angle, and artificially
// make that side thicker based on chord length through detector
// volume
double omegaX = 0.0;
double omegaY = 0.0;
double omegaZ = 0.0;

if ( srcXPos < detXPos - my_detXDim0/2.0 )
{
    nx[0] = -1.0;
    omegaX = solidAngle( srcXPos,srcYPos,srcZPos,
        xMin,yMax,zMax,
        xMin,yMin,zMin,
        xMin,yMax,zMin )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMin,yMax,zMax,
        xMin,yMin,zMin,
        xMin,yMin,zMax );
}
else if ( srcXPos > detXPos + my_detXDim0/2.0 )
{
    nx[0] = 1.0;
    omegaX = solidAngle( srcXPos,srcYPos,srcZPos,
```

```
        xMax,yMax,zMax,
        xMax,yMin,zMin,
        xMax,yMax,zMin )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMax,
        xMax,yMin,zMin,
        xMax,yMin,zMax );
}
else
{
    nx[0] = 1.0;
    omegaX = 0.0;
}

if ( srcYPos < detYPos - my_detYDim0/2.0 )
{
    ny[1] = -1.0;
    omegaY = solidAngle( srcXPos,srcYPos,srcZPos,
        xMax,yMin,zMax,
        xMin,yMin,zMin,
        xMax,yMin,zMin )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMax,yMin,zMax,
        xMin,yMin,zMin,
        xMin,yMin,zMax );
}
else if ( srcYPos > detYPos + my_detYDim0/2.0 )
{
    ny[1] = 1.0;
    omegaY = solidAngle( srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMax,
        xMin,yMax,zMin,
        xMax,yMax,zMin )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMax,
        xMin,yMax,zMin,
        xMin,yMax,zMax );
}
else
{
    ny[1] = 1.0;
    omegaY = 0.0;
}

if ( srcZPos < detZPos - my_detZDim0/2.0 )
{
    nz[2] = -1.0;
    omegaZ = solidAngle( srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMin,
        xMin,yMin,zMin,
        xMax,yMin,zMin )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMin,
        xMin,yMin,zMin,
        xMin,yMax,zMin );
}
else if ( srcZPos > detZPos + my_detZDim0/2.0 )
{
    nz[2] = 1.0;
    omegaZ = solidAngle( srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMax,
        xMin,yMin,zMax,
        xMax,yMin,zMax )
        +
        solidAngle(srcXPos,srcYPos,srcZPos,
        xMax,yMax,zMax,
        xMin,yMin,zMax,
        xMin,yMax,zMax );
}
else
{
    nz[2] = 1.0;
```

```
    omegaZ = 0.0;
}

// calculate angle between
// source-detector vector and cartesian axes
std::vector<double> d2s(3);
d2s[0] = srcXPos - detXPos;
d2s[1] = srcYPos - detYPos;
d2s[2] = srcZPos - detZPos;

//std::cout << "omegaX = " << omegaX << std::endl;
//std::cout << "omegaY = " << omegaY << std::endl;
//std::cout << "omegaZ = " << omegaZ << std::endl;
// calculate equivalent area of
// detector face based on sum of solid angles
my_equivArea = (omegaX+omegaY+omegaZ) * pow(mag(d2s),2.0);
//my_equivArea = detYDim*detZDim;

//std::cout << "my_equivArea = "
//  << my_equivArea << std::endl;

// the detector data is such that the +Y
//  direction is the beam direction
// so that will be the primary dimension,
//  or the dimension with the largest
// solid angle. the other two dimensions
//  order doesn't really matter

if ( omegaX > omegaY && omegaX > omegaZ )
{
    // angle formula and projection of a vector onto a plane with normal n
    std::vector<double> projy = projVecOnPlane(d2s,nz);
    // in radians
    double yAngle = angle(ny,projVecOnPlane(d2s,nz));
    // in radians
    double zAngle = angle(nz,projVecOnPlane(d2s,ny));
    // for x-angle, use total angle not
    // projection (this does not conserve mass of detector)
    double xAngle = angle(nx,d2s); // in radians
    // need to find "equivalent dimensions"
    // increase thickness of detector
    my_detYDim = my_detXDim0 / cos(xAngle);
    // reduce apparent width
    my_detXDim = my_detYDim0 * sin(yAngle);
    // reduce apparent height
    my_detZDim = my_detZDim0 * sin(zAngle);
}
else if ( omegaY > omegaX && omegaY > omegaZ )
{
    double xAngle = angle(nx,projVecOnPlane(d2s,nz));
    double zAngle = angle(nz,projVecOnPlane(d2s,nx));
    double yAngle = angle(ny,d2s);
    my_detYDim = my_detYDim0 / cos(yAngle);
    my_detXDim = my_detXDim0 * sin(xAngle);
    my_detZDim = my_detZDim0 * sin(zAngle);
}
else /* if ( omegaZ > omegaY && omegaZ > omegaX ) */
{
    double xAngle = angle(nx,projVecOnPlane(d2s,ny));
    double yAngle = angle(ny,projVecOnPlane(d2s,nx));
    double zAngle = angle(nz,d2s);
    my_detYDim = my_detZDim0 / cos(zAngle);
    my_detXDim = my_detXDim0 * sin(xAngle);
    my_detZDim = my_detYDim0 * sin(yAngle);
}

// need to interpolate between all three
// dimensions so need six data points
// find six closest dimensions
my_x0 = 200.0;
my_x1 = 200.0;
my_y0 = 200.0;
my_y1 = 200.0;
my_z0 = 200.0;
my_z1 = 200.0;
for ( unsigned int i=1;i<my_detDim.size();++i )
```

```
{
    if ( my_detDim[i][0] >= my_detXDim
        && my_detDim[i-1][0] < my_detXDim )
    {
        my_x0 = my_detDim[i-1][0];
        my_x1 = my_detDim[i][0];
    }
    if ( my_detDim[i][1] >= my_detYDim
        && my_detDim[i-1][1] < my_detYDim )
    {
        my_y0 = my_detDim[i-1][1];
        my_y1 = my_detDim[i][1];
    }
    if ( my_detDim[i][2] >= my_detZDim
        && my_detDim[i-1][2] < my_detZDim )
    {
        my_z0 = my_detDim[i-1][2];
        my_z1 = my_detDim[i][2];
    }
}


const double eps = 1e-5;
for ( unsigned int i=0;i<my_detDim.size();++i )
{
    if ( eq(my_detDim[i][0],my_x0,eps)
        && eq(my_detDim[i][1],my_y0,eps)
        && eq(my_detDim[i][2],my_z0,eps) )
    {
        my_idx000 = i;
    }
    if ( eq(my_detDim[i][0],my_x1,eps)
        && eq(my_detDim[i][1],my_y0,eps)
        && eq(my_detDim[i][2],my_z0,eps) )
    {
        my_idx100 = i;
    }
    if ( eq(my_detDim[i][0],my_x0,eps)
        && eq(my_detDim[i][1],my_y1,eps)
        && eq(my_detDim[i][2],my_z0,eps) )
    {
        my_idx010 = i;
    }
    if ( eq(my_detDim[i][0],my_x0,eps)
        && eq(my_detDim[i][1],my_y0,eps)
        && eq(my_detDim[i][2],my_z1,eps) )
    {
        my_idx001 = i;
    }
    if ( eq(my_detDim[i][0],my_x1,eps)
        && eq(my_detDim[i][1],my_y1,eps)
        && eq(my_detDim[i][2],my_z0,eps) )
    {
        my_idx110 = i;
    }
    if ( eq(my_detDim[i][0],my_x0,eps)
        && eq(my_detDim[i][1],my_y1,eps)
        && eq(my_detDim[i][2],my_z1,eps) )
    {
        my_idx011 = i;
    }
    if ( eq(my_detDim[i][0],my_x1,eps)
        && eq(my_detDim[i][1],my_y0,eps)
        && eq(my_detDim[i][2],my_z1,eps) )
    {
        my_idx101 = i;
    }
    if ( eq(my_detDim[i][0],my_x1,eps)
        && eq(my_detDim[i][1],my_y1,eps)
        && eq(my_detDim[i][2],my_z1,eps) )
    {
        my_idx111 = i;
    }
}

// keep tallies in scope outside of loop for efficiency
```

```
    // don't have to read files as often
    tal1 = tallyPtr( new tally(my_redFact) );
    tal3 = tallyPtr( new tally(my_redFact) );

    // build response function matrix
    computeResponse( maxErg );

    // perfect detector
    //my_I.resize( my_R.numergout(),my_R.numergin() );
    //my_I.ergout() = my_R.ergout();
    //my_I.ergin() = my_R.ergin();
    //my_I.identity();
    //my_I = my_I * my_equivArea;
    //my_R = my_I;

    // multiply by equivalent area because
    // cargo spectrum is in units of per unit area
    my_R = my_R * my_equivArea;

    //std::cout << "finished building detector response" << std::endl;

    return;
}


void detector::buildResponse ( double detYPos,
                               double maxErg,
                               int redFact )
{
    //std::cout << "building detector response" << std::endl;

    double detXPos = my_detXPos0;
    double detZPos = my_detZPos0;

    // set all indices to -1
    srcIdx1 = -1;
    srcIdx3 = -1;

    my_redFact = redFact;
    // store dimensions/position of detector
    my_xPos = detXPos;
    my_yPos = detYPos;
    my_zPos = detZPos;


    // hard-code in detector planes,
    // should read this in from file eventually
    const double tside = 129.54;
    const double ttop = 259.08;
    const double soff = 68.56;
    double my_detXPlane = tside+soff; // side of truck + standoff
    double my_detZPlane = ttop+soff; // side of truck + standoff

    // find out if this detector is a "side" or "top" detector
    if ( my_zPos < my_detZPlane && my_xPos > tside )
    {
        my_detPosType = side;
    }
    else if ( my_zPos > ttop && my_xPos < my_detXPlane )
    {
        my_detPosType = top;
    }
    else
    {
        throw fatal_error("invalid detector position");
    }

    // detector face's normals
    std::vector<double> nx(3,0.0);
    std::vector<double> ny(3,0.0);
    std::vector<double> nz(3,0.0);


    // detector dimension min/maxes
    // are different because detector is
    // always specified by front center face
```

```
double xMin,xMax,yMin,yMax,zMin,zMax;
if ( my_detPosType == side )
{
    xMin = detXPos;
    xMax = detXPos + my_detXDim0;
    yMin = detYPos - my_detYDim0/2.0;
    yMax = detYPos + my_detYDim0/2.0;
    zMin = detZPos - my_detZDim0/2.0;
    zMax = detZPos + my_detZDim0/2.0;
}
else /*if ( my_detPosType == top )*/
{
    xMin = detXPos - my_detXDim0/2.0;
    xMax = detXPos + my_detXDim0/2.0;
    yMin = detYPos - my_detYDim0/2.0;
    yMax = detYPos + my_detYDim0/2.0;
    zMin = detZPos;
    zMax = detZPos + my_detZDim0;
}

// calculate area of detector face
// this assumes the other sides are well shielded from
// background radiation
if ( my_detPosType == side )
{
    my_equivArea = (yMax-yMin)*(zMax-zMin);
}
else /*if ( my_detPosType == top )*/
{
    my_equivArea = (yMax-yMin)*(xMax-xMin);
}


// use unmodified detector sizes
my_detXDim = my_detXDim0;
my_detYDim = my_detYDim0;
my_detZDim = my_detZDim0;


// need to interpolate between all three
// dimensions so need six data points
// find six closest dimensions
for ( unsigned int i=1;i<my_detDim.size();++i )
{
    if ( my_detDim[i][0] >= my_detXDim
        && my_detDim[i-1][0] < my_detXDim )
    {
        my_x0 = my_detDim[i-1][0];
        my_x1 = my_detDim[i][0];
    }
    if ( my_detDim[i][1] >= my_detYDim
        && my_detDim[i-1][1] < my_detYDim )
    {
        my_y0 = my_detDim[i-1][1];
        my_y1 = my_detDim[i][1];
    }
    if ( my_detDim[i][2] >= my_detZDim
        && my_detDim[i-1][2] < my_detZDim )
    {
        my_z0 = my_detDim[i-1][2];
        my_z1 = my_detDim[i][2];
    }
}


const double eps = 1e-5;
for ( unsigned int i=0;i<my_detDim.size();++i )
{
    if ( eq(my_detDim[i][0],my_x0,eps)
        && eq(my_detDim[i][1],my_y0,eps)
        && eq(my_detDim[i][2],my_z0,eps) )
    {
        my_idx000 = i;
    }
    if ( eq(my_detDim[i][0],my_x1,eps)
        && eq(my_detDim[i][1],my_y0,eps)
```

```
            && eq(my_detDim[i][2],my_z0,eps) )
        {
            my_idx100 = i;
        }
        if ( eq(my_detDim[i][0],my_x0,eps)
            && eq(my_detDim[i][1],my_y1,eps)
            && eq(my_detDim[i][2],my_z0,eps) )
        {
            my_idx010 = i;
        }
        if ( eq(my_detDim[i][0],my_x0,eps)
            && eq(my_detDim[i][1],my_y0,eps)
            && eq(my_detDim[i][2],my_z1,eps) )
        {
            my_idx001 = i;
        }
        if ( eq(my_detDim[i][0],my_x1,eps)
            && eq(my_detDim[i][1],my_y1,eps)
            && eq(my_detDim[i][2],my_z0,eps) )
        {
            my_idx110 = i;
        }
        if ( eq(my_detDim[i][0],my_x0,eps)
            && eq(my_detDim[i][1],my_y1,eps)
            && eq(my_detDim[i][2],my_z1,eps) )
        {
            my_idx011 = i;
        }
        if ( eq(my_detDim[i][0],my_x1,eps)
            && eq(my_detDim[i][1],my_y0,eps)
            && eq(my_detDim[i][2],my_z1,eps) )
        {
            my_idx101 = i;
        }
        if ( eq(my_detDim[i][0],my_x1,eps)
            && eq(my_detDim[i][1],my_y1,eps)
            && eq(my_detDim[i][2],my_z1,eps) )
        {
            my_idx111 = i;
        }
    }

    // keep tallies in scope outside of loop for efficiency
    // don't have to read files as often
    tal1 = tallyPtr( new tally(my_redFact) );
    tal3 = tallyPtr( new tally(my_redFact) );

    // build response function matrix
    computeResponse( maxErg );

    // multiply by equivalent area because
    // cargo spectrum is in units of per unit area
    my_R = my_R * my_equivArea;

    //std::cout << "finished building detector response" << std::endl;
    return;
}


spectrum detector::operator() ( const spectrum& S_car )
{
    return my_R * S_car;
}

spectrum detector::applyDR( const dspectrum& S_car )
{
    return my_R * S_car;
}


spectrum detector::GEB( const spectrum& tal )
{
    //std::cout << "applying GEB" << std::endl;

    spectrum newtal(tal.numerg());
    newtal.erg() = tal.erg();
```

```
    double a = my_A;
    double b = my_B;
    double c = my_C;

    for ( int i=1;i<tal.numerg();++i )
    {
        if ( tal(i-1).get() > 1e-50 )
        {
            // use mean bin energy
            const double erg = (tal.erg(i)+tal.erg(i-1))/2.0;
            const double FWHM = a+b*sqrt(erg+c*pow(erg,2.0));
            const double sigma = FWHM/2.35482;

            // go +/- 4 sigmas
            const double mine = erg-4*sigma;
            const double maxe = erg+4*sigma;

            // get min/max index
            int minidx = 1;
            for ( int j=i;j>0;--j )
            {
                if ( mine > tal.erg(j) )
                {
                    minidx = j+1;
                    break;
                }
            }
            int maxidx = tal.numerg()-1;
            for ( int j=i;j<tal.numerg();++j )
            {
                if ( maxe < tal.erg(j) )
                {
                    maxidx = j;
                    break;
                }
            }

            // apply broadening to bins
            for ( int j=minidx;j<=maxidx;++j )
            {
                const double frac
                    = normcdf( tal.erg(j),erg,sigma )
                    - normcdf( tal.erg(j-1),erg,sigma );
                newtal(j-1) = newtal(j-1) + tal(i-1)*frac;
            }
        }
    }
    return newtal;
}
```

Listing B.12: detector.hpp

```
#ifndef _detector_hpp_included_
#define _detector_hpp_included_
#include <string>
#include <vector>
#include <tr1/memory>

#include "response.hpp"
#include "spectrum.hpp"
#include "submodel.hpp"


class alarma;


class detector : public submodel
{
    public:

    typedef std::tr1::shared_ptr<alarma> alarmPtr;

    int numInErgs ( );
    int numInErgs ( ) const;
    int numOutErgs ( );
    int numOutErgs ( ) const;

    double getX0( ) { return my_detXPos0; };
    double getY0( ) { return my_detYPos0; };
    double getZ0( ) { return my_detZPos0; };
    double getA( ) { return my_A; };
    double getB( ) { return my_B; };
    double getC( ) { return my_C; };

    enum dpos { side,top };

    void buildResponse ( double,double,double,double,double,int );
    void buildResponse ( double,double,int );

    void initialize ( const std::string& path );

    void addAlarm( alarmPtr a )
    {
        my_alarm.push_back(a);
    }
    int numAlarm() const { return static_cast<int>(my_alarm.size()); };
    alarmPtr alarm(int a) { return my_alarm[a]; };
    alarmPtr lastAlarm() { return my_alarm.back(); };

    detector ( const std::string& path,
        double detXPos,double detYPos,double detZPos,
        double detXDim,double detYDim,double detZDim,
        double eff,double A,double B,double C );
    detector ( ) { };


    spectrum operator() ( const spectrum& );
    spectrum applyDR( const dspectrum& );
    spectrum GEB( const spectrum& tal );

    private:

    std::string getTallyEnergyPath( int );
    std::vector<std::string> getTallyPath( int );
    std::vector<int> getRadiusIndex( int );
    std::vector<double> getRadius( int );
    std::vector<datapoint> interpolateTallies( double,
        const std::vector< double >&,int,int );



    void readDataFile();

    // original user specified detector dimensions
    double my_detXDim0;
    double my_detYDim0;
    double my_detZDim0;
```

```
    // original user specified positions
    double my_detXPos0;
    double my_detYPos0;
    double my_detZPos0;

    // effective dimensions
    double my_detXDim;
    double my_detYDim;
    double my_detZDim;

    // collection efficiency
    double my_eff;
    // GEB parameters
    double my_A;
    double my_B;
    double my_C;

    // equivalent area of detector based on solid angles
    double my_equivArea;

    // user specified detector position
    double my_xPos;
    double my_yPos;
    double my_zPos;
    // dimension indices on which to interpolate
    int my_idx000;
    int my_idx100;
    int my_idx010;
    int my_idx001;
    int my_idx110;
    int my_idx011;
    int my_idx101;
    int my_idx111;

    double my_x0, my_x1, my_y0, my_y1, my_z0, my_z1;

    dpos my_detPosType;

    // detector dimensions from data library
    std::vector< std::vector<double> > my_detDim;
    int my_nDim;

    // identity matrix that transforms incoming
    //  energies to outgoing energies
    response my_I;

    // parsed/interpolated tallies
    tallyPtr tal1;
    tallyPtr tal3;

    // parsed tallies indices
    int srcIdx1;
    int srcIdx3;


    std::vector<alarmPtr> my_alarm;

};

#endif
```

Listing B.13: diagnostic.cpp

```
#include "diagnostic.hpp"
#include "errh.hpp"

namespace global
{
    double init;
    double scat;
    double brem;
    double intrpscat;
    double intrpbrem;
    double intrpxray;
    double intrpannh;
    double intrpuncl;
    double intrp;
    double assign;

    warningmsg warn;

    fatalmsg fatal;

    void warningmsg::add( std::string msg )
    {
        my_msg.push_back(msg);
    }

    void warningmsg::flush( )
    {
        for ( unsigned int i=0;i<my_msg.size();++i )
        {
            std::cout << "warning: " << my_msg[i] << std::endl;
        }
        my_msg.clear();
    }

    void fatalmsg::flush( )
    {
        if ( my_msg.size() == 1 )
        {
            throw fatal_error(my_msg[0]);
        }
        for ( unsigned int i=0;i<my_msg.size();++i )
        {
            std::cout << "fatal_error: " << my_msg[i] << std::endl;
        }
        my_msg.clear();
        if ( my_msg.size() > 1 )
        {
            throw fatal_error("multiple fatal errors occurred");
        }
    }

}
```

Listing B.14: diagnostic.hpp

```
#ifndef _diagnostic_hpp_included_
#define _diagnostic_hpp_included_

#define echo(variable) std::cout <<#variable": "<<(variable)<<std::endl;

#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <ctime>
#include <string>
#include <vector>

namespace global
{
    extern double init;
    extern double scat;
    extern double brem;
    extern double intrpscat;
    extern double intrpbrem;
    extern double intrpxray;
    extern double intrpannh;
    extern double intrpuncl;
    extern double intrp;
    extern double assign;

    class warningmsg
    {
        public:

        void add( std::string msg );

        virtual void flush( );

        protected:

        std::vector<std::string> my_msg;
    };

    class fatalmsg : public warningmsg
    {
        public:

        void flush( );

        private:
    };

    extern warningmsg warn;
    extern fatalmsg fatal;
}


class timer
{
    public:

    timer( )
    {
        my_totalTime = 0.0;
    }

    void start( )
    {
        my_clock = std::clock();
    }
    void stop( )
    {
        my_totalTime += ( ( std::clock() - my_clock )
            / ( double ) CLOCKS_PER_SEC );
    }

    double getTime( )
    {
        return my_totalTime;
    }
```

```
    void print( )
    {
        std::cout << std::setw ( 20 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 3 ) << my_totalTime
            << " seconds " << std::endl;;
    }


    private:

    double my_totalTime;
    clock_t my_clock;

};


#endif
```

Listing B.15: dspectrum.cpp

```
#include "dspectrum.hpp"

#include <iostream>
#include <iomanip>
#include <algorithm>

#include "matrix.hpp"


void dspectrum::resize ( int numergs )
{
    my_data.resize ( numergs );
    my_energy.resize ( numergs );
    return;
}

void dspectrum::print ( )
{

    for ( int i=0;i<numdat();++i )
    {
        std::cout << std::setw ( 15 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_energy[i];
        std::cout << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_data[i].get();
        std::cout << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_data[i].getErr()
            << std::endl;
    }
    std::cout << std::endl;
    std::cout << std::setw ( 15 ) << "Total"
        << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 3 ) << sum();
    return;
}


void dspectrum::print ( std::ofstream& outfile,
                        const std::string& header,
        const std::vector<std::string>& title,
  const std::vector<std::string>& unit )
{
    outfile << "NUMERGS_" << numerg() << std::endl;

    outfile << std::endl << std::endl;
    outfile << header << std::endl << std::endl;
    outfile << std::setw ( 15 ) << title[0];
    outfile << std::setw ( 25 ) << title[1];
    outfile << std::setw ( 25 ) << title[2] << std::endl;
    std::string newunit = "[" + unit[0] + "]";
    outfile << std::setw ( 15 ) << newunit;
    newunit = "[" + unit[1] + "]";
    outfile << std::setw ( 25 ) << newunit;
    newunit = "[" + unit[2] + "]";
    outfile << std::setw ( 25 ) << newunit << std::endl;

    for ( int i=0;i<numdat();++i )
    {
        outfile << std::setw ( 15 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_energy[i];
        outfile << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_data[i].get();
        outfile << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 ) << my_data[i].getErr()
            << std::endl;
    }
    outfile << std::endl;
```

```
    outfile << std::setw ( 15 ) << "Total"
            << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 3 ) << sum();
    return;
}


dspectrum::dspectrum ( int numergs )
{
    my_isSorted = false;
    initialize ( numergs );
}

dspectrum::dspectrum ( )
{
    my_isSorted = false;
}

dspectrum::dspectrum ( int numergs,bool sorted )
{
    initialize ( numergs );
    my_isSorted = sorted;
}

dspectrum::dspectrum ( bool sorted )
{
    my_isSorted = sorted;
}

void dspectrum::sort()
{
    if ( ! my_isSorted )
    {
        std::sort( my_data.begin(),my_data.end() );
    }
    my_isSorted = true;
}

void dspectrum::initialize ( int numergs )
{
    my_data.resize ( numergs );
    my_energy.resize ( numergs );
    return;
}

bool dspectrum::isSorted( ) const
{
    return my_isSorted;
}


void dspectrum::operator= ( const dspectrum& a )
{
    erg() = a.erg();
    data() = a.data();
    return;
}

void dspectrum::operator= ( const datapoint& a )
{
    for ( int i=0;i<numdat();++i )
    {
        my_data[i] = a;
    }
    return;
}

void dspectrum::operator= ( double a )
{
    for ( int i=0;i<numdat();++i )
    {
        my_data[i].set ( a );
        my_data[i].set ( 0.0 );
    }
```

```
    return;
}


dspectrum operator+ ( const dspectrum& a,const dspectrum& b )
{
    // These energies should be sorted
    dspectrum result( a.numerg() + b.numerg() );
    for ( int i=0;i<a.numerg();++i )
    {
        result.erg(i) = a.erg(i);
        result(i) = a(i);
    }
    for ( int i=a.numerg();i<a.numerg()+b.numerg();++i )
    {
        result.erg(i) = b.erg(i-a.numerg());
        result(i) = b(i-a.numerg());
    }
    return result;
}

dspectrum operator* ( const dspectrum& a,double b )
{
    dspectrum result;
    result.erg() = a.erg();
    result.data() = a.data() *b;
    return result;
}

dspectrum operator* ( double b,const dspectrum& a )
{
    return a*b;
}
```

Listing B.16: dspectrum.hpp

```
#ifndef _dspectrum_hpp_included_
#define _dspectrum_hpp_included_

#include <vector>
#include <string>
#include <fstream>

#include "spectrum.hpp"


class dspectrum : public spectrum
{
    public:

        void resize ( int );

        void print ( std::ofstream&,const std::string&,
            const std::vector<std::string>&,
            const std::vector<std::string>& );
        void print ( );

        void initialize ( int );

        dspectrum ( );
        dspectrum ( int );

        dspectrum ( bool );
        dspectrum ( int,bool );

        void sort();

        //bool isSorted();
        bool isSorted() const;

    void operator= ( const dspectrum& );
    void operator= ( const datapoint& );
    void operator= ( double );


    protected:

    bool my_isSorted;

};

dspectrum operator+ ( const dspectrum&,const dspectrum& );

dspectrum operator* ( const dspectrum&,double );

dspectrum operator* ( double,const dspectrum& );


#endif
```

Listing B.17: extras.cpp

```cpp
#include "extras.hpp"
#include "fileio.hpp"
#include <iostream>
#include <assert.h>
#include <map>
#include "matrix.hpp"
#include "phys.hpp"

double assurePositivity( double a )
{
    if ( isinf(a) || isnan(a) || a < 0.0 )
    {
        return 0.0;
    }
    else
    {
        return a;
    }
}

int find( const std::vector< double >& vec,double key )
{
    // search for energy erg in the my_erg
    // array using a binary algorithm
    // first index
    int first = 1;
    // last index
    int last = vec.size()-1;
    int oldmp = 0;
    // Check for keys that are outside of the range of values
    if ( key < vec[1] )
    {
        return 2;
    }
    if ( key > *(vec.rbegin()) )
    {
        return static_cast<int>(vec.size()-1);
    }
    while ( true )
    {
        // compute midpoint (average)
        int mp = (last+first)/2;
        if ( mp == oldmp )
        {
            mp++;
        }
        oldmp = mp;

        // check if bounded by mp-1 and mp
        //if ( key > vec[mp-1] && key <= vec[mp] )

        if ( key-vec[mp] <= 1e-10 )
        {
            if ( key > vec[mp-1] )
            {
                return mp;
            }
            else
            {
                // bisect domain downard
                // use (-1) because we know upper bound can't be mp
                last = mp-1;
            }
        }
        else
        {
            // bisect domain upward
            first = mp;
        }
        if ( first == last ) // then we are stuck somewhere
        {
            return mp;
        }
    }
}
```

```
double normcdf( double x )
{
    return 0.5 + 0.5*erf(x/sqrt(2));
}

double normcdf( double x,double mu,double sig )
{
    return normcdf((x-mu)/sig);
}


double distance( double x1,double y1,double z1,
    double x2,double y2,double z2 )
{
    return sqrt( pow(x2-x1,2.0)+pow(y2-y1,2.0)+pow(z2-z1,2.0) );
}

// calculate the solid angle (in steradians) of a triangle
// absolute coordinates of source
// absolute coordinates of three points
double solidAngle ( double r0x,double r0y,double r0z,
                    double d1x,double d1y,double d1z,
                    double d2x,double d2y,double d2z,
                    double d3x,double d3y,double d3z )
{
    // calculate the area of the triangle using spherical excess
    double a,b,c;

    // d1,d3 form angle alpha
    a = sqrt ( (d3x-r0x)*(d3x-r0x) + (d3y-r0y)*(d3y-r0y)
        + (d3z-r0z)*(d3z-r0z) );
    b = sqrt ( (d1x-r0x)*(d1x-r0x) + (d1y-r0y)*(d1y-r0y)
        + (d1z-r0z)*(d1z-r0z) );
    c = sqrt ( (d3x-d1x)*(d3x-d1x) + (d3y-d1y)*(d3y-d1y)
        + (d3z-d1z)*(d3z-d1z) );
    const double alpha = acos ( ( a*a + b*b - c*c )
        / ( 2.0*a*b ) );

    // d1,d2 form angle beta
    a = sqrt ( (d2x-r0x)*(d2x-r0x) + (d2y-r0y)*(d2y-r0y)
        + (d2z-r0z)*(d2z-r0z) );
    b = sqrt ( (d1x-r0x)*(d1x-r0x) + (d1y-r0y)*(d1y-r0y)
        + (d1z-r0z)*(d1z-r0z) );
    c = sqrt ( (d2x-d1x)*(d2x-d1x) + (d2y-d1y)*(d2y-d1y)
        + (d2z-d1z)*(d2z-d1z) );
    const double beta = acos ( ( a*a + b*b - c*c )
        / ( 2.0*a*b ) );

    // d3,d2 form angle gamma
    a = sqrt ( (d2x-r0x)*(d2x-r0x) + (d2y-r0y)*(d2y-r0y)
        + (d2z-r0z)*(d2z-r0z) );
    b = sqrt ( (d3x-r0x)*(d3x-r0x) + (d3y-r0y)*(d3y-r0y)
        + (d3z-r0z)*(d3z-r0z) );
    c = sqrt ( (d2x-d3x)*(d2x-d3x) + (d2y-d3y)*(d2y-d3y)
        + (d2z-d3z)*(d2z-d3z) );
    const double gamma = acos ( ( a*a + b*b - c*c )
        / ( 2.0*a*b ) );


    // calculate sines and cosines of these angles
    const double cos_a = cos ( alpha );
    const double cos_b = cos ( beta );
    const double cos_g = cos ( gamma );
    const double sin_a = sin ( alpha );
    const double sin_b = sin ( beta );
    const double sin_g = sin ( gamma );

    // calculate angles of triangles projected on sphere
    // using law of cosines (spherical)
    const double A  = acos ( ( cos_a-cos_g*cos_b )
        / ( sin_g*sin_b ) );
    const double B  = acos ( ( cos_b-cos_a*cos_g )
        / ( sin_a*sin_g ) );
    const double G  = acos ( ( cos_g-cos_b*cos_a )
        / ( sin_b*sin_a ) );
```

```
    return ( A + B + G - phys::pi );
}


double solidAngle ( const std::vector<double>& r,
                    double width,
                    double height )
{
    // height boundaries for detector
    const double z1 = fabs ( r[2] ) - height/2.0;
    const double z2 = fabs ( r[2] ) + height/2.0;

    // cartesian distances to differential area from center
    // of detector
    const double x = fabs ( r[0] );
    const double y = fabs ( r[1] );
    const double xsq = x*x;

    // detector dimensions
    // width
    const double y1 = y - width/2.0;
    const double y2 = y + width/2.0;
    const double y1sq = y1*y1;
    const double y2sq = y2*y2;

    const double h1 = z1;
    const double h2 = z2;
    const double h1sq = h1*h1;
    const double h2sq = h2*h2;

    // split up detector face into two triangles, with diagonal
    // running from bottom-left to top-right
    // calculate the area of each triangle using spherical excess


    double a,b,c;
    // calculate right side of detector angle, theta_right
    a = sqrt ( xsq + y2sq + h2sq );
    b = sqrt ( xsq + y2sq + h1sq );
    c = h2 - h1;
    const double th_r = acos ( ( a*a + b*b - c*c ) / ( 2.0*a*b ) );
    // calculate left side of detector, theta_left (c is same)
    a = sqrt ( xsq + y1sq + h2sq );
    b = sqrt ( xsq + y1sq + h1sq );
    const double th_l = acos ( ( a*a + b*b - c*c ) / ( 2.0*a*b ) );

    // calculate top side of detector angle, theta_top
    a = sqrt ( xsq + y2sq + h2sq );
    b = sqrt ( xsq + y1sq + h2sq );
    c = y2 - y1;
    const double th_t = acos ( ( a*a + b*b - c*c ) / ( 2.0*a*b ) );
    // calculate bottom side of detector angle, theta_bottom
    a = sqrt ( xsq + y2sq + h1sq );
    b = sqrt ( xsq + y1sq + h1sq );
    const double th_b = acos ( ( a*a + b*b - c*c ) / ( 2.0*a*b ) );

    // calculate detector diagonal angle, theta_diagonal
    a = sqrt ( xsq + y1sq + h2sq );
    b = sqrt ( xsq + y2sq + h1sq );
    c = sqrt ( ( h2-h1 ) * ( h2-h1 ) + ( y2-y1 ) * ( y2-y1 ) );
    const double th_d = acos ( ( a*a + b*b - c*c ) / ( 2.0*a*b ) );

    // calculate sines and cosines of these angles
    const double cos_r = cos ( th_r );
    const double cos_l = cos ( th_l );
    const double cos_t = cos ( th_t );
    const double cos_b = cos ( th_b );
    const double cos_d = cos ( th_d );
    const double sin_r = sin ( th_r );
    const double sin_l = sin ( th_l );
    const double sin_t = sin ( th_t );
    const double sin_b = sin ( th_b );
    const double sin_d = sin ( th_d );
```

```
    // calculate angles of triangles projected on sphere
    // using law of cosines (spherical)
    const double R  = acos ( ( cos_r-cos_t*cos_d ) / ( sin_t*sin_d ) );
    const double L  = acos ( ( cos_l-cos_b*cos_d ) / ( sin_b*sin_d ) );
    const double T  = acos ( ( cos_t-cos_r*cos_d ) / ( sin_r*sin_d ) );
    const double B  = acos ( ( cos_b-cos_l*cos_d ) / ( sin_l*sin_d ) );
    const double D1 = acos ( ( cos_d-cos_r*cos_t ) / ( sin_r*sin_t ) );
    const double D2 = acos ( ( cos_d-cos_l*cos_b ) / ( sin_l*sin_b ) );

    return ( R + T + D1 + L + B + D2 - 2.0*phys::pi );
}


std::vector<double> LU_Solve (
    matrix<double> &a, std::vector<double> &b )
{

    const int N = b.size();
    matrix<double> temp = a;

    //double const BIG = std::numeric_limits<double>::max();
    for ( int k=0; k< ( N-1 ); ++k )
    {
        for ( int i= ( k+1 ); i<N; ++i )
        {
            double const m = temp(i,k)/temp(k,k);
            temp(i,k) = m;
            for ( int j= ( k+1 ); j<N; ++j )
            {
                temp(i,j) = temp(i,j)-m*temp(k,j);
            }
        }
    }

    std::vector<double> sol(b);

    for ( int i=1; i<N; ++i )
    {
        for ( int j=0; j<= ( i-1 ); ++j )
        {
            sol[i] = sol[i] - temp(i,j)*sol[j];
        }
    }

    sol[N-1] = sol[N-1]/temp(N-1,N-1);
    for ( int i= ( N-2 ); i>=0; --i )
    {
        for ( int j= ( N-1 ); j>= ( i+1 ); --j )
        {
            sol[i] = sol[i] - temp(i,j)*sol[j];
        }
        sol[i] = sol[i]/temp(i,i);
    }
    return sol;
}

std::vector<double> LeastSquaresFit (
    std::vector<double>& Y, std::vector<double>& X, int order )
{
    assert ( Y.size() == X.size() );
    const int npts = Y.size();
    const int N = order + 1;

    matrix<double> A(N,N);
    std::vector<double> b ( N );


    for ( int i=0; i<npts; ++i )
    {
        for ( int j=0; j<N; ++j )
        {
            for ( int k=0; k<N; ++k )
            {
                A(N-j-1,N-k-1) = A(N-j-1,N-k-1)
                    + pow ( X[i],2.0* ( N-1 )-j-k );
            }
```

```
            b[N-j-1] = b[N-j-1] + Y[i]*pow ( X[i],1.0*(N-1-j) );
        }
    }

    std::vector<double> sol = LU_Solve ( A,b );

    return sol;
}

std::vector<double> ExpLeastSquaresFit (
    std::vector<double>& Y, std::vector<double>& X )
{
    assert ( Y.size() == X.size() );
    const int npts = Y.size();
    const int N = 2;

    matrix<double> A(N,N);
    std::vector<double> b ( N );


    for ( int i=0; i<npts; ++i )
    {
        A(0,0) += Y[i];
        A(0,1) += X[i]*Y[i];
        A(1,0) += X[i]*Y[i];
        A(1,1) += X[i]*X[i]*Y[i];

        b[0] += Y[i]*log(Y[i]);
        b[1] += X[i]*Y[i]*log(Y[i]);


        //for ( int j=0; j<N; ++j )
        //{
            //for ( int k=0; k<N; ++k )
            //{
                //A[N-j-1][N-k-1] +=  Y[i]
                //   * pow ( X[i],2* ( N-1 )-j-k );
            //}
            //b[N-j-1] += log ( Y[i] ) * Y[i] * pow ( X[i],N-1-j );
        //}
    }

    std::vector<double> sol = LU_Solve ( A,b );

    sol[0] = exp ( sol[0] );
    return sol;
}

int sign( double x )
{
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}


double erfinv ( double z )
{
    const double pi = phys::pi;
    const double a =0.140012;
    return sign(z)*sqrt(sqrt( pow(2.0/(pi*a)+log(1-z*z)/2.0,2.0)
        -log(1-z*z)/a) - (2.0/(pi*a)+log(1-z*z)/2.0));
    //return ( 1.0/2.0 ) *sqrt ( pi ) * ( z+ ( pi/12.0 ) *pow ( z,3.0 )
    //   +7*pow ( pi,2.0 ) /480*pow ( z,5.0 )+127*pow ( pi,3.0 )
    //   /40320*pow ( z,7.0 ) +4369*pow ( pi,4.0 )
    //   /5806080*pow ( z,9.0 ) );
}


template < class T >
std::vector< T > logspace( T minVal,T maxVal,int numSteps )
{
    std::vector< T > value( numSteps );
    for ( int i=0;i<numSteps;++i )
    {
        value[i] = exp( log( minVal ) + i
```

```
            *( log( maxVal ) - log( minVal ) ) /(numSteps-1)   );
    }
    return value;
}

template std::vector<double> logspace<double>(double,double,int);

template < class T >
std::vector< T > linspace( T minVal,T maxVal,int numSteps )
{
    std::vector< T > value( numSteps );
    for ( int i=0;i<numSteps;++i )
    {
        value[i] = minVal + i *( maxVal -minVal ) /(numSteps-1);
    }
    return value;
}

template std::vector<double> linspace<double>(double,double,int);

int pow ( int base, int power )
{
    if ( power == 0 ) return 1;
    if ( power == 1 ) return base;
    if ( power % 2 == 0 ) return pow ( base * base, power / 2 );
    if ( power % 2 == 1 ) return base * pow ( base * base, power / 2 );
    return 1;
}

int factorial ( int n )
{
    if ( n == 0 ) return 1;
    return n*factorial(n-1);
}
```

Listing B.18: extras.hpp

```
#ifndef _extras_hpp_included_
#define _extras_hpp_included_

#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <istream>
#include <limits>
#include <utility>
#include <cmath>
#include <iostream>
#include <tr1/memory>
#include "errh.hpp"
#include "datapoint.hpp"
#include "tally.hpp"
#include "matrix.hpp"



double assurePositivity( double a );

double normcdf( double x );

double normcdf( double x,double mu,double sig );

int find( const std::vector< double >& vec,double key );

double distance( double,double,double,double,double,double );

template <class T>
bool eq( T a,T b,T eps )
{
    return fabs(a-b) < eps;
}

double solidAngle ( const std::vector<double>&,double,double );

double solidAngle ( double r0x,double r0y,double r0z,
                    double d1x,double d1y,double d1z,
                    double d2x,double d2y,double d2z,
                    double d3x,double d3y,double d3z );

std::vector<double> LU_Solve ( matrix<double> &a,
    std::vector<double> &b );

std::vector<double> LeastSquaresFit ( std::vector<double>&,
    std::vector<double>&, int );

std::vector<double> ExpLeastSquaresFit ( std::vector<double>&,
    std::vector<double>& );

double erfinv ( double );

int pow ( int base, int power );

int factorial( int );


template < class T >
void mySize ( std::vector< T >& v, int a)
{
    v.resize(a);
    return;
}

template < class T >
void mySize ( std::vector< std::vector< T > >& v, int a, int b )
{
    v.resize(a);
    for ( int i=0;i<a;++i )
    {
        v[i].resize(b);
```

```
    }
    return;
}

template < class T >
void mySize ( std::vector< std::vector< std::vector< T > > >& v,
    int a, int b, int c )
{
    v.resize(a);
    for ( int i=0;i<a;++i )
    {
        v[i].resize(b);
        for ( int j=0;j<b;++j )
        {
            v[i][j].resize(c);
        }
    }
    return;
}

template < class T >
void mySize (
    std::vector< std::vector< std::vector< std::vector< T > > > >& v,
    int a, int b, int c, int d )
{
    v.resize(a);
    for ( int i=0;i<a;++i )
    {
        v[i].resize(b);
        for ( int j=0;j<b;++j )
        {
            v[i][j].resize(c);
            for ( int k=0;k<c;++k )
            {
                v[i][j][k].resize(d);
            }
        }
    }
    return;
}

template < class T >
extern std::vector< T > logspace( T minVal,T maxVal,int numSteps );

template < class T >
extern std::vector< T > linspace( T minVal,T maxVal,int numSteps );

template < class T >
void swap( T& a,T& b )
{
    T temp = a;
    a = b;
    b = temp;
}

#endif
```

Listing B.19: gammalines.cpp

```
#include "gammalines.hpp"
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <limits>
#include "radsource.h"
#include "mcinput.h"
#include "useriface.h"
#include "dbmanager.h"
#include "datapoint.hpp"
#include "matrix.hpp"

// NOTE
//
// Most of this file is logic adopted from the
// radsrc source code. I was too lazy to figure
// out a proper API.

using namespace radsrc;
void PrepareMCInput(const radsrc::CRadSource& radsource,
    const Options& options, MCInput& mci);
void InputConfig(const std::string&,CRadSource& radsource,
    Options& options, std::ostream& input_log);


spectrum getBremsstrahlung( const std::string& inputstring )
{
    //std::cout << "RADSRC_HOME is "
    //  << getenv("RADSRC_HOME") << std::endl;

    if (CDatabaseManager::getIsotopeDatabase(
        CDatabaseManager::LEGACY) == 0) {
      std::cerr
         << "Unable to load database. Exiting..."
         << std::endl;
      return -1;
    }
    //std::cout << "loaded database" << std::endl;
    Options options;

    // Set default XPASS options desired
    // no command line interactions or output
    options.interactive = false;
    options.quiet = true;
    // no monte carlo input decks
    options.do_mci = false;
    // set source energy range from 10 keV to 3.2 MeV
    // i don't think these have any effect
    options.source_min = 10.0; // keV
    options.source_max = 3200.0; // keV
    // set tally energy to same
    options.tallyu_min = options.source_min;
    options.tallyu_max = options.source_max;
    // no binning of lines, keep everything discrete
    options.lines = BIN_NONE;

    CRadSource radsource;
    std::ifstream filestream;
    std::ostringstream input_log;

    std::stringstream ss;
    ss << inputstring << " ";
    bool success = radsource.loadConfig(ss);
    if ( ! success )
    {
        throw fatal_error("error in initial isotopic mixture");
    }
    options.sample_brem = radsource.getPhotonComputer().sampleBrem();
    options.brembin_options.filename  = "dfltbrem.dat";
    //std::cout << "reading bins" << std::endl;
    //SetBins(radsource,options.brembin_options,
    //  CPhotonComputer::BIN_BREM);
    //void SetBins(CRadSource& radsource,
    //  const Options::SubOptions& options,
    //  CPhotonComputer::BinSubject what) {
```

```
    std::vector<double> vtmp;
    CPhotonComputer& pc = radsource.getPhotonComputer();
    //for (int foo = 0; foo < 2; foo++)
    //{
        //pc.setBinning(CPhotonComputer::BIN_BREM,
        //  DefaultBins,NDEFAULTBINS+1);
        ReadBins(options.brembin_options.filename,vtmp,PF_DEFAULT);
        pc.setBinning(CPhotonComputer::BIN_BREM,vtmp);
    //}



    // get bremsstrahlung
    //std::cout << "computing gammas" << std::endl;
    radsource.getPhotonComputer().computeGammas();

    //std::cout << "Total Bremmstrahlung Intensity: "
    //  << radsource.getPhotonComputer().getBremIntensity()
    //  << " ph/s/gm" << std::endl;

    //MCInput mci;
    //PrepareMCInput(radsource, options, mci);
    // convert to spectrum
    //std::vector< double > erg = mci.vBremBinBoundaries;
    std::vector< double > erg =
        radsource.getPhotonComputer().getBrem().m_energy;
    erg = erg/1000.0; // convert to MeV
    std::vector< datapoint > val;
    for ( unsigned int i=1;i<erg.size();++i )
    {
        val.push_back( datapoint(
            radsource.getPhotonComputer().getBrem().m_intensity[i],
            0.0 ) );
    }

    spectrum result;
    result.erg() = erg;
    result.data() = val;

    return result;
}


dspectrum getGammaLines( const std::string& inputstring,
    double minErg,double maxErg )
{
    //std::cout << "RADSRC_HOME is "
    //  << getenv("RADSRC_HOME") << std::endl;

    if (CDatabaseManager::getIsotopeDatabase(
        CDatabaseManager::LEGACY) == 0) {
      std::cerr
        << "Unable to load database. Exiting..."
        << std::endl;
      return -1;
    }

    Options options;

    // Set default XPASS options desired
    // no command line interactions or output
    options.interactive = false;
    options.quiet = true;
    // no monte carlo input decks
    options.do_mci = false;
    // set source energy range from 10 keV to 3.2 MeV
    // i don't think these have any effect
    options.source_min = 1.0; // keV
    options.source_max = 3200.0; // keV
    // set tally energy to same
    options.tallyu_min = options.source_min;
    options.tallyu_max = options.source_max;
    // no binning of lines, keep everything discrete
    options.lines = BIN_NONE;

    CRadSource radsource;
```

```
    std::ifstream filestream;
    std::ostringstream input_log;

    std::stringstream ss;
    ss << inputstring << " ";
    bool success = radsource.loadConfig(ss);
    if ( ! success )
    {
        throw fatal_error("error in initial isotopic mixture");
    }
    //options.sample_brem
    //  = radsource.getPhotonComputer().sampleBrem();
    //radsource.getPhotonComputer().getLinesRange(
    //  options.source_min,options.source_max);
    //InputConfig(inputstring,radsource, options, input_log);



    // Create vector of energies and vector of intensities
    //
    std::vector< double > erg;
    std::vector< datapoint > val;
    // this block is adapted from WriteLinesFile()
    //  function in $RADSRC_HOME/src/radsrc/radsrc.cc
    // get gamma lines
    radsource.getPhotonComputer().computeGammas(
        CPhotonComputer::ENERGY);

    // convert gamma lines to discrete spectrum
    CPhotonComputer::CPhotonIterator it, endit;
    it = radsource.getPhotonComputer().beginGammas();
    endit = radsource.getPhotonComputer().endGammas();
    while (it != endit)
    {
        //convert keV to MeV
        const double MeVErg = it->getEnergy()/1000.0;
        // only add it if the intensity is non-zero
        if ( it->getIntensity() > std::numeric_limits<double>::min()
            && MeVErg >= minErg && MeVErg <= maxErg )
        {
            erg.push_back( MeVErg );
            val.push_back( datapoint(it->getIntensity(),0.0) );
        }
        ++it;
    }

    dspectrum result(true); // radsrc always sorts the energies
    result.erg() = erg;
    result.data() = val;

    return result;
}



void InputConfig(const std::string& inputstring,
    CRadSource& radsource, Options& options,
    std::ostream& input_log)
{
   std::stringstream ss;
   ss << inputstring;
   ss.seekg(0);
   bool success;
   success = radsource.loadConfig(ss);

   options.sample_brem
    = radsource.getPhotonComputer().sampleBrem();
   radsource.getPhotonComputer().getLinesRange(
    options.source_min,options.source_max);
}
```

Listing B.20: gammalines.hpp

```
#ifndef _gammalines_hpp_included_
#define _gammalines_hpp_included_


#include "dspectrum.hpp"
#include "spectrum.hpp"

dspectrum getGammaLines( const std::string& inputstring,
    double minErg,double maxErg );
spectrum getBremsstrahlung( const std::string& inputstring );


#endif
```

Listing B.21: interpolation.cpp

```
#include "interpolation.hpp"

#include "diagnostic.hpp"

#include "matrix.hpp"
#include "phys.hpp"

int interpolatorbase::find( const std::vector< double >& vec,double key )
{
    // linear search for energy key in the vec array
    for ( unsigned int i=0;i<vec.size();++i )
    {
        if ( vec[i] >= key )
        {
            return i;
        }
    }
    return vec.size()-1;


    // search for energy erg in the my_erg array using a binary algorithm
    // first index
    // don't want to use first energy bin (1 keV),
    // because it can't upperbound anything
    int first = 1;
    // last index
    int last = vec.size()-1;
    int oldmp = 0;
    // Check for keys that are outside of the range of values
    if ( key < vec[1] )
    {
        return 2;
    }
    if ( key > *(vec.rbegin()) )
    {
        return static_cast<int>(vec.size()-1);
    }
    while ( true )
    {
        // compute midpoint (average)
        int mp = (last+first)/2;
        // if they are the same, then we aren't
        // rounding up, so force it
        if ( mp == oldmp )
        {
            mp++;
        }
        oldmp = mp;

        // check if bounded by mp-1 and mp
        //if ( key > vec[mp-1] && key <= vec[mp] )

        if ( key-vec[mp] <= 1e-10 )
        {
            if ( key > vec[mp-1] )
            {
                return mp;
            }
            else
            {
                // bisect domain downard
                // use (-1) because we know upper bound can't be mp
                last = mp-1;
            }
        }
        else
        {
            // bisect domain upward
            first = mp;
        }
        if ( first == last ) // then we are stuck somewhere
        {
            return mp;
        }
    }
```

```
}

int interpolatorbase::find( const std::vector< double >& vec,
    double key,int lastidx )
{
    // search for energy key in the vec array using
    // lastidx as a starting point
    // first check if the last index works
    if ( vec[lastidx] >= key && vec[lastidx-1] < key )
    {
        return lastidx;
    }
    for ( unsigned int i=lastidx;i<vec.size();++i )
    {
        if ( vec[i] >= key )
        {
            //std::cout << "they match!"<<std::endl;
            return i;
        }
    }
    return vec.size()-1;
}


std::vector< double > detinterpolator::transformdetscat(
    const std::vector< double >& E1,double comperg1,int compidx1,
    double E1p,int E1pidx,double comperg2,double E2p )
{
    std::vector<double> E1_1;
    std::vector<double> E1_2;
    if ( comperg1 > E1[0] && comperg2 > E1[0] )
    {
        E1_1.assign( E1.begin(),E1.begin()+compidx1 );
        E1_1 = lintran( E1_1,E1[0],comperg1,E1[0],comperg2 );
        E1_2.assign( E1.begin()+compidx1,E1.end() );
        E1_2 = lintran( E1_2,comperg1,E1p,comperg2,E2p );
    }
    else
    {
        E1_2.assign( E1.begin(),E1.end() );
        E1_2 = lintran( E1_2,E1[0],E1p,E1[0],E2p );
    }
    std::vector< double > E1_scat;
    E1_scat.reserve( E1pidx+1 );
    E1_scat.insert( E1_scat.end(),E1_1.begin(),E1_1.end() );
    E1_scat.insert( E1_scat.end(),E1_2.begin(),E1_2.end() );

    if ( E1_scat.size() != E1.size() )
    {
        std::cout << "E1_scat.size() = "
            << E1_scat.size() << std::endl;
        std::cout << "E1.size() = " << E1.size() << std::endl;
        throw fatal_error("detector energy transformation \
size mismatch");
    }

    return E1_scat;
}

std::vector< double > interpolatorbase::transformscat(
    const std::vector< double >& E1,double comperg1,int compidx1,
    double E1p,int E1pidx,double comperg2,double E2p )
{
    // Split energy at compton edge
    std::vector< double > E1_1;
    std::vector< double > E1_2;

    E1_1.assign( E1.begin(),E1.begin()+compidx1 );
    E1_2.assign( E1.begin()+compidx1,E1.end() );
    E1_1 = lintran( E1_1,E1[0],comperg1,E1[0],comperg2 );
    E1_2 = lintran( E1_2,comperg1,E1p,comperg2,E2p );


    // Recombine into transformed scattering energy
    std::vector< double > E1_scat;
    E1_scat.reserve( E1pidx+1 );
```

```
    E1_scat.insert( E1_scat.end(),E1_1.begin(),E1_1.end() );
    E1_scat.insert( E1_scat.end(),E1_2.begin(),E1_2.end() );

    if ( E1_scat.size() != E1.size() )
    {
        std::cout << "E1_scat.size() = "
            << E1_scat.size() << std::endl;
        std::cout << "E1.size() = " << E1.size() << std::endl;
        throw fatal_error("energy transformation size mismatch");
    }


    return E1_scat;
}

datapoint interpolatorbase::interpolateDiscreteBins(
    const std::vector<double>& erg1,
    const std::vector<datapoint>& tal1,
    const std::vector<double>& erg3,
    const std::vector<datapoint>& tal3,
    const std::vector<double>& erg2,
    int idx2,
    std::vector<double> slope=std::vector<double>() )
{
    // Find index of upper bin boundary
    int idx1 = 0;
    int idx3 = 0;

    idx1 = find( erg1,erg2[idx2],lastidx1 );
    lastidx1 = idx1;
    idx3 = find( erg3,erg2[idx2],lastidx3 );
    lastidx3 = idx3;

    datapoint p1;
    datapoint p3;

    // just use one bin and scale by original energy width
    p1 = tal1[idx1]*( erg1[idx1] - erg1[idx1-1] );
    p3 = tal3[idx3]*( erg3[idx3] - erg3[idx3-1] );

    // now interpolate these based on source energies
    //datapoint result = logInterpolate( p1,p3,ergdist );
    datapoint result = linearInterpolate( p1,p3,my_ergDist );

    return result;
}



datapoint interpolatorbase::interpolateBins(
    const std::vector<double>& erg1,
    const std::vector<datapoint>& tal1,
    const std::vector<double>& erg3,
    const std::vector<datapoint>& tal3,
    const std::vector<double>& erg2,
    int idx2,
    std::vector<double> slope=std::vector<double>() )
{

    // Find index of upper bin boundary
    int idx1 = 0;
    int idx3 = 0;

    idx1 = find( erg1,erg2[idx2],lastidx1 );
    lastidx1 = idx1;
    idx3 = find( erg3,erg2[idx2],lastidx3 );
    lastidx3 = idx3;

    datapoint p1;
    datapoint p3;

    //// Starting at the found indices, compute weights for each bin
    //// moving downward until we completely encompass erg2's bin
    //// erg1's weights
    //std::vector<double> w1;
    //double ub = erg2[idx2]; // initial upper bound
```

```
//for ( int e=idx1-1;e>=0;--e )
//{
    //// if the bottom boundary of energy 2's bin exceeds it
    //if ( erg2[idx2-1] < erg1[e] )
    //{
        //// then add entire remaining bin
        ////w1.push_back( (ub-erg1[e])/(erg2[idx2]-erg2[idx2-1]) );
        ////w1.push_back( (ub-erg1[e])/(erg1[e+1]-erg1[e]) );
        //w1.push_back( ub-erg1[e] ); // width of energy bin
        //ub = erg1[e];
    //}
    //// if the bottom boundary of energy 2's bin is contained
    //else /*if ( erg2[idx2-1] > erg1[e] )*/
    //{
        ////w1.push_back((ub-erg2[idx2-1])/(erg2[idx2]-erg2[idx2-1]));
        ////w1.push_back((ub-erg2[idx2-1])/(erg1[e+1]-erg1[e]));
        //w1.push_back( ub-erg2[idx2-1] );
        //break;
    //}
//}

//// erg3's weights
//std::vector<double> w3;
//ub = erg2[idx2]; // initial upper bound
//for ( int e=idx3-1;e>=0;--e )
//{
    //// if the bottom boundary of energy 2's bin exceeds it
    //if ( erg2[idx2-1] < erg3[e] )
    //{
        ////w3.push_back( (ub-erg3[e])/(erg2[idx2]-erg2[idx2-1]) );
        ////w3.push_back( (ub-erg3[e])/(erg3[e+1]-erg3[e]) );
        //w3.push_back( ub-erg3[e] );
        //ub = erg3[e];
    //}
    //// if the bottom boundary of energy 2's bin is contained
    //else /*if ( erg2[idx2-1] > erg3[e] )*/
    //{
        ////w3.push_back((ub-erg2[idx2-1])/(erg2[idx2]-erg2[idx2-1]));
        ////w3.push_back((ub-erg2[idx2-1])/(erg3[e+1]-erg3[e]));
        //w3.push_back( ub-erg2[idx2-1] );
        //break;
    //}
//}

//// compute bin-averaged values for tal1,tal3 based on weights
//for ( unsigned int w=0;w<w1.size();++w )
//{
    //p1 = p1 + tal1[idx1-w]*w1[w];
//}
//for ( unsigned int w=0;w<w3.size();++w )
//{
    //p3 = p3 + tal3[idx3-w]*w3[w];
//}


// simple and fast
p1 = tal1[idx1]*( erg2[idx2]-erg2[idx2-1] );
p3 = tal3[idx3]*( erg2[idx2]-erg2[idx2-1] );

// now interpolate these based on source energies
if ( my_useSlopes )
{
    // higher order interpolation schemes
    // (don't really do anything for us)
    //datapoint result = p1 + exp(slope[1]*(my_E2p)
    //  + slope[2]*pow(my_E2p,2.0));

    //const double a = slope[0];
    //const double b = slope[1];
    //const double c = slope[2];
    //const double d = slope[3];
    //const double x = my_E2p;
    //const double dydx = (b+2*c*x+3*d*x*x)*exp(a+b*x+c*x*x+d*x*x*x);
    //const double d2ydx2 = (2*c+6*d*x)*exp(a+b*x+c*x*x+d*x*x*x)
    //  + pow((b+2*c*x+3*d*x*x),2.0)*exp(a+b*x+c*x*x+d*x*x*x);
    //const double d3ydx3 = (6*d)*exp(a+b*x+c*x*x+d*x*x*x)
```

```
        //  + (2*c+6*d*x)*(b+2*c*x+3*d*x*x)*exp(a+b*x+c*x*x+d*x*x*x)
        // + 2*(b+2*c*x+3*d*x*x)*(2*c+6*d*x)*exp(a+b*x+c*x*x+d*x*x*x)
        //  + pow((b+2*c*x+3*d*x*x),3.0)*exp(a+b*x+c*x*x+d*x*x*x);
        //datapoint result = p1 + x*dydx+x*x*d2ydx2/2.0+x*x*x*d3ydx3/6.0;

        const double a = slope[0];
        const double b = slope[1];
        const double c = slope[2];
        const double x = my_E2p;
        const double x0 = my_E1p;
        const double dydx = (b+2*c*x0)*exp(a+b*x0+c*x0*x0);
        const double d2ydx2 = (2*c)*exp(a+b*x0+c*x0*x0)
            + pow((b+2*c*x0),2.0)*exp(a+b*x0+c*x0*x0);
        datapoint result = p1 + (x-x0)*dydx + pow(x-x0,2.0)*d2ydx2/2.0;

        //datapoint result = p1*exp(slope[1]*(my_E2p-my_E1p));
        if ( isnan(result.get()) || isinf(result.get()) )
        {
            return datapoint(0.0,0.0);
        }
        return result;

        //return logInterpolate2 ( my_E2p,my_E1p,p1,slope );
        //return logInterpolate( p1,p3,ergdist,slope );
    }
    //datapoint result = logInterpolate( p1,p3,ergdist );
    datapoint result = linearInterpolate( p1,p3,my_ergDist );

    return result;
}




std::vector< datapoint > taginterpolator::interpolate(
    const std::vector< double >& newerg,
    std::tr1::shared_ptr<ptally> _tal1_,
    std::tr1::shared_ptr<ptally> _tal3_ )
{
    // Dynamic cast tallies to tallytag class
    std::tr1::shared_ptr<tallytag> tal1
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_ );
    std::tr1::shared_ptr<tallytag> tal3
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal3_ );
    if ( ! tal1 || ! tal3 )
    {
        throw fatal_error("could not dynamic cast ptally to tallytag");
    }
    //timer init;
    //init.start( );

    // peak 1's index in energy vector 1
    const int E1p1idx = tal1->srcergidx;
    // peak 2's index in energy vector 1
    const int E1p2idx = find(tal1->erg(),my_E2p);
    // peak 2's index in energy vector 2
    const int E2p2idx = find(newerg,my_E2p);
    // peak 3's index in energy vector 3
    const int E3p3idx = tal3->srcergidx;
    // peak 2's index in energy vector 3
    const int E3p2idx = find(tal3->erg(),my_E2p);

    // Extract energy and tally vectors
    //
    my_E1.assign( tal1->erg().begin(),
        tal1->erg().begin()+E1p1idx+1 );
    my_T1_uncoll = tal1->talUncl(E1p1idx);
    my_T1_brem.assign( tal1->talBrem().begin(),
        tal1->talBrem().begin()+E1p1idx+1 );
    my_T1_xray.assign( tal1->talXray().begin(),
        tal1->talXray().begin()+E1p2idx+1 );
    my_T1_annih.assign( tal1->talAnnh().begin(),
        tal1->talAnnh().begin()+E1p2idx+1 );
    my_T1_scat.assign( tal1->talScat().begin(),
        tal1->talScat().begin()+E1p1idx+1 );
```

```
my_E2.assign( newerg.begin(),newerg.begin()+E2p2idx+1 );

my_E3.assign( tal3->erg().begin(),
    tal3->erg().begin()+E3p3idx+1 );
my_T3_uncoll = tal3->talUncl(E3p3idx);
my_T3_brem.assign( tal3->talBrem().begin(),
    tal3->talBrem().begin()+E3p3idx+1 );
my_T3_xray.assign( tal3->talXray().begin(),
    tal3->talXray().begin()+E3p2idx+1 );
my_T3_annih.assign( tal3->talAnnh().begin(),
    tal3->talAnnh().begin()+E3p2idx+1 );
my_T3_scat.assign( tal3->talScat().begin(),
    tal3->talScat().begin()+E3p3idx+1 );

//init.stop( );
//global::init += init.getTime();

// Get compton edges
//
const double comperg1 = tal1->comperg;
const double comperg2 = phys::scaterg( my_E2p,phys::pi );
const double comperg3 = tal3->comperg;
const int compidx1 = tal1->compidx;
//const int compidx2 = find( E2,comperg2 ); // not used right now
const int compidx3 = tal3->compidx;

//timer scat;
//scat.start();
// Linearly transform scattering energy
// above and below compton edge
//
my_E1_scat = transformscat( my_E1,comperg1,compidx1,
    my_E1p,E1p1idx,comperg2,my_E2p );
//my_E1_scat = my_E1;
my_E3_scat = transformscat( my_E3,comperg3,compidx3,
    my_E3p,E3p3idx,comperg2,my_E2p );
//my_E3_scat = my_E3;
//scat.stop();
//global::scat += scat.getTime();

//timer brem;
//brem.start();
// Linearly transform bremsstrahlung
//
my_E1_brem = lintran( my_E1,my_E1[0],my_E1p,my_E2[0],my_E2p );
my_E3_brem = lintran( my_E3,my_E3[0],my_E3p,my_E2[0],my_E2p );
//E1_brem = transformscat( E1,comperg1,compidx1,
//  E1p,E1p1idx,comperg2,E2p );
//E3_brem = transformscat( E3,comperg3,compidx3,
//  E3p,E3p3idx,comperg2,E2p );
//brem.stop();
//global::brem += brem.getTime();

//timer intrp;
//intrp.start();
// Interpolate scattering component at E2 energies
//
my_T2_uncoll.resize( my_E2.size() );
my_T2_brem.resize( my_E2.size() );
my_T2_xray.resize( my_E2.size() );
my_T2_annih.resize( my_E2.size() );
my_T2_scat.resize( my_E2.size() );



//timer intrpscat;
//intrpscat.start();
resetIndices();
for ( int e=1;e<E2p2idx+1;++e )
{
    my_T2_scat[e] = my_T2_scat[e] + interpolateBins(
        my_E1_scat,my_T1_scat,my_E3_scat,
        my_T3_scat,my_E2,e,my_scatSlope );
}
//intrpscat.stop();
```

```
    //global::intrpscat += intrpscat.getTime();

    // Interpolate bremsstrahlung component
    //timer intrpbrem;
    //intrpbrem.start();
    resetIndices();
    for ( int e=1;e<E2p2idx+1;++e )
    {
        my_T2_brem[e] = my_T2_brem[e] + interpolateBins(
            my_E1_brem,my_T1_brem,my_E3_brem,
            my_T3_brem,my_E2,e,my_bremSlope );
    }
    //intrpbrem.stop();
    //global::intrpbrem += intrpbrem.getTime();

    // Add discrete peaks, using unmodified energy spectrum
    //timer intrpxray;
    //intrpxray.start();
    resetIndices();
    //for ( int e=1;e<E2p2idx+1;++e )
    //{
        //my_T2_xray[e] = my_T2_xray[e]
        //  + interpolateDiscreteBins( my_E1,my_T1_xray,
        //  my_E3,my_T3_xray,my_E2,e,my_xraySlope );
    //}
    for ( int e=1;e<E2p2idx+1;++e )
    {
        //if ( my_T1_xray[e] > 1e-20 && my_T3_xray[e] > 1e-20 )
        //{
            my_T2_xray[e] = my_T2_xray[e]
                + interpolateDiscreteBins( my_E1,my_T1_xray,
                my_E3,my_T3_xray,my_E2,e,my_xraySlope );
        //}
    }
    //intrpxray.stop();
    //global::intrpxray += intrpxray.getTime();

    //timer intrpannh;
    //intrpannh.start();
    resetIndices();
    if ( my_E2p > 2*phys::me )
    {
        for ( int e=1;e<E2p2idx+1;++e )
        {
            my_T2_annih[e] = my_T2_annih[e]
                + interpolateDiscreteBins( my_E1,my_T1_annih,
                my_E3,my_T3_annih,my_E2,e,my_annhSlope );
        }
    }
    //intrpannh.stop();
    //global::intrpannh += intrpannh.getTime();

    // Interpolate uncollided point
    //timer intrpuncl;
    //intrpuncl.start();
    datapoint p1 = my_T1_uncoll*( my_E1[E1p1idx] - my_E1[E1p1idx-1] );
    datapoint p3 = my_T3_uncoll*( my_E3[E3p3idx] - my_E3[E3p3idx-1] );
    my_T2_uncoll[E2p2idx] = my_T2_uncoll[E2p2idx]
        + logInterpolate( p1,p3,my_ergDist );
    //intrpuncl.stop();
    //global::intrpuncl += intrpuncl.getTime();


    // Sum components
    std::vector<datapoint> result( my_E2.size() );
    for ( int e=1;e<E2p2idx+1;++e )
    {
        result[e] = my_T2_uncoll[e] + my_T2_brem[e] + my_T2_xray[e]
            + my_T2_annih[e] + my_T2_scat[e];
    }

    //intrp.stop();
    //global::intrp += intrp.getTime();
    //std::cout <<"done interpolating" <<std::endl;
    return result;
}
```

```
std::vector< datapoint > interpolator::interpolate(
    const std::vector< double >& newerg,
    std::tr1::shared_ptr<ptally> _tal1_,
    std::tr1::shared_ptr<ptally> _tal3_ ,
    std::vector< double > peak )
{
    // Dynamic cast tallies to tally class
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal3
    = std::tr1::dynamic_pointer_cast<tally>( _tal3_ );
    if ( ! tal1 || ! tal3 )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tally");
    }

    // peak 1's index in energy vector 1
    const int E1p1idx = tal1->srcergidx;
    // peak 2's index in energy vector 1
    //const int E1p2idx = find(tal1->erg(),E2p);
    // peak 2's index in energy vector 2
    const int E2p2idx = find(newerg,my_E2p);
    // peak 3's index in energy vector 3
    const int E3p3idx = tal3->srcergidx;
    // peak 2's index in energy vector 3
    //const int E3p2idx = find(tal3->erg(),E2p);

    // Extract energy and tally vectors
    //
    my_E1.assign( tal1->erg().begin(),
        tal1->erg().begin()+E1p1idx+1 );
    my_T1.assign( tal1->tal().begin(),
        tal1->tal().begin()+E1p1idx+1 );
    my_E2.assign( newerg.begin(),
        newerg.begin()+E2p2idx+1 );
    my_E3.assign( tal3->erg().begin(),
        tal3->erg().begin()+E3p3idx+1 );
    my_T3.assign( tal3->tal().begin(),
        tal3->tal().begin()+E3p3idx+1 );

    // Add any additional peaks
    // check for annihilation peak
    if ( my_E1p >= 2*phys::me && my_E3p >= 2*phys::me )
    {
        peak.push_back( phys::me );
    }
    // Get Peaks Indices, these should be sorted by energy
    // the +1 is for the source peak
    const int numpeak
        = static_cast<int>( peak.size() )+1;
    std::vector< int > peakidx1( numpeak );
    std::vector< int > peakidx2( numpeak );
    std::vector< int > peakidx3( numpeak );
    for ( int p=0;p<numpeak-1;++p )
    {
        peakidx1[p] = find( my_E1,peak[p] );
        peakidx2[p] = find( my_E2,peak[p] );
        peakidx3[p] = find( my_E3,peak[p] );
    }
    // Last Peak should always be the energy's source peak
    peakidx1[numpeak-1] = E1p1idx;
    peakidx2[numpeak-1] = E2p2idx;
    peakidx3[numpeak-1] = E3p3idx;

    // Extract scattering energy vectors and tally vectors
    // We want to avoid all peaks
    my_E1_cont.reserve( E1p1idx-numpeak+1 );
    my_T1_cont.reserve( E1p1idx-numpeak+1 );
    my_E3_cont.reserve( E3p3idx-numpeak+1 );
    my_T3_cont.reserve( E3p3idx-numpeak+1 );

    for ( int p=0;p<numpeak;++p )
    {
```

```
    // determine first and last bins between peaks to extract
    int first1,last1,first3,last3;

    // get first point
    if ( p==0 ) // then we are on the first peak
    {
        // set the first indices to zero
        first1 = 0;
        first3 = 0;
    }
    else
    {
        // set the first bin to be the bin
        // just after the last peak
        first1 = peakidx1[p-1]+1;
        first3 = peakidx3[p-1]+1;
    }

    // get last point
    if ( p == numpeak-1 )
    {
        // then we are on the source energy peak
        // set the last indices to right below
        // the source energy peak
        last1 = peakidx1[numpeak-1]-1;
        last3 = peakidx3[numpeak-1]-1;
    }
    else
    {
        // set the last index to be just before next peak
        last1 = peakidx1[p]-1;
        last3 = peakidx3[p]-1;
    }

    for ( int e=first1;e<=last1;++e )
    {
        my_E1_cont.push_back( tal1->erg(e) );
        my_T1_cont.push_back( tal1->tal(e) );
    }

    for ( int e=first3;e<=last3;++e )
    {
        my_E3_cont.push_back( tal3->erg(e) );
        my_T3_cont.push_back( tal3->tal(e) );
    }

}

// Get compton edges
//
const double comperg1 = tal1->comperg;
const double comperg2 = phys::scaterg( my_E2p,phys::pi );
const double comperg3 = tal3->comperg;
const int compidx1 = tal1->compidx;
//const int compidx2 = find( E2,comperg2 ); // not used right now
const int compidx3 = tal3->compidx;

// Linearly transform scattering energy above and below compton edge
//
my_E1_cont = transformscat( my_E1_cont,comperg1,compidx1,
    my_E1p,E1p1idx,comperg2,my_E2p );

my_E3_cont = transformscat( my_E3_cont,comperg3,compidx3,
    my_E3p,E3p3idx,comperg2,my_E2p );



// Interpolate scattering component at E2 energies
my_T2_cont.resize( my_E2.size() );
resetIndices();
for ( int e=1;e<E2p2idx+1;++e )
{
    my_T2_cont[e] = my_T2_cont[e] + interpolateBins(
        my_E1_cont,my_T1_cont,my_E3_cont,my_T3_cont,my_E2,e );
}
```

```
    // Add discrete peaks
    my_T2_peak.resize( my_E2.size() );
    const double eps = 1e-50;
    for ( int p=0;p<numpeak;++p )
    {
        const int idx1 = peakidx1[p];
        const int idx2 = peakidx2[p];
        const int idx3 = peakidx3[p];

        if ( my_T1[idx1].get() > eps
            && my_T3[idx3].get() > eps )
        {
            datapoint p1 = my_T1[idx1]*( my_E1[idx1]-my_E1[idx1-1] );
            datapoint p3 = my_T3[idx3]*( my_E3[idx3]-my_E3[idx3-1] );
            my_T2_peak[idx2] = my_T2_peak[idx2]
                + logInterpolate( p1,p3,my_ergDist );
        }
    }

    std::vector< datapoint > result( my_E2.size() );
    for ( int e=1;e<E2p2idx+1;++e )
    {
        result[e] = my_T2_cont[e] + my_T2_peak[e];
    }

    return result;
}




std::vector< datapoint > detinterpolator::interpolate(
    const std::vector< double >& newerg,
    std::tr1::shared_ptr<ptally> _tal1_,
    std::tr1::shared_ptr<ptally> _tal3_,
    std::vector< double > peak  )
{
    // Dynamic cast tallies to tally class
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal3
        = std::tr1::dynamic_pointer_cast<tally>( _tal3_ );
    if ( ! tal1 || ! tal3 )
    {
        throw fatal_error("could not dynamic cast ptally to tally");
    }

    // peak 1's index in energy vector 1
    const int E1p1idx = tal1->srcergidx;
    // peak 2's index in energy vector 2
    const int E2p2idx = find(newerg,my_E2p);
    // peak 3's index in energy vector 3
    const int E3p3idx = tal3->srcergidx;

    // Extract energy and tally vectors
    //
    my_E1.assign( tal1->erg().begin(),
        tal1->erg().begin()+E1p1idx+1 );
    my_T1.assign( tal1->tal().begin(),
        tal1->tal().begin()+E1p1idx+1 );
    my_E2.assign( newerg.begin(),
        newerg.begin()+E2p2idx+1 );
    my_E3.assign( tal3->erg().begin(),
        tal3->erg().begin()+E3p3idx+1 );
    my_T3.assign( tal3->tal().begin(),
        tal3->tal().begin()+E3p3idx+1 );


    // Add any additional peaks
    // Get Peaks Indices, these should be sorted by energy
    // the +1 is for the source peak
    int numpeak = static_cast<int>( peak.size() )+1;
    // Check if greater than 1.022 mev,
    // then add escape and double escape peaks
    if ( my_E1p >= 2*phys::me && my_E3p >= 2*phys::me )
```

```
{
    numpeak+=2;
}
std::vector< int > peakidx1( numpeak );
std::vector< int > peakidx2( numpeak );
std::vector< int > peakidx3( numpeak );
for ( unsigned int p=0;p<peak.size();++p )
{
    peakidx1[p] = find( my_E1,peak[p] );
    peakidx2[p] = find( my_E2,peak[p] );
    peakidx3[p] = find( my_E3,peak[p] );
}
if ( my_E1p >= 2*phys::me && my_E3p >= 2*phys::me )
{
    peakidx1[peak.size()]   = find( my_E1,my_E1p - 2*phys::me );
    peakidx1[peak.size()+1] = find( my_E1,my_E1p - phys::me );
    peakidx2[peak.size()]   = find( my_E2,my_E2p - 2*phys::me );
    peakidx2[peak.size()+1] = find( my_E2,my_E2p - phys::me );
    peakidx3[peak.size()]   = find( my_E3,my_E3p - 2*phys::me );
    peakidx3[peak.size()+1] = find( my_E3,my_E3p - phys::me );
}
// Last Peak should always be the energy's source peak
peakidx1[numpeak-1] = E1p1idx;
peakidx2[numpeak-1] = E2p2idx;
peakidx3[numpeak-1] = E3p3idx;


// Extract energy vectors and tally vectors
// We want to avoid all peaks
my_E1_cont.reserve( E1p1idx-numpeak+1 );
my_T1_cont.reserve( E1p1idx-numpeak+1 );
my_E3_cont.reserve( E3p3idx-numpeak+1 );
my_T3_cont.reserve( E3p3idx-numpeak+1 );
for ( int p=0;p<numpeak;++p )
{
    // indices of spectrum endpoints to extract
    int first1,last1,first3,last3;
    if ( p==0 ) // then we are on the first peak
    {
        // set the first indices to zero
        first1 = 0;
        first3 = 0;
    }
    else
    {
        first1 = peakidx1[p-1]+1;
        first3 = peakidx3[p-1]+1;
    }
    // then we are on the source energy peak
    if ( p == numpeak-1 )
    {
        // set the last indices to right
        // below the source energy peak
        last1 = peakidx1[numpeak-1]-1;
        last3 = peakidx3[numpeak-1]-1;
    }
    else
    {
        last1 = peakidx1[p]-1;
        last3 = peakidx3[p]-1;
    }

    for ( int e=first1;e<=last1;++e )
    {
        my_E1_cont.push_back( tal1->erg(e) );
        my_T1_cont.push_back( tal1->tal(e) );
    }
    for ( int e=first3;e<=last3;++e )
    {
        my_E3_cont.push_back( tal3->erg(e) );
        my_T3_cont.push_back( tal3->tal(e) );
    }
}

// Get compton edges
//
```

```
    //double comperg1 = my_E1p - tal1->comperg;
    double comperg1 = my_E1p - phys::scaterg( my_E1p,phys::pi );
    //std::cout << "comperg1 = " << comperg1 << std::endl;
    double comperg2 = my_E2p - phys::scaterg( my_E2p,phys::pi );
    //std::cout << "comperg2 = " << comperg2 << std::endl;
    //double comperg3 = my_E3p - tal3->comperg;
    double comperg3 = my_E3p - phys::scaterg( my_E3p,phys::pi );
    //std::cout << "comperg3 = " << comperg3 << std::endl;
    const int compidx1 = find( my_E1_cont,comperg1 );
    //const int compidx2 = find( E2,comperg2 ); // unused right now
    const int compidx3 = find( my_E3_cont,comperg3 );


    // Linearly transform scattering energy above and below compton edge
    //
    my_E1_cont = transformscat( my_E1_cont,comperg1,
        compidx1,my_E1p,E1p1idx,comperg2,my_E2p );
    my_E3_cont = transformscat( my_E3_cont,comperg3,
        compidx3,my_E3p,E3p3idx,comperg2,my_E2p );


    // Interpolate scattering component at E2 energies
    //
    my_T2_cont.resize( my_E2.size() );
    resetIndices();
    for ( int e=1;e<E2p2idx+1;++e )
    {
        my_T2_cont[e] = my_T2_cont[e] + interpolateBins(
            my_E1_cont,my_T1_cont,my_E3_cont,my_T3_cont,my_E2,e );
    }

    // Add discrete peaks
    my_T2_peak.resize( my_E2.size() );
    const double eps = 1e-50;
    for ( int p=0;p<numpeak;++p )
    {
        const int idx1 = peakidx1[p];
        const int idx2 = peakidx2[p];
        const int idx3 = peakidx3[p];

        if ( my_T1[idx1].get() > eps && my_T3[idx3].get() > eps )
        {
            datapoint p1 = my_T1[idx1]*( my_E1[idx1]-my_E1[idx1-1] );
            datapoint p3 = my_T3[idx3]*( my_E3[idx3]-my_E3[idx3-1] );
            my_T2_peak[idx2] = my_T2_peak[idx2]
                + logInterpolate( p1,p3,my_ergDist );
        }
    }

    std::vector< datapoint > result( my_E2.size() );
    for ( int e=1;e<E2p2idx+1;++e )
    {
        result[e] = my_T2_cont[e] + my_T2_peak[e];
    }


    return result;
}


// linear transformation
std::vector<double>
interpolatorbase::lintran( const std::vector< double >& vec,
    double oldfirstval,double oldlastval,
    double newfirstval,double newlastval )
{
    std::vector<double> result;
    // get beginning and ending indices of segment
    //const int first = find( vec,oldfirstval );
    //const int last = find( vec,oldlastval );

    const double A1 = oldfirstval;
    const double A2 = oldlastval;
    const double B1 = newfirstval;
    const double B2 = newlastval;
```

```
    // use function f(x) = mx + b
    // calculate slope
    const double m = (B2-B1)/(A2-A1);
    // calculate intercept
    const double b = B1-A1*m;
    // transform!!!
    //result.resize( last-first );
    result.resize(vec.size());
    //for ( int i=first;i<last;++i )
    //{
    //   result[i-first] = m*vec[i]+b;
    //}
    for ( size_t i=0;i<vec.size();++i )
    {
        result[i] = m*vec[i]+b;
    }
    return result;
}

// exponential transformation
void interpolatorbase::exptran( std::vector< double >& vec,
    double oldfirstval,double oldlastval,
    double newfirstval,double newlastval )
{
    // get beginning and ending indices of segment
    const int first = find( vec,oldfirstval );
    const int last = find( vec,oldlastval );

    const double A1 = oldfirstval;
    const double A2 = oldlastval;
    const double B1 = newfirstval;
    const double B2 = newlastval;

    // use function f(x) = b * exp(mx)
    const double m = log(B2/B1)/(A2-A1);
    const double b = B1/exp(A1*m);
    // transform!!!
    for ( int i=first;i<last;++i )
    {
        vec[i] = b*exp(m*vec[i]);
    }
}
```

Listing B.22: interpolation.hpp

```
#ifndef _interpolation_hpp_included_
#define _interpolation_hpp_included_

#include <vector>
#include "datapoint.hpp"
#include <tr1/memory>
#include "tally.hpp"

class interpolatorbase
{
    public:

    interpolatorbase( )
    {
        my_useSlopes = false;
    }

    void setSourceEnergies( double E1p,double E2p,double E3p )
    {
        my_E1p = E1p;
        my_E2p = E2p;
        my_E3p = E3p;
        my_ergDist = (my_E2p-my_E1p)/(my_E3p-my_E1p);
    }

    std::vector< double > erg( ) { return my_E2; };


    protected:

    double my_E1p;
    double my_E2p;
    double my_E3p;
    double my_ergDist;

    bool my_useSlopes;

    std::vector< double > my_E2;

    std::vector< double >
    transformscat( const std::vector< double >& E1,
        double comperg1,int compidx1,double E1p,int E1pidx,
        double comperg2,double E2p );

    datapoint interpolateDiscreteBins( const std::vector<double>& erg1,
        const std::vector<datapoint>& tal1,
        const std::vector<double>& erg3,
        const std::vector<datapoint>& tal3,
        const std::vector<double>& erg2,
        int idx2,
        std::vector<double> slope );

    datapoint interpolateBins( const std::vector<double>& erg1,
        const std::vector<datapoint>& tal1,
        const std::vector<double>& erg3,
        const std::vector<datapoint>& tal3,
        const std::vector<double>& erg2,
        int idx2,
        std::vector<double> slope );

    std::vector<double> lintran( const std::vector< double >& vec,
        double oldfirstval,double oldlastval,
        double newfirstval,double newlastval );
    void exptran( std::vector< double >& vec,double oldfirstval,
        double oldlastval,double newfirstval,double newlastval );

    void resetIndices() { lastidx1 = 1; lastidx3 = 1; };

    int find( const std::vector< double >& vec,double key,int lastidx );
    int find( const std::vector< double >& vec,double key );

    private:

    int lastidx1;
    int lastidx3;
```

```
};

class taginterpolator : public interpolatorbase
{
    public:

    std::vector< datapoint > interpolate(
        const std::vector< double >& newerg,
        std::tr1::shared_ptr<ptally> tal1,
        std::tr1::shared_ptr<ptally> tal3 );

    std::vector< datapoint > uncl( ) { return my_T2_uncoll; };
    std::vector< datapoint > brem( ) { return my_T2_brem; };
    std::vector< datapoint > xray( ) { return my_T2_xray; };
    std::vector< datapoint > annh( ) { return my_T2_annih; };
    std::vector< datapoint > scat( ) { return my_T2_scat; };

    void setSlope(std::vector<double> unclslope,
        std::vector<double> bremslope,
        std::vector<double> xrayslope,
        std::vector<double> annhslope,
        std::vector<double> scatslope )
    {
        my_unclSlope = unclslope;
        my_bremSlope = bremslope;
        my_xraySlope = xrayslope;
        my_annhSlope = annhslope;
        my_scatSlope = scatslope;
        my_useSlopes = true;
    }



    private:




    std::vector<double> my_unclSlope;
    std::vector<double> my_bremSlope;
    std::vector<double> my_xraySlope;
    std::vector<double> my_annhSlope;
    std::vector<double> my_scatSlope;

    // energy spectrum up to source energy
    std::vector< double > my_E1;
    // transformed bremsstrahlung spectrum
    std::vector< double > my_E1_brem;
    // transformed scattered spectrum
    std::vector< double > my_E1_scat;
    datapoint my_T1_uncoll; // uncollided is only in one bin
    // bremsstrahlung up to source energy
    std::vector< datapoint > my_T1_brem;
    // xray up to E2's peak (we dont transform this one)
    std::vector< datapoint > my_T1_xray;
    // annihilation up to E2's peak
    std::vector< datapoint > my_T1_annih;
    // scattering up to source energy
    std::vector< datapoint > my_T1_scat;


    std::vector< datapoint > my_T2_uncoll;
    std::vector< datapoint > my_T2_brem;
    std::vector< datapoint > my_T2_xray;
    std::vector< datapoint > my_T2_annih;
    std::vector< datapoint > my_T2_scat;

    // energy spectrum up to source energy
    std::vector< double > my_E3;
    // transformed bremsstrahlung spectrum
    std::vector< double > my_E3_brem;
    // transformed scattered spectrum
    std::vector< double > my_E3_scat;
    // uncollided is only in one bin
    datapoint my_T3_uncoll;
```

```
    // bremsstrahlung up to source energy
    std::vector< datapoint > my_T3_brem;
    // xray up to E2's peak (we dont transform this one)
    std::vector< datapoint > my_T3_xray;
    // annihilation up to E2's peak
    std::vector< datapoint > my_T3_annih;
    // scattering up to source energy
    std::vector< datapoint > my_T3_scat;

};

class interpolator : public interpolatorbase
{
    public:

    std::vector< datapoint > interpolate(
        const std::vector< double >& newerg,
        std::tr1::shared_ptr<ptally> _tal1_,
        std::tr1::shared_ptr<ptally> _tal3_ ,
        std::vector< double > peak );

    std::vector< datapoint > cont( ) { return my_T2_cont; };
    std::vector< datapoint > peak( ) { return my_T2_peak; };

    protected:

    // energy spectrum up to source energy
    std::vector< double > my_E1;
    // transformed scattered spectrum
    std::vector< double > my_E1_cont;
    // scattering up to source energy
    std::vector< datapoint > my_T1_cont;
    std::vector< datapoint > my_T1;

    std::vector< datapoint > my_T2_cont;
    std::vector< datapoint > my_T2_peak;

    // energy spectrum up to source energy
    std::vector< double > my_E3;
    // transformed scattered spectrum
    std::vector< double > my_E3_cont;
    // scattering up to source energy
    std::vector< datapoint > my_T3_cont;
    std::vector< datapoint > my_T3;
};


class detinterpolator : public interpolator
{
    public:

    std::vector< datapoint > interpolate(
        const std::vector< double >& newerg,
        std::tr1::shared_ptr<ptally> _tal1_,
        std::tr1::shared_ptr<ptally> _tal3_,
        std::vector< double > peak  );

    private:

    std::vector< double > transformdetscat(
        const std::vector< double >& E1,double comperg1,
        int compidx1,double E1p,int E1pidx,double comperg2,
        double E2p );

};

#endif
```

Listing B.23: model.cpp

```
#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cmath>
#include <ctime>
#include <limits>
#include <pthread.h>
#include <string.h>

#include "alarm.hpp"
#include "extras.hpp"
#include "errh.hpp"
#include "model.hpp"
#include "fileio.hpp"
#include "diagnostic.hpp"

spectrum model::totalPSignal( int detIdx,int timeIdx )
{
    spectrum total;
    for ( int i=0;i<numSNM();++i )
    {
        total = total + S_det(i,detIdx,timeIdx);
    }
    if ( ! data.norm.empty() )
        total = total + S_det_norm(detIdx,timeIdx);
    total = total + S_det_bgsp( detIdx,timeIdx );
    return total;
}

spectrum model::totalPSignal( int detIdx,int t1,int t2 )
{

    spectrum total;
    for ( int i=0;i<numSNM();++i )
    {
        for ( int t=t1;t<t2;++t )
        {
            total = total + S_det(i,detIdx,t);
        }
    }
    if ( ! data.norm.empty() )
    {
        for ( int t=t1;t<t2;++t )
        {
            total = total + S_det_norm(detIdx,t);
        }
    }
    for ( int t=t1;t<t2;++t )
    {
        total = total + S_det_bgsp(detIdx,t);
    }
    return total;
}

spectrum model::totalPBg( int detIdx )
{
    spectrum total = S_det_bg(detIdx);
    return total;
}

spectrum model::totalPSuppBg( int detIdx,int t1,int t2 )
{
    spectrum total = S_det_bgsp(detIdx,0);
    for ( int t=t1+1;t<t2;++t )
    {
        total = total + S_det_bgsp(detIdx,t);
    }
    return total;
}
```

```
tspectrum model::totalNSignal( int detIdx,int timeIdx )
{
    tspectrum total = N_det(0,detIdx,timeIdx);
    for ( int i=1;i<numSNM();++i )
    {
        total = total + N_det(i,detIdx,timeIdx);
    }
    total = total + tspectrum(N_det_bgsp( detIdx,timeIdx ),
        N_det(0,detIdx,timeIdx).getTime());

    return total;
}

tspectrum model::totalNSignal( int detIdx,int t1,int t2 )
{
    tspectrum total = N_det(0,detIdx,t1);

    for ( int t=t1+1;t<t2;++t )
    {
        total = total + N_det(0,detIdx,t);
    }
    for ( int i=1;i<numSNM();++i )
    {
        for ( int t=t1;t<t2;++t )
        {
            total = total + N_det(i,detIdx,t);
        }
    }
    for ( int t=t1;t<t2;++t )
    {
        total = total + tspectrum(N_det_bgsp(detIdx,t),
            N_det(0,detIdx,0).getTime());
    }
    return total;
}

tspectrum model::totalNBg( int detIdx )
{
    tspectrum total(N_det_bg(detIdx),N_det(0,detIdx,0).getTime());
    return total;
}

tspectrum model::totalNSuppBg( int detIdx,int t1,int t2 )
{
    tspectrum total(N_det_bgsp(detIdx,0),
        N_det(0,detIdx,0).getTime());
    for ( int t=t1+1;t<t2;++t )
    {
        total = total + tspectrum(N_det_bgsp(detIdx,t),
            N_det(0,detIdx,0).getTime());
    }
    return total;
}



tspectrum& model::N_snm ( int snmIdx )
{
    return my_N_snm[snmIdx];
}

spectrum& model::N_snm_gam ( int snmIdx )
{
    return my_N_snm_gam[snmIdx];
}

tspectrum& model::N_shld ( int snmIdx,int shldIdx )
{
    return my_N_shld[snmIdx][shldIdx];
}

tspectrum& model::N_car ( int snmIdx,int detIdx,int timeIdx )
{
    return my_N_car[ snmIdx + numSNM()
        * ( detIdx + numNDets() *timeIdx ) ];
}
```

```
tspectrum& model::N_det ( int snmIdx,int detIdx,int timeIdx )
{
    return my_N_det[ snmIdx + numSNM()
        * ( detIdx + numNDets() *timeIdx ) ];
}

spectrum& model::N_det_bg ( int detIdx )
{
    return my_N_det_bg[ detIdx ];
}

spectrum& model::N_det_bgsp ( int detIdx,int timeIdx )
{
    return my_N_det_bgsp[ ( detIdx + numNDets() *timeIdx ) ];
}

tspectrum& model::N_src ( int snmIdx )
{
    if ( numLayer(snmIdx) > 0 )
    {
        return N_shld(snmIdx,numLayer(snmIdx)-1);
    }
    else
    {
        return N_snm(snmIdx);
    }
}

std::vector< double >& model::normPos( int timeIdx,int posIdx )
{
    return my_normPos[timeIdx][posIdx];
}

dspectrum& model::S_norm ( )
{
    return my_S_norm;
}

dspectrum& model::S_gam ( int snmIdx )
{
    return my_S_src_gam[snmIdx];
}

spectrum& model::S_brem ( int snmIdx )
{
    return my_S_src_brem[snmIdx];
}

spectrum& model::S_snm ( int snmIdx )
{
    return my_S_snm[snmIdx];
}

spectrum& model::S_shld ( int snmIdx,int layerIdx )
{
    return my_S_shld[snmIdx][layerIdx];
}

spectrum& model::S_car ( int snmIdx,int detIdx,int timeIdx )
{
    return my_S_car[ snmIdx + numSNM()
        * ( detIdx + numPDets() *timeIdx ) ];
}

spectrum& model::S_car_bg ( int detIdx,int timeIdx )
{
    return my_S_car_bg[ detIdx + numPDets() *timeIdx ];
}

spectrum& model::S_car_norm ( int detIdx,int timeIdx )
{
    return my_S_car_norm[ detIdx + numPDets() *timeIdx ];
}

spectrum& model::S_det ( int snmIdx,int detIdx,int timeIdx )
```

```
{
    return my_S_det[ snmIdx + numSNM()
        * ( detIdx + numPDets() *timeIdx ) ];
}

spectrum& model::S_det_norm ( int detIdx,int timeIdx )
{
    return my_S_det_norm[ detIdx + numPDets() *timeIdx ];
}

spectrum& model::S_det_bg ( int detIdx )
{
    return my_S_det_bg[ detIdx ];
}

spectrum& model::S_det_bgsp ( int detIdx,int timeIdx )
{
    return my_S_det_bgsp[ detIdx + numPDets() *timeIdx ];
}

spectrum& model::S_bg ( int detIdx )
{
    return my_S_bg[ detIdx ];
}

spectrum& model::S_terr ( )
{
    return my_S_terr;
}

spectrum& model::S_src ( int snmIdx )
{
    if ( numLayer(snmIdx) > 0 )
    {
        return S_shld(snmIdx,numLayer(snmIdx)-1);
    }
    else
    {
        return S_snm(snmIdx);
    }
}

void model::checkData( )
{
    // check source
    for ( int i=0;i<numSNM();++i )
    {
        if ( data.trackPhoton )
        {
            if ( data.mysnm[i].mass < 1000 )
            {
                global::warn.add("SNM photon transport \
below 1 kg is not benchmarked");
            }
        }
    }
    // check NORM
    if ( ! data.norm.empty() )
    {
        if ( data.normfrac < 0.0 || data.normfrac > 1.0 )
        {
            global::fatal.add("norm fraction is not \
between 0.0 and 1.0");
        }
        if ( ! data.hasVehicle )
        {
            global::fatal.add("cannot specify no vehicle \
with NORM present");
        }
    }


}

model::model ( const input::data& inputData )
{
```

```
    dataDump = true;
    quickRun = false;
    dataPath = "data_dump/";

    data = inputData;

    // check data for errors/warnings
    checkData( );
    // print and flush any warnings
    global::warn.flush();
    global::fatal.flush();

    // set environment varialbes
    std::string radsrchome = "RADSRC_HOME="+data.mypath.radsrchome;
    std::string radsrcdata = "RADSRC_DATA="+data.mypath.radsrcdata;
    putenv( strdupa(radsrchome.c_str()) );
    putenv( strdupa(radsrcdata.c_str()) );

    // initialize response function data
    //std::cout << "allocating memory" << std::endl;
    allocateResponseMemory ( );
    initializeTime ( );
    allocateSpectrumMemory ( );
    //std::cout << "done" << std::endl;

}

void model::allocateResponseMemory ( )
{
    //std::cout << "numdets = " << numDets() << std::endl;
    //R_src.resize( numSNM() );
    //R_snm.resize( numSNM() );
    //R_shld.resize( numSNM() );
    R_det.reserve( numPDets() );
    for ( unsigned int i=0;i<data.mypdet.size();++i )
    {
        R_det.push_back( detector(
            data.mypath.pdet+sep()+data.mypdet[i]->type,
            data.mypdet[i]->posx,
            data.mypdet[i]->posy,
            data.mypdet[i]->posz,
            data.mypdet[i]->dimx,
            data.mypdet[i]->dimy,
            data.mypdet[i]->dimz,
            data.mypdet[i]->eff,
            data.mypdet[i]->A,
            data.mypdet[i]->B,
            data.mypdet[i]->C) );
        for ( unsigned int j=0;j<data.mypdet[i]->gcalarm.size();++j )
        {
            R_det.back().addAlarm( alarmPtr( new grosscount(
                data.mypdet[i]->fanofac ) ) );
            if ( ! data.mypdet[i]->gcalarm[j].jobName.empty() )
            {
                R_det.back().lastAlarm()->setNuisance(
                    data.mypath.savedir
                    + sep() + data.mypdet[i]->gcalarm[j].jobName
                    + sep() + data.mypdet[i]->gcalarm[j].specName+"_"
                    + data.mypdet[i]->gcalarm[j].detName );
            }
        }
        for ( unsigned int j=0;j<data.mypdet[i]->ewalarm.size();++j )
        {
            if ( data.mypdet[i]->ewalarm[j].windowBins.size() == 0 )
                R_det.back().addAlarm( alarmPtr( new energywindow(
                    data.mypdet[i]->fanofac,
                    data.mypdet[i]->ewalarm[j].nwindow,
                    data.mypdet[i]->ewalarm[j].spacing ) ) );
            else
                R_det.back().addAlarm( alarmPtr( new energywindow(
                    data.mypdet[i]->fanofac,
                    data.mypdet[i]->ewalarm[j].windowBins ) ) );
            if ( ! data.mypdet[i]->ewalarm[j].jobName.empty() )
            {
                R_det.back().lastAlarm()->setNuisance(
                    data.mypath.savedir
```

```
                + sep() + data.mypdet[i]->ewalarm[j].jobName
                + sep() + data.mypdet[i]->ewalarm[j].specName+"_"
                + data.mypdet[i]->ewalarm[j].detName );
        }

    }
    for ( unsigned int j=0;j<data.mypdet[i]->tempalarm.size();++j )
    {
        R_det.back().addAlarm( alarmPtr( new templatematch(
            data.mypdet[i]->fanofac,
            data.mypdet[i]->A,
            data.mypdet[i]->B,
            data.mypdet[i]->C,
            data.mypath.alarmtemp+sep(),
            data.mypdet[i]->tempalarm[j].tempname ) ) );
        if ( ! data.mypdet[i]->tempalarm[j].jobName.empty() )
        {
            R_det.back().lastAlarm()->setNuisance(
                data.mypath.savedir
                + sep() + data.mypdet[i]->tempalarm[j].jobName
                + sep() + data.mypdet[i]->tempalarm[j].specName+"_"
                + data.mypdet[i]->tempalarm[j].detName );
        }
    }
}


K_det.reserve( numNDets() );
for ( unsigned int i=0;i<data.myhedet.size();++i )
{
    K_det.push_back(  ndetPtr( new hedetector(
        data.mypath.ndet+sep()+data.myhedet[i].type,
        data.myhedet[i].posx,
        data.myhedet[i].posy,
        data.myhedet[i].posz,
        data.myhedet[i].height,
        data.myhedet[i].modrad,
        data.myhedet[i].refrad,
        data.myhedet[i].eff) )  );

    for ( unsigned int j=0;j<data.myhedet[i].gcalarm.size();++j )
    {
        K_det.back()->addAlarm( alarmPtr( new grosscount(
            data.myhedet[i].fanofac ) ) );
    }

    K_det_bg.push_back( ndetBgPtr( new hedetectorbg(
        data.mypath.ndet+sep()+data.myhedet[i].type,
        data.myhedet[i].height,
        data.myhedet[i].modrad,
        data.myhedet[i].refrad,
        data.myhedet[i].eff) )  );
}
for ( unsigned int i=0;i<data.myssdet.size();++i )
{
    K_det.push_back(  ndetPtr( new ssdetector(
        data.mypath.ndet+sep()+data.myssdet[i].type,
        data.myssdet[i].posx,
        data.myssdet[i].posy,
        data.myssdet[i].posz,
        data.myssdet[i].modt,
        data.myssdet[i].area,
        data.myssdet[i].eff) ) );

    for ( unsigned int j=0;j<data.myssdet[i].gcalarm.size();++j )
    {
        K_det.back()->addAlarm( alarmPtr( new grosscount(
            data.myssdet[i].fanofac ) ) );
    }

    K_det_bg.push_back(  ndetBgPtr( new ssdetectorbg(
        data.mypath.ndet+sep()+data.myssdet[i].type,
        data.myssdet[i].modt,
        data.myssdet[i].area,
        data.myssdet[i].eff) )  );
}
```

```
}

void model::allocateSpectrumMemory ( )
{
    //my_S_read.resize( numReadSrc() );
    //for ( int i=0;i<numReadSrc();++i )
    //{
        //my_S_read.read( data.myreadsrc[i].filename );
    //}


    my_S_src_gam.resize( numSNM() );
    my_S_src_brem.resize( numSNM() );
    my_S_snm.resize( numSNM() );
    my_S_shld.resize( numSNM() );
    for ( int i=0;i<numSNM();++i )
    {
        my_S_shld[i].resize(numLayer(i));
    }
    my_S_car.resize( numSNM() *numPDets() *numTime() );
    my_S_car_bg.resize( numPDets() *numTime() );
    my_S_car_norm.resize( numPDets() *numTime() );
    my_S_det.resize( numSNM() *numPDets() *numTime() );
    my_S_det_norm.resize( numPDets() *numTime() );
    my_S_det_bg.resize( numPDets() );
    my_S_det_bgsp.resize( numPDets() *numTime() );
    my_S_bg.resize( numPDets() );

    my_N_snm.resize( numSNM() );
    my_N_snm_gam.resize( numSNM() );
    my_N_shld.resize( numSNM() );
    for ( int i=0;i<numSNM();++i )
    {
        my_N_shld[i].resize(numLayer(i));
    }
    my_N_car.resize( numSNM()*numNDets()*numTime() );
    my_N_det.resize( numSNM()*numNDets()*numTime() );
    my_N_det_bg.resize( numNDets() );
    my_N_det_bgsp.resize( numNDets()*numTime() );
}





void model::initializeTime ( )
{
    // calculate length of vehicle
    const double length = data.mycargo.vehicley1
        -data.mycargo.vehicley0;
    // convert velocity to cm/s
    const double velocity = data.mycargo.velocity*1000*100/3600;
    int numtime = 1;

    if ( data.macroTime )
    {
        // calculate total time truck transits RPM in seconds
        const double totalTime = length/velocity;
        // calculate time intervals and distance intervals
        const double dt = data.interval;
        numtime = ceil ( totalTime/dt );
        time.resize ( numtime );
        for ( int i=0;i<numtime;++i )
        {
            time[i] = i*dt;
        }
    }
    else
    {
        // just one time at the center of the detector array
        // get average source position
        double avgsrc = 0.0;
        for ( int i=0;i<numSNM();++i )
        {
```

```
            avgsrc += data.mysnm[i].posy;
        }
        if ( numSNM() > 0 )
            avgsrc /= numSNM();
        // if there's no SNM, it will just be cargo center

        // calculate distance between front of vehicle and source
        const double srcdist = data.mycargo.vehicley1-avgsrc;

        // calculate time it takes snm to get in front of detector
        numtime = 1;
        time.resize ( numtime );
        time[0] = srcdist/velocity;
    }


    y.resize( numtime );
    for ( int i=0;i<numtime;++i )
    {
        y[i] = time[i]*velocity;
    }
    // calculate detector position in time
    pDetPosY.resize ( numPDets() );
    for ( int i=0;i<numPDets();++i )
    {
        pDetPosY[i].resize ( numtime );
        for ( int j=0;j<numtime;++j )
        {
            pDetPosY[i][j] = data.mycargo.vehicley1
                - y[j] + R_det[i].getY0();
        }
    }
    nDetPosY.resize ( numNDets() );
    for ( int i=0;i<numNDets();++i )
    {
        nDetPosY[i].resize ( numtime );
        for ( int j=0;j<numtime;++j )
        {
            nDetPosY[i][j] = data.mycargo.vehicley1
                - y[j] + K_det[i]->getY0();
        }
    }

    return;
}




void model::buildNeutronSNM ( int snmSphereNum )
{
    const int i = snmSphereNum;

    K_snm.initialize ( data.mypath.nsnm + sep()
        + data.mysnm[i].type,data.mypath.sfa );

    K_snm.buildResponse ( data.mysnm[i].mass,
                          data.mysnm[i].isoVector,
                          data.mysnm[i].fraction );

    return;
}

void model::buildNeutronSNMOut ( int snmSphereNum )
{
    const int i = snmSphereNum;

    K_snmout.initialize ( data.mypath.nsnm + sep()
        + "r" + data.mysnm[i].type );

    K_snmout.buildResponse ( data.mysnm[i].mass,
                          data.mysnm[i].isoVector,
                          data.mysnm[i].fraction );

    return;
```

```
}

void model::buildNeutronShield( int snmSphereNum,int layerNum )
{
    const int i = snmSphereNum;
    const int j = layerNum;

    K_shld.initialize( data.mypath.nshield + sep()
        + data.mysnm[i].myshield[j].type + sep() );

    double innerRadius = K_snm.getRadius();

    K_shld.buildResponse ( innerRadius,
        data.mysnm[i].myshield[j].thickness );
}

void model::buildNeutronShieldIn( int snmSphereNum,int layerNum )
{
    const int i = snmSphereNum;
    const int j = layerNum;

    K_shldin.initialize( data.mypath.nshield + sep()
        + "r" + data.mysnm[i].myshield[j].type + sep() );

    double innerRadius = K_snm.getRadius();

    K_shldin.buildResponse ( innerRadius,
        data.mysnm[i].myshield[j].thickness );

}

void model::buildNeutronShieldOut( int snmSphereNum,int layerNum )
{
    const int i = snmSphereNum;
    const int j = layerNum;

    K_shldout.initialize( data.mypath.nshield + sep()
        + "r" + data.mysnm[i].myshield[j].type + sep() );

    double innerRadius = K_snm.getRadius();

    K_shldout.buildResponse ( innerRadius,
        data.mysnm[i].myshield[j].thickness );

}

void model::buildNeutronCargo ( int snmIdx,int detIdx,int timeIdx )
{
    const int i = snmIdx;
    const int j = timeIdx;
    const int k = detIdx;

    if ( ! data.hasVehicle )
    {
        //std::cout << "about to build no vehicle" << std::endl;
        K_car.buildNoVehicle( data.mysnm[i].posx,
            data.mysnm[i].posy,
            data.mysnm[i].posz,
            K_det[k]->getX0(),
            nDetPosY[k][j],
            K_det[k]->getZ0(),
            N_src(i) );
        return;
    }
    // if detector is on left side of truck, flip source on x-axis
    // because detectors are mapped for right-side only
    double posFlipx = data.mysnm[i].posx;
    double posFlipy = data.mysnm[i].posy;
    double posFlipz = data.mysnm[i].posz;
    if ( K_det[k]->getX0() < 0 )
    {
        posFlipx *= -1.0;
    }
    K_car.initialize ( data.mypath.ncargo + sep()
        + data.mycargo.type[0] );
```

```
    K_car.buildResponse ( posFlipx ,posFlipy ,posFlipz ,
        K_det[k]->getX0(),nDetPosY[k][j],K_det[k]->getZ0() );
    K_car.scaleBy( data.mycargo.typeFrac[0] );
    const int nType = static_cast<int>( data.mycargo.type.size() );

    for ( int n=1;n<nType;++n )
    {
        nvehicle R;
        R.initialize ( data.mypath.ncargo + sep()
            + data.mycargo.type[n] );
        R.buildResponse ( posFlipx ,posFlipy ,posFlipz ,
            K_det[k]->getX0(),nDetPosY[k][j],K_det[k]->getZ0() );
        K_car.addResponse( R.getK(),data.mycargo.typeFrac[n] );
    }
}

void model::buildNeutronDetector ( int snmIdx,int detIdx,int timeIdx )
{
    const int i = snmIdx;
    const int j = timeIdx;
    const int k = detIdx;

    K_det[k]->buildResponse ( data.mysnm[i].posx,
                              data.mysnm[i].posy,
                              data.mysnm[i].posz,
                              nDetPosY[k][j] );
}


void model::buildNeutronBackground( int detIdx,int timeIdx )
{
    const int k = detIdx;
    const int j = timeIdx;

    if ( ! data.hasVehicle )
    {
        R_nbg.initialize( data.mypath.nbg );
        R_nbg.buildSource( K_det[k]->getX0(),nDetPosY[k][j],
            K_det[k]->getZ0(),data.elevation ,data.solarMod,
            data.latitude ,data.longitude );
        return;
    }

    R_nbg.initialize( data.mypath.ncargo+sep()
        + data.mycargo.type[0] + sep()
        + "nbackground" );
    R_nbg.buildSource( K_det[k]->getX0(),nDetPosY[k][j],
        K_det[k]->getZ0(),data.elevation ,data.solarMod,
        data.latitude ,data.longitude );
    R_nbg.scaleBy( data.mycargo.typeFrac[0] );
    const int nType = static_cast<int>( data.mycargo.type.size() );
    for ( int n=1;n<nType;++n )
    {
        neutronbg R;
        R.initialize ( data.mypath.ncargo+sep()
            + data.mycargo.type[0] + sep()
            + "nbackground" );
        R.buildSource ( K_det[k]->getX0(),nDetPosY[k][j],
            K_det[k]->getZ0(),data.elevation ,data.solarMod,
            data.latitude ,data.longitude );
        R_nbg.addResponse( R,data.mycargo.typeFrac[n] );
    }

}

void model::buildNeutronBackground( int detIdx )
{
    const int k = detIdx;

    R_nbg.initialize( data.mypath.nbg );
    R_nbg.buildSource( K_det[k]->getX0(),nDetPosY[k][0],
        K_det[k]->getZ0(),data.elevation ,data.solarMod,
        data.latitude ,data.longitude );
}

void model::buildNeutronDetectorBg( int detIdx )
```

```
{
    const int k = detIdx;

    K_det_bg[k]->buildResponse( );
}


void model::buildSource ( int snmSphereNum )
{
    const int i = snmSphereNum;

    R_src.buildResponse ( data.mysnm[i].mass,
                          data.mysnm[i].isoVector,
                          data.mysnm[i].fraction,
                          data.mysnm[i].age );

    // store maximum energy for efficiency
    maxErg = R_src.getMaxErg();

    return;
}

void model::buildSNM ( int snmSphereNum )
{
    const int i = snmSphereNum;

    R_snm.initialize ( data.mypath.psnm );

    double density = 18.95;
    if ( data.mysnm[i].type.compare("heu") == 0
        || data.mysnm[i].type.compare("vheu") == 0
        || data.mysnm[i].type.compare("du") == 0 )
    {
        density = 18.95;
    }
    else if ( data.mysnm[i].type.compare("wgpu") == 0
        || data.mysnm[i].type.compare("rgpu") == 0 )
    {
        density = 15.75;
    }


    R_snm.buildResponse ( data.mysnm[i].mass,
                          density,
                          maxErg,
                          data.ergRedFact );

    my_snmRadius = R_snm.getRadius( );

    return;
}


void model::buildShield ( int snmSphereNum,int layerNum )
{
    const int i = snmSphereNum;
    const int j = layerNum;

    R_shld.initialize( data.mypath.pshield
        + sep() + data.mysnm[i].myshield[j].type );

    R_shld.setAngularDistribution( R_snm.getAngularEnergy(),
        R_snm.getAngularDistribution(R_src.getGamma(),
        R_src.getBrem()) );


    R_shld.buildResponse ( data.mysnm[i].myshield[j].thickness,
        data.mysnm[i].myshield[j].omegaStream,
        maxErg,
        data.ergRedFact,
        &R_snm );
    return;
}


void model::buildNORM( )
```

```
{
    // full cargo weight of truck trailer
    const double cargomass = 20454.55; // kg

    // compute norm spectra
    //
    R_norm.initialize( data.mypath.norm,
            data.mypath.k40norm,
            data.mypath.ra226norm,
            data.mypath.u238norm,
            data.mypath.th232norm );
    R_norm.buildResponse (  data.norm,
        cargomass,data.normfrac );

    maxErg = R_norm.getMaxErg();
    //std::cout << "maxErg = " << maxErg << std::endl;
}

void model::buildCargoNORM ( int detIdx,int timeIdx )
{
    const int j = timeIdx;
    const int k = detIdx;
    //std::cout << "path " << data.mypath.pcargo + sep()
        //+ data.mycargo.type[0] + sep() + "distributed" << std::endl;
    R_car_norm.initialize ( data.mypath.pcargo + sep()
        + data.mycargo.type[0] + sep() + "distributed" );

    R_car_norm.buildResponse ( R_det[k].getX0(),pDetPosY[k][j],
        R_det[k].getZ0(),maxErg,data.ergRedFact );
    R_car_norm.scaleBy( data.mycargo.typeFrac[0] );
    const int nType = static_cast<int>( data.mycargo.type.size() );
    for ( int n=1;n<nType;++n )
    {
        cargonorm R;
        R.initialize ( data.mypath.pcargo + sep()
            + data.mycargo.type[n] + sep() + "distributed" );
        R.buildResponse ( R_det[k].getX0(),pDetPosY[k][j],
            R_det[k].getZ0(),maxErg,data.ergRedFact );
        R_car_norm.addResponse( R.getR(),data.mycargo.typeFrac[n] );
    }
}


void model::buildCargo ( int snmIdx,int detIdx,int timeIdx )
{
    const int i = snmIdx;
    const int j = timeIdx;
    const int k = detIdx;

    if ( ! data.hasVehicle )
    {
        R_car.buildNoVehicle( data.mysnm[i].posx,
            data.mysnm[i].posy,data.mysnm[i].posz,
            R_det[k].getX0(),pDetPosY[k][j],
            R_det[k].getZ0(),S_src(i) );
        return;
    }

    // if detector is on left side of truck, flip source on x-axis
    // because importance is mapped for right-side detectors only
    double posFlipx = data.mysnm[i].posx;
    double posFlipy = data.mysnm[i].posy;
    double posFlipz = data.mysnm[i].posz;
    if ( R_det[k].getX0() < 0 )
    {
        posFlipx *= -1.0;
    }

    R_car.initialize ( data.mypath.pcargo + sep()
        + data.mycargo.type[0] );

    R_car.buildResponse ( posFlipx,posFlipy,posFlipz,
        R_det[k].getX0(),pDetPosY[k][j],R_det[k].getZ0(),
        maxErg,data.ergRedFact );
    R_car.scaleBy( data.mycargo.typeFrac[0] );
    const int nType = static_cast<int>( data.mycargo.type.size() );
```

```
    for ( int n=1;n<nType;++n )
    {
        cargo R;
        R.initialize ( data.mypath.pcargo + sep()
            + data.mycargo.type[n] );
        R.buildResponse ( posFlipx,posFlipy,posFlipz,
            R_det[k].getX0(),pDetPosY[k][j],R_det[k].getZ0(),
            maxErg,data.ergRedFact );
        R_car.addResponse( R.getR(),data.mycargo.typeFrac[n] );
    }
}

void model::buildCargo ( int detIdx,int timeIdx )
{
    const int j = timeIdx;
    const int k = detIdx;

    if ( ! data.hasVehicle )
    {
        R_car_bg.initialize ( data.mypath.pbg );
        R_car_bg.buildResponse ( R_det[k].getX0(),pDetPosY[k][j],
            R_det[k].getZ0(),maxErg,data.ergRedFact );
        return;
    }

    //std::cout << "parsing " << data.mypath.pcargo + sep()
    //  + data.mycargo.type[0] + sep() + "background" << std::endl;
    R_car_bg.initialize ( data.mypath.pcargo + sep()
        + data.mycargo.type[0] + sep() + "background" );
    R_car_bg.buildResponse ( R_det[k].getX0(),pDetPosY[k][j],
        R_det[k].getZ0(),maxErg,data.ergRedFact );
    R_car_bg.scaleBy( data.mycargo.typeFrac[0] );
    const int nType = static_cast<int>( data.mycargo.type.size() );
    for ( int n=1;n<nType;++n )
    {
        background R;
        R.initialize ( data.mypath.pcargo + sep()
            + data.mycargo.type[n] + sep() + "background" );
        R.buildResponse ( R_det[k].getX0(),pDetPosY[k][j],
            R_det[k].getZ0(),maxErg,data.ergRedFact );
        R_car_bg.addResponse( R.getR(),data.mycargo.typeFrac[n] );
    }
    return;
}


void model::buildDetector ( int snmIdx,int detIdx,int timeIdx )
{
    const int i = snmIdx;
    const int j = timeIdx;
    const int k = detIdx;

    R_det[k].buildResponse ( data.mysnm[i].posx,
                             data.mysnm[i].posy,
                             data.mysnm[i].posz,
                             pDetPosY[k][j],
                             maxErg,
                             data.ergRedFact );
    return;
}

void model::buildDetector ( int detIdx,int timeIdx )
{
    const int j = timeIdx;
    const int k = detIdx;

    R_det[k].buildResponse( pDetPosY[k][j],
                             maxErg,
                             data.ergRedFact );
    return;
}

void model::buildTerrestrial ( )
{
    R_terr.initialize ( data.mypath.uranium_soil,
                        data.mypath.uranium_concrete,
```

```
                        data.mypath.potassium_soil,
                        data.mypath.potassium_concrete,
                        data.mypath.thorium_soil,
                        data.mypath.thorium_concrete );

    R_terr.buildResponse ( data.usoil,
                           data.uconc,
                           data.ksoil,
                           data.kconc,
                           data.thsoil,
                           data.thconc );

    maxErg = R_terr.getMaxErg( );
    return;
}

void model::buildBackground ( int detIdx )
{
    const int i = detIdx;

    R_bg.initialize ( data.mypath.pbg );

    R_bg.buildResponse ( R_det[i].getX0(),
        R_det[i].getY0(),
        R_det[i].getZ0(),
        maxErg,
        data.ergRedFact );

    return;
}

void model::build ( )
{

    if ( data.trackNeutron )
    {
        int maxRefIter = 100;
        for ( int i=0;i<numSNM();++i )
        {
            buildNeutronSNM(i);
            N_snm(i) = K_snm.getNeutronSpectrum();

            if ( consoleOutput() )
            {
                std::cout << "N_snm_(bare)________________:_"
                    << std::setw(15) << N_snm(i).sum()
                    << "_n/s" << std::endl;
            }

            //writetxt(N_snm(i).getTime(),"rgputest/time.dat");
            //writetxt(N_snm(i).getErg(),"rgputest/erg.dat");
            //writetxt(N_snm(i).getData(),"rgputest/data.dat");
            //writetxt(N_snm(i).getErr(),"rgputest/err.dat");

            if ( numLayer(i) > 0 )
            {
                buildNeutronShieldIn(i,0);
                buildNeutronSNMOut(i);
                tspectrum refspec = N_snm(i);
                if ( consoleOutput() )
                {
                    std::cout << "\treflection_iteration_0:_"
                        << refspec.sum() << std::endl;
                }
                double totSum = N_snm(i).sum();

                //// Using I/(I-K) method
                //tresponse newK = K_snmout.getK()*K_shldin.getK();
                //tresponse I;
                //I.setTime( newK.getTime() );
                //I.setErg(newK.getErgIn(),newK.getErgOut());
                //I.identity();
                //tresponse sumKinf = I-newK;
                //sumKinf = sumKinf.inverse();
                //N_snm(i) = N_snm(i) + sumKinf*refspec;
```

```
for ( int r=1;r<=maxRefIter;++r )
{
    refspec = K_snmout(K_shldin(refspec));
    //refspec = newK*refspec;
    if ( consoleOutput() )
    {
        std::cout << "\treflection iteration "
            << r << ": " << refspec.sum()
            << std::endl;
    }
    N_snm(i) = N_snm(i) + refspec;
    const double refSum = refspec.sum();
    totSum += refSum;
    if ( refSum/totSum < data.refeps )
    {
        break;
    }
}

if ( consoleOutput() )
{
    std::cout << "N_snm(" << i
    << ")                    : "
    << std::setw(15) << N_snm(i).sum()
    << " n/s" << std::endl;
}

buildNeutronShield(i,0);
N_shld(i,0) = K_shld( N_snm(i) );

if ( consoleOutput() )
{
    std::cout << "N_shld(" << i
    << ",0)                  : "
    << std::setw(15) << N_shld(i,0).sum()
    << " n/s" << std::endl;
}

for ( int j=1;j<numLayer(i);++j )
{
    buildNeutronShieldIn(i,j);
    buildNeutronShieldOut(i,j-1);
    tspectrum refspec = N_shld(i,j-1);
    if ( consoleOutput() )
    {
        std::cout << "\treflection iteration 0: "
        << refspec.sum() << std::endl;
    }
    double totSum = N_shld(i,j-1).sum();
    for ( int r=1;r<=maxRefIter;++r )
    {
        refspec = K_shldout(K_shldin(refspec));
        if ( consoleOutput() )
        {
            std::cout << "\treflection iteration "
            << r << ": " << refspec.sum() << std::endl;
        }
        N_shld(i,j-1) = N_shld(i,j-1) + refspec;
        const double refSum = refspec.sum();
        if ( refSum/totSum < data.refeps )
        {
            break;
        }
        totSum += refSum;
    }
    buildNeutronShield(i,j);
    N_shld(i,j) = K_shld( N_shld(i,j-1) );

    if ( consoleOutput() )
    {
        std::cout << "N_shld(" << i << ","
        << j << ")                  : "
        << std::setw(15) << N_shld(i,j).sum()
        << " n/s" << std::endl;
    }
}
```

```
}
//writetxt(N_shld(i).getTime(),"nbench/shldtime.dat");
//writetxt(N_shld(i).getErg(),"nbench/shlderg.dat");
//writetxt(N_shld(i).getData(),"nbench/shlddata.dat");
//writetxt(N_shld(i).getErr(),"nbench/shlderr.dat");

for ( int j=0;j<numNDets();++j )
{
    for ( int k=0;k<numTime();++k )
    {
        buildNeutronCargo( i,j,k );

        N_car(i,j,k) = K_car( N_src(i) );

        if ( consoleOutput() )
        {
            std::cout << "N_car(" << i << "," << j
            << "," << k << ")                : "
            << std::setw(15) << N_car(i,j,k).sum()
            << " n/s/cm^2" << std::endl;
        }

        //writetxt(N_car(i,j,k).getTime(),
        //  "nbench/cartime.dat");
        //writetxt(N_car(i,j,k).getErg(),
        //  "nbench/carerg.dat");
        //writetxt(N_car(i,j,k).getData(),
        //  "nbench/cardata.dat");
        //writetxt(N_car(i,j,k).getErr(),
        //  "nbench/carerr.dat");

        //const double cut = 1e-6; // 1 eV
        //int ergidx = 0;
        //for ( int e=0;e<N_car(i,j,k).nErg();++e )
        //{
            //if ( N_car(i,j,k).getErg()[e] > cut )
            //{
                //ergidx = e;
                //break;
            //}
        //}
        //for ( int e=0;e<ergidx;++e )
        //{
            //for ( int t=0;t<N_car(i,j,k).nTime()-1;++t )
            //{
                //N_car(i,j,k)(e,t) = datapoint(0.0,0.0);
            //}
        //}

        buildNeutronDetector( i,j,k );

        N_det(i,j,k) = K_det[j]->K(N_car(i,j,k));

        //writetxt(N_det(i,j,k).getTime(),
        //  "nbench/detlowztime.dat");
        //writetxt(N_det(i,j,k).getErg(),
        //  "nbench/detlowzerg.dat");
        //writetxt(N_det(i,j,k).getData(),
        //  "nbench/detlowzdata.dat");
        //writetxt(N_det(i,j,k).getErr(),
        //  "nbench/detlowzerr.dat");

        if ( consoleOutput() )
        {
            std::cout << "N_det(" << i << "," << j
            << "," << k << ")                : "
            << std::setw(15) << N_det(i,j,k).sum()
            << " counts/s" << std::endl;
            //std::cout << N_det(i,j,k).sum() << std::endl;
        }



    }
}
```

```
        }

        if ( consoleOutput() )
            global::warn.flush();

        // Neutron Background
        if ( data.toggleBackground )
        {
            for ( int j=0;j<numNDets();++j )
            {
                buildNeutronDetectorBg( j );

                for ( int k=0;k<numTime();++k )
                {
                    buildNeutronBackground( j,k );

                    N_det_bgsp(j,k) = K_det_bg[j]->R( R_nbg );

                    if ( consoleOutput() )
                    {
                        std::cout << "N_det_bgsp(" << j << ","
                        << k << ")              : "
                        << std::setw(15) << N_det_bgsp(j,k).sum()
                        << " counts/s" << std::endl;
                    }

                }

                buildNeutronBackground( j );

                N_det_bg(j) = K_det_bg[j]->R( R_nbg );

                if ( consoleOutput() )
                {
                    std::cout << "N_det_bg(" << j
                    << ")                 : " << std::setw(15)
                    << N_det_bg(j).sum()
                    << " counts/s" << std::endl;
                }
            }
        }

        if ( consoleOutput() )
            global::warn.flush();

    }

    if ( data.trackPhoton )
    {

        if ( numSNM() > 0 )
        {
            for ( int i=0;i<numSNM();++i )
            {
                buildSource( i );

                S_gam( i ) = R_src.getGamma();
                S_brem( i ) = R_src.getBrem();

                if ( data.trackFissGam )
                {
                    buildNeutronSNM(i);
                    N_snm_gam(i) = K_snm.getGammaSpectrum();
                    if ( consoleOutput() )
                    {
                        std::cout << "N_snm_gam(" << i
                        << ")                 : " << std::setw(15)
                        << N_snm_gam(i).sum() << " g/s" << std::endl;
                    }
                }

                buildSNM( i );

                S_snm( i ) = R_snm( S_gam(i) )
                    + R_snm( S_brem(i) ) + R_snm( N_snm_gam(i) );
```

```
R_snm.clear();

if ( consoleOutput() )
{
    std::cout << "S_snm(" << i
    << ")                    : "
    << std::setw(15) << S_snm(i).sum()
    << " g/s" << std::endl;
}

if ( numLayer(i) > 0 )
{
    buildShield( i,0 );
    S_shld(i,0) = R_shld( S_snm(i) );

    if ( consoleOutput() )
    {
        std::cout << "S_shld(" << i
        << ",0)                 : "
        << std::setw(15) << S_shld(i,0).sum()
        << " g/s" << std::endl;
    }

    for ( int j=1;j<numLayer(i);++j )
    {
        buildShield( i,j );
        S_shld(i,j) = R_shld( S_shld(i,j-1) );

        if ( consoleOutput() )
        {
            std::cout << "S_shld(" << i << ","
            << j << ")                 : "
            << std::setw(15) << S_shld(i,j).sum()
            << " g/s" << std::endl;
        }
    }

    R_shld.clear();
}

//S_src(i).print();

for ( int j=0;j<numPDets();++j )
{
    for ( int k=0;k<numTime();++k )
    {
        buildCargo ( i,j,k );

        S_car ( i,j,k ) = R_car( S_src(i) );

        //S_car(i,j,k).print();

        R_car.clear();

        if ( consoleOutput() )
        {
            std::cout << "S_car(" << i << "," << j
            << "," << k << ")                : "
            << std::setw(15) << S_car(i,j,k).sum()
            << " g/s/cm^2" << std::endl;
        }

        buildDetector ( i,j,k );

        S_det ( i,j,k ) = R_det[j]( S_car ( i,j,k ) );

        S_det(i,j,k) = R_det[j].GEB( S_det(i,j,k) );

        R_det[j].clear();

        if ( consoleOutput() )
        {
            std::cout << "S_det(" << i << ","
            << j << "," << k << ")                : "
            << std::setw(15) << S_det(i,j,k).sum()
```

```
                    << " counts/s" << std::endl;
                }
            }
        }
    }
}

if ( consoleOutput() )
    global::warn.flush();

// compute NORM signal
if ( ! data.norm.empty() )
{
    buildNORM( );

    S_norm( ) = R_norm( );

    if ( consoleOutput() )
    {
        std::cout << "S_norm                        : "
        << std::setw(15) << S_norm().sum()
        << " g/s" << std::endl;
    }

    for ( int i=0;i<numTime();++i )
    {
        for ( int j=0;j<numPDets();++j )
        {
            buildCargoNORM( j,i );

            S_car_norm ( j,i ) =  R_car_norm( S_norm() );

            //R_car_norm.getR().print();
            //S_car_norm(j,i).print();

            R_car_norm.clear();

            if ( consoleOutput() )
            {
                std::cout << "S_car_norm(" << j << ","
                << i << ")              : "
                << std::setw(15) << S_car_norm(j,i).sum()
                << " g/s/cm^2" << std::endl;
            }

            buildDetector ( j,i );

            S_det_norm ( j,i ) =
                R_det[j] ( S_car_norm ( j,i ) );

            R_det[j].clear();

            if ( consoleOutput() )
            {
                std::cout << "S_det_norm(" << j << ","
                << i << ")              : " << std::setw(15)
                << S_det_norm(j,i).sum() << " counts/s"
                << std::endl;
            }

        }
    }
}


if ( consoleOutput() )
    global::warn.flush();

if ( data.toggleBackground )
{
    buildTerrestrial( );

    S_terr( ) = R_terr( );

    if ( consoleOutput() )
    {
```

```
            std::cout << "S_terr                      : "
            << std::setw(15) << S_terr().sum() << " g/s"
            << std::endl;
        }

        for ( int i=0;i<numPDets();++i )
        {
            buildBackground ( i );

            S_bg ( i ) = R_bg ( S_terr( ) );

            R_bg.clear();

            if ( consoleOutput() )
            {
                std::cout << "S_bg(" << i
                << ")                     : "
                << std::setw(15) << S_bg(i).sum()
                << " g/s/cm^2" << std::endl;
            }

            //S_bg ( i ).print();

            buildDetector ( i,0 );

            S_det_bg ( i ) = R_det[i] ( S_bg ( i ) );

            S_det_bg ( i ) = R_det[i].GEB(S_det_bg(i));

            if ( consoleOutput() )
            {
                std::cout << "S_det_bg(" << i
                << ")                 : "
                << std::setw(15) << S_det_bg(i).sum()
                << " counts/s" << std::endl;
            }

            for ( int k=0;k<numTime();++k )
            {
                buildCargo( i,k );

                S_car_bg( i,k ) = R_car_bg( S_terr() );

                R_car_bg.clear();

                if ( consoleOutput() )
                {
                    std::cout << "S_car_bg(" << i << ","
                    << k << ")               : "
                    << std::setw(15) << S_car_bg(i,k).sum()
                    << " g/s/cm^2" << std::endl;
                }

                S_det_bgsp( i,k ) = R_det[i]( S_car_bg(i,k) );

                S_det_bgsp( i,k ) =
                    R_det[i].GEB( S_det_bgsp(i,k) );

                if ( consoleOutput() )
                {
                    std::cout << "S_det_bgsp(" << i << ","
                    << k << ")             : " << std::setw(15)
                    << S_det_bgsp(i,k).sum() << " counts/s"
                    << std::endl;
                }
            }

            R_det[i].clear();
        }
    }

    if ( consoleOutput() )
        global::warn.flush();

}
```

```
// apply alarm algorithms to signals
//const double FAP = 0.01;
std::vector<double> FAP;
FAP = logspace(1e-3,1.0,100);
//FAP.push_back(0.01); // 1 %
const int numFAP = static_cast<int>(FAP.size());
for ( int i=0;i<numPDets();++i )
{
    for ( int j=0;j<numTime();++j )
    {
        for ( int a=0;a<R_det[i].numAlarm();++a )
        {
            R_det[i].alarm(a)->setSignal( totalPSignal(i,0,1) );

            //if ( ! R_det[i].alarm(a)->useNuisanceFile() )
            //{
                //R_det[i].alarm(a)->setNuisance( totalPBg(i) );
            //}
            //else
            //{
                //R_det[i].alarm(a)->getNuisanceFromFile( j );
            //}
            //for ( int f=0;f<numFAP;++f )
            //{
                //std::cout << 1.0-
                //R_det[i].alarm(a)->calculateEvasionProb(1.0,FAP[f])
                //<< std::endl;
            //}

            std::cout << "USING_NORMAL_BACKGROUND_AS_NUISANCE"
                << std::endl;
            R_det[i].alarm(a)->setNuisance( totalPBg(i) );
            std::cout << 1.0-
            R_det[i].alarm(a)->calculateEvasionProb(1.0,FAP[0])
            << std::endl;
            if ( R_det[i].alarm(a)->useNuisanceFile() )
            {
                std::cout
                << "USING_REAL-TIME_BACKGROUND_AS_NUISANCE"
                << std::endl;
                R_det[i].alarm(a)->getNuisanceFromFile( j );
                std::cout << 1.0-
                R_det[i].alarm(a)->calculateEvasionProb(1.0,FAP[0])
                << std::endl;
            }
        }
    }
}


for ( int i=0;i<numNDets();++i )
{
    for ( int j=0;j<numTime();++j )
    {
        for ( int a=0;a<K_det[i]->numAlarm();++a )
        {
            //std::cout << "setting signal" << std::endl;
            K_det[i]->alarm(a)->setSignal(
                totalNSignal(i,0,1).toSpectrum() );
            //std::cout << "setting nuisance" << std::endl;
            K_det[i]->alarm(a)->setNuisance( totalNBg(i).toSpectrum() );
            for ( int f=0;f<numFAP;++f )
            {
                std::cout << 1.0-
                K_det[i]->alarm(a)->calculateEvasionProb(1.0,FAP[f])
                << std::endl;
            }
        }
    }
}


// save any data
if ( data.mysave.doSave )
{
```

```
std::cout << "saving data" << std::endl;
std::string savedir = data.mypath.savedir+sep()
    +data.mysave.jobName;
mkdir( savedir );
writeVal<int>( numTime(),savedir+sep()+"numtime.dat" );

for ( int i=0;i<numPDets();++i )
{
    for ( int j=0;j<numTime();++j )
    {
        if ( data.mysave.saveSuppBackground )
        {
            writetxt( S_det_bgsp(i,j).erg(),savedir+sep()
                +"S_det_bgsp_"+data.mypdet[i]->type+str(i)
                +"_time"+str(j)+"_energy.dat" );
            writetxt( S_det_bgsp(i,j).get(),savedir+sep()
                +"S_det_bgsp_"+data.mypdet[i]->type+str(i)
                +"_time"+str(j)+"_value.dat" );
            writetxt( S_det_bgsp(i,j).getErr(),savedir
                +sep()+"S_det_bgsp_"+data.mypdet[i]->type
                +str(i)+"_time"+str(j)+"_error.dat" );
        }
        if ( data.mysave.saveSignal )
        {
            //std::cout << "saving signal" << std::endl;
            writetxt( totalPSignal(i,j).erg(),savedir+sep()
                +"S_det_"+data.mypdet[i]->type+str(i)
                +"_time"+str(j)+"_energy.dat" );
            writetxt( totalPSignal(i,j).get(),savedir+sep()
                +"S_det_"+data.mypdet[i]->type+str(i)
                +"_time"+str(j)+"_value.dat" );
            writetxt( totalPSignal(i,j).getErr(),savedir
                +sep()+"S_det_"+data.mypdet[i]->type+str(i)
                +"_time"+str(j)+"_error.dat" );
        }
    }
    if ( data.mysave.saveBackground )
    {
        //std::cout << "saving background" << std::endl;
        writetxt( S_det_bg(i).erg(),savedir+sep()
            +"S_det_bg_"+data.mypdet[i]->type+str(i)
            +"_energy.dat" );
        writetxt( S_det_bg(i).get(),savedir+sep()
            +"S_det_bg_"+data.mypdet[i]->type+str(i)
            +"_value.dat" );
        writetxt( S_det_bg(i).getErr(),savedir+sep()
            +"S_det_bg_"+data.mypdet[i]->type+str(i)
            +"_error.dat" );
    }
}

for ( int i=0;i<numNDets();++i )
{
    for ( int j=0;j<numTime();++j )
    {
        if ( data.mysave.saveSuppBackground )
        {
            writetxt( N_det_bgsp(i,j).erg(),savedir+sep()
                +"N_det_bgsp_"+data.myndet[i]->type+str(i)
                +"_time"+str(j)+"_energy.dat" );
            writetxt( N_det_bgsp(i,j).get(),savedir+sep()
                +"N_det_bgsp_"+data.myndet[i]->type+str(i)
                +"_time"+str(j)+"_value.dat" );
            writetxt( N_det_bgsp(i,j).getErr(),savedir
                +sep()+"N_det_bgsp_"+data.myndet[i]->type
                +str(i)+"_time"+str(j)+"_error.dat" );
        }
        if ( data.mysave.saveSignal )
        {
            //std::cout << "saving signal" << std::endl;
            writetxt( totalNSignal(i,j).getErg(),savedir+sep()
                +"N_det_"+data.myndet[i]->type+str(i)
                +"_time"+str(j)+"_energy.dat" );
            writetxt( totalNSignal(i,j).getTime(),savedir+sep()
                +"N_det_"+data.myndet[i]->type+str(i)
                +"_time"+str(j)+"_time.dat" );
```

```
                    writetxt( totalNSignal(i,j).getData(),savedir+sep()
                        +"N_det_"+data.myndet[i]->type+str(i)
                        +"_time"+str(j)+"_value.dat" );
                    writetxt( totalNSignal(i,j).getErr(),savedir
                        +sep()+"N_det_"+data.myndet[i]->type+str(i)
                        +"_time"+str(j)+"_error.dat" );
                }
            }
            if ( data.mysave.saveBackground )
            {
                //std::cout << "saving background" << std::endl;
                writetxt( N_det_bg(i).erg(),savedir+sep()
                    +"N_det_bg_"+data.myndet[i]->type+str(i)
                    +"_energy.dat" );
                writetxt( N_det_bg(i).get(),savedir+sep()
                    +"N_det_bg_"+data.myndet[i]->type+str(i)
                    +"_value.dat" );
                writetxt( N_det_bg(i).getErr(),savedir+sep()
                    +"N_det_bg_"+data.myndet[i]->type+str(i)
                    +"_error.dat" );
            }
        }

    }


}
```

Listing B.24: model.hpp

```
#ifndef _model_hpp_included_
#define _model_hpp_included_
#include <cstdlib>
#include "spectrum.hpp"
#include "response.hpp"
#include "tspectrum.hpp"
#include "tresponse.hpp"
#include "detector.hpp"
#include "cargo.hpp"
#include "snm.hpp"
#include "source.hpp"
#include "normsource.hpp"
#include "shield.hpp"
#include "background.hpp"
#include "terrestrial.hpp"
#include "nsnm.hpp"
#include "nshield.hpp"
#include "nvehicle.hpp"
#include "ndetector.hpp"
#include "paths.hpp"
#include "data.hpp"
#include "alarm.hpp"

class alarma;

class model
{
    public:

    typedef std::tr1::shared_ptr<alarma> alarmPtr;
    typedef std::tr1::shared_ptr<ndetectorbase> ndetPtr;
    typedef std::tr1::shared_ptr<ndetectorbgbase> ndetBgPtr;

    int numPDets()
    {
        return ( static_cast<int>(data.mypdet.size()) );
    }
    int numNDets()
    {
        return ( static_cast<int>(data.myhedet.size())
            + static_cast<int>(data.myssdet.size()) );
    }
    int numSNM() { return static_cast<int>(data.mysnm.size()); };
    int numPSrc()
    {
        return ( static_cast<int>(data.myreadsrc.size())
            + static_cast<int>(data.mysnm.size())
            + (data.norm.empty()?0:1) );
    }
    int numTime() { return static_cast<int>(time.size()); };

    int numLayer(int i)
    {
        return static_cast<int>(data.mysnm[i].myshield.size());
    }

    void setDataDump(bool a) { dataDump = a; return; };

    void setDataPath(const std::string& a) { dataPath = a; return; };
    void setQuickRun ( bool a ) { quickRun = a; return; };


    void build (  );

    model ( const input::data& );

    spectrum totalPSignal( int detIdx,int timeIdx );
    spectrum totalPSignal( int detIdx,int t1,int t2 );
    spectrum totalPBg( int detIdx );
    spectrum totalPSuppBg( int detIdx,int t1,int t2 );

    tspectrum totalNSignal( int detIdx,int timeIdx );
    tspectrum totalNSignal( int detIdx,int t1,int t2 );
    tspectrum totalNBg( int detIdx );
    tspectrum totalNSuppBg( int detIdx,int t1,int t2 );
```

```
tspectrum& N_snm ( int );
spectrum& N_snm_gam ( int );
tspectrum& N_shld ( int,int );
tspectrum& N_car ( int,int,int );
tspectrum& N_det ( int,int,int );
spectrum& N_det_bg ( int );
spectrum& N_det_bgsp ( int,int );
tspectrum& N_src( int );

dspectrum& S_norm( );
dspectrum& S_gam ( int );
spectrum& S_brem ( int );
spectrum& S_snm ( int );
spectrum& S_shld ( int,int );
spectrum& S_car ( int,int,int );
spectrum& S_car_bg ( int,int );
spectrum& S_car_norm ( int,int );
spectrum& S_det ( int,int,int );
spectrum& S_det_norm ( int,int );
spectrum& S_det_bg ( int );
spectrum& S_det_bgsp ( int,int );
spectrum& S_bg ( int );
spectrum& S_terr (  );
spectrum& S_src( int );

void checkData();

bool consoleOutput() { return my_consoleOutput; };
void setConsoleOutput(bool a) { my_consoleOutput = a; };

private:

bool my_consoleOutput;

void allocateResponseMemory (  );
void allocateSpectrumMemory (  );
void initializeTime (  );

void buildNeutronSNM( int );
void buildNeutronSNMOut( int );
void buildNeutronShield( int,int );
void buildNeutronShieldIn( int,int );
void buildNeutronShieldOut( int,int );
void buildNeutronCargo( int,int,int );
void buildNeutronDetector( int,int,int );
void buildNeutronBackground( int detIdx,int timeIdx );
void buildNeutronDetectorBg( int detIdx );
void buildNeutronBackground( int detIdx );

void buildSource ( int );
void buildSNM ( int );
void buildShield ( int,int );
void buildCargo ( int,int,int );
void buildCargo ( int,int );
void buildDetector ( int,int,int );
void buildDetector ( int,int );
void buildBackground ( int );
void buildTerrestrial (  );
void buildNORM ( );
void buildCargoNORM ( int,int );

input::data data;



// current maximum energy of the problem
double maxErg;

// radius of snm
double my_snmRadius;


bool quickRun;
bool dataDump;
std::string dataPath;
```

```
    std::vector< std::vector< double > > pDetPosY;
    std::vector< std::vector< double > > nDetPosY;
    std::vector< double >& normPos( int,int );
    std::vector< double >& normPos( int,int,int,int );

    int deadCenterTime;

    nsnm K_snm;
    nsnmout K_snmout;
    nshield K_shld;
    nshieldrin K_shldin;
    nshieldrout K_shldout;
    nvehicle K_car;
    std::vector< ndetPtr > K_det;
    std::vector< ndetBgPtr > K_det_bg;
    neutronbg R_nbg;

    source R_src;
    snm R_snm;
    shield R_shld;
    cargo R_car;
    cargonorm R_car_norm;
    background R_car_bg;
    std::vector < detector > R_det;
    background R_bg;
    terrestrial R_terr;
    normsource R_norm;


    std::vector < tspectrum > my_N_snm;
    std::vector < spectrum > my_N_snm_gam;
    std::vector< std::vector < tspectrum > > my_N_shld;
    std::vector < tspectrum > my_N_car;
    std::vector < tspectrum > my_N_det;
    std::vector < spectrum > my_N_det_bg;
    std::vector < spectrum > my_N_det_bgsp;

    dspectrum my_S_norm;
    std::vector < spectrum > my_S_src_pgam;
    std::vector < dspectrum > my_S_src_gam;
    std::vector < spectrum > my_S_src_brem;
    std::vector < spectrum > my_S_snm;
    std::vector< std::vector < spectrum > > my_S_shld;
    std::vector < spectrum > my_S_car;
    std::vector < spectrum > my_S_car_bg;
    std::vector < spectrum > my_S_car_norm;
    std::vector < spectrum > my_S_det;
    std::vector < spectrum > my_S_det_norm;
    std::vector < spectrum > my_S_det_bg;
    std::vector < spectrum > my_S_det_bgsp;
    spectrum my_S_terr;
    std::vector < spectrum > my_S_bg;

    std::vector< std::vector< std::vector< double > > > my_normPos;

    std::vector<double> time; // time intervals
    std::vector<double> y; // cargo distance traveled in time

};

#endif
```

Listing B.25: ndetector.cpp

```
#include "ndetector.hpp"
#include "fileio.hpp"
#include "extras.hpp"

using namespace std;


void ndetectorbase::initialize ( const string& path )
{
    my_datapath = path + sep();
    readDataFile();
}

void ndetectorbase::genResponse( )
{
    const int Ein = static_cast<int>(my_srcErg.size());
    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {
        //std::cout << "erg " << e <<  std::endl;
        my_data[e] = interpolateTally( e );

    }
    assignDataToResponse( );
}

void ndetectorbase::determineDetectorPlane( )
{
    // hard-code in detector planes, should read this in from file eventually
    const double tside = 129.54;
    const double ttop = 259.08;
    const double soff = 68.56;
    my_detXPlane = tside+soff; // side of truck + standoff
    my_detZPlane = ttop+soff; // side of truck + standoff

    // find out if this detector is a "side" or "top" detector
    if ( my_detZPos0 < my_detZPlane && my_detXPos0 > tside )
    {
        my_detPosType = side;
    }
    else if ( my_detZPos0 > ttop && my_detXPos0 < my_detXPlane )
    {
        my_detPosType = top;
    }
    else
    {
        throw fatal_error("invalid detector position");
    }
}

void ndetectorbase::buildResponse ( double srcXPos,double srcYPos,
    double srcZPos,double detYPos )
{
    // store dimensions/position of detector
    my_srcXPos = srcXPos;
    my_srcYPos = srcYPos;
    my_srcZPos = srcZPos;
    my_detYPos = detYPos;

    buildResponse( );
}

tspectrum ndetectorbase::operator() ( const tspectrum& S_car )
{
    return K(S_car);
}


void hedetectorbg::initialize ( const string& path )
{
    my_datapath = path + sep();
    readDataFile();
}
```

```
hedetectorbg::hedetectorbg( const std::string& path,
    double height,double modrad,
    double refrad,double eff )
{
    initialize(path);
    my_height = height;
    my_modrad = modrad;
    my_refrad = refrad;
    my_eff = eff;
    my_herad = 2.495; // 1.96 inch diameter he-3 tube
}

void hedetectorbg::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector dimensions
    my_h = readbin( my_datapath+"height.dat" );
    my_m = readbin( my_datapath+"modrad.dat" );
    my_r = readbin( my_datapath+"refrad.dat" );
}

string hedetectorbg::getTallyEnergyPath( int ergIdx )
{
    return my_datapath + my_side + sep() + "erg" + str(ergIdx) + sep();
}

string hedetectorbg::getTallyPath( int ergIdx,
    int hidx, int midx, int ridx )
{
    return my_datapath+my_side+sep()+"erg"+str(ergIdx)+sep()
        +"height"+str(hidx)+sep()
        +"modrad"+str(midx)+sep()
        +"refrad"+str(ridx)+sep();
}

datapoint hedetectorbg::interpolateTally( int ergIdx )
{
    string talPath;

    datapoint tal000;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx-1,my_refIdx-1);
    tal000 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal100;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx-1,my_refIdx-1);
    tal100 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal010;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx,my_refIdx-1);
    tal010 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal001;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx-1,my_refIdx);
    tal001 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal110;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx,my_refIdx-1);
    tal110 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal011;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx,my_refIdx);
    tal011 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));
```

```
    datapoint tal101;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx-1,my_refIdx);
    tal101 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    datapoint tal111;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx,my_refIdx);
    tal111 = datapoint(readVal<double>( talPath+"tal.dat" ),
        readVal<double>( talPath+"err.dat" ));

    // interpolate between dimensions
    // interpolate out height dimension
    datapoint tal00 = linearInterpolate( my_h[my_heightIdx-1],
        tal000,my_h[my_heightIdx],tal001,my_height );
    datapoint tal10 = linearInterpolate( my_h[my_heightIdx-1],
        tal100,my_h[my_heightIdx],tal101,my_height );
    datapoint tal01 = linearInterpolate( my_h[my_heightIdx-1],
        tal010,my_h[my_heightIdx],tal011,my_height );
    datapoint tal11 = linearInterpolate( my_h[my_heightIdx-1],
        tal110,my_h[my_heightIdx],tal111,my_height );

    // interpolate out moderator radius
    datapoint tal0 = linearInterpolate( my_m[my_modIdx-1],
        tal00,my_m[my_modIdx],tal01,my_modrad );
    datapoint tal1 = linearInterpolate( my_m[my_modIdx-1],
        tal10,my_m[my_modIdx],tal11,my_modrad );

    // interpolate out reflector radius
    datapoint result = linearInterpolate( my_r[my_refIdx-1],
        tal0,my_r[my_refIdx],tal1,my_refrad );

    return result;
}

void hedetectorbg::buildResponse( )
{
    //cout << "building helium detector background response" << endl;

    // need to interpolate between all three dimensions
    const int H = static_cast<int>(my_h.size());
    const int M = static_cast<int>(my_m.size());
    const int R = static_cast<int>(my_r.size());
    my_heightIdx = H-1;
    my_modIdx = M-1;
    my_refIdx = R-1;
    for ( int i=0;i<H-1;++i )
    {
        if ( my_height >= my_h[i] && my_height <= my_h[i+1] )
        {
            my_heightIdx = i+1;
            break;
        }
    }

    for ( int i=0;i<M-1;++i )
    {
        if ( my_modrad >= my_m[i] && my_modrad <= my_m[i+1] )
        {
            my_modIdx = i+1;
            break;
        }
    }
    for ( int i=0;i<R-1;++i )
    {
        if ( my_refrad >= my_r[i] && my_refrad <= my_r[i+1] )
        {
            my_refIdx = i+1;
            break;
        }
    }
```

```
    my_side = "side";
    genData( );
    assignDataToResponse(my_R_side);
    my_R_side = my_R_side * (300.0*60.0*my_eff);

    my_side = "top";
    genData( );
    assignDataToResponse(my_R_top);
    my_R_top = my_R_top * (60.0*58.0*my_eff);

    my_side = "bottom";
    genData( );
    assignDataToResponse(my_R_bottom);
    my_R_bottom = my_R_bottom * (60.0*58.0*my_eff);

    my_side = "front";
    genData( );
    assignDataToResponse(my_R_front);
    my_R_front = my_R_front * (300.0*58.0*my_eff);

    my_side = "back";
    genData( );
    assignDataToResponse(my_R_back);
    my_R_back = my_R_back * (300.0*58.0*my_eff);


    //cout << "finished building helium detector background response" << endl;
}



void hedetectorbg::genData( )
{
    const int Ein = static_cast<int>(my_srcErg.size());
    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {
        my_data[e] = interpolateTally( e );
    }
}

void hedetectorbg::assignDataToResponse( response& R )
{
    const int Eout = 2;
    const int Ein = static_cast<int>(my_srcErg.size());
    std::vector<double> ergout(2);
    ergout[0] = 1e-10;
    ergout[1] = 20.0;
    R.initialize( my_srcErg,ergout );
    for ( int ei=0;ei<Ein-1;++ei )
    {
        for ( int eo=0;eo<Eout-1;++eo )
        {
            R(eo,ei) = my_data[ei];
        }
    }
}

spectrum hedetectorbg::operator() ( const neutronbg& nbg )
{
    return R(nbg);
}

spectrum hedetectorbg::R( const neutronbg& nbg )
{
    if ( nbg.getDetPos() == neutronbg::side )
    {
        return my_R_side * (nbg.getY0Current()+nbg.getY1Current())
            + my_R_bottom * nbg.getZ0Current()
            + my_R_top * nbg.getZ1Current()
            + my_R_front * nbg.getX0Current()
            + my_R_back * nbg.getX1Current();
    }
    else /* if ( nbg.getDetPos() == nbg::top ) */
    {
        return my_R_side * (nbg.getY0Current()+nbg.getY1Current())
```

```
            + my_R_bottom * nbg.getX1Current()
            + my_R_top * nbg.getX0Current()
            + my_R_front * nbg.getZ0Current()
            + my_R_back * nbg.getZ1Current();
    }
}

void ssdetectorbg::initialize ( const string& path )
{
    my_datapath = path + sep();
    readDataFile();
}

ssdetectorbg::ssdetectorbg( const std::string& path,double modt,
    double area,double eff )
{
    initialize(path);
    my_modt = modt;
    my_area = area;
    my_eff = eff;
}

void ssdetectorbg::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector dimensions
    my_m = readbin( my_datapath+"modt.dat" );
}

string ssdetectorbg::getTallyPath( int ergIdx,int midx )
{
    return my_datapath+my_side+sep()+"erg"+str(ergIdx)+sep()
        +"modt"+str(midx)+sep();
}

tally ssdetectorbg::interpolateTally( int ergIdx )
{
    string talPath;

    tally tal0;
    talPath = getTallyPath(ergIdx,my_modIdx-1);
    tal0.parse( talPath,talPath );

    tally tal1;
    talPath = getTallyPath(ergIdx,my_modIdx);
    tal1.parse( talPath,talPath );

    // interpolate out reflector radius
    tally result = tally::interpolate( my_m[my_modIdx-1],
        tal0,my_m[my_modIdx],tal1,my_modt );

    return result;
}

void ssdetectorbg::buildResponse( )
{
    //cout << "building helium detector background response" << endl;

    const int M = static_cast<int>(my_m.size());
    my_modIdx = M-1;
    for ( int i=0;i<M-1;++i )
    {
        if ( my_modt >= my_m[i] && my_modt <= my_m[i+1] )
        {
            my_modIdx = i+1;
            break;
        }
    }

    my_side = "side";
    genData( );
    assignDataToResponse(my_R_side);
    my_R_side = my_R_side * (0.0303*sqrt(my_area)*my_eff);

    my_side = "front";
```

```
    genData( );
    assignDataToResponse(my_R_front);
    my_R_front = my_R_front * (my_area*my_eff);

    my_side = "back";
    genData( );
    assignDataToResponse(my_R_back);
    my_R_back = my_R_back * (my_area*my_eff);

    //cout << "finished building helium detector background response" << endl;
}

void ssdetectorbg::genData( )
{
    const int Ein = static_cast<int>(my_srcErg.size());
    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {
        my_data[e] = interpolateTally( e );
    }
}

void ssdetectorbg::assignDataToResponse( response& R )
{
    const int Eout = my_data[0].nErg();
    const int Ein = static_cast<int>(my_srcErg.size());
    R.initialize( my_srcErg,my_data[0].erg() );
    for ( int ei=0;ei<Ein-1;++ei )
    {
        for ( int eo=0;eo<Eout-1;++eo )
        {
            R(eo,ei) = my_data[ei].tal(eo);
        }
    }
}

spectrum ssdetectorbg::operator() ( const neutronbg& nbg )
{
    return R(nbg);
}

spectrum ssdetectorbg::R( const neutronbg& nbg )
{
    if ( nbg.getDetPos() == neutronbg::side )
    {
        return my_R_side * ( nbg.getY0Current()
            + nbg.getY1Current() + nbg.getZ0Current()
            + nbg.getZ1Current() )
            + my_R_front * nbg.getX0Current()
            + my_R_back * nbg.getX1Current();
    }
    else /* if ( nbg.getDetPos() == nbg::top ) */
    {
        return my_R_side * ( nbg.getY0Current()
            + nbg.getY1Current() + nbg.getX1Current()
            + nbg.getX0Current() )
            + my_R_front * nbg.getZ0Current()
            + my_R_back * nbg.getZ1Current();
    }
}


hedetector::hedetector( const std::string& path,
    double detXPos,double detYPos,double detZPos,
    double height,double modrad,
    double refrad, double eff )
{
    initialize(path);
    my_detXPos0 = detXPos;
    my_detYPos0 = detYPos;
    my_detZPos0 = detZPos;
    my_height = height;
    my_modrad = modrad;
    my_refrad = refrad;
    my_eff = eff;
    my_herad = 2.495; // 1.96 inch diameter he-3 tube
```

```
    my_gap = 20; //- my_modrad - my_herad;
    determineDetectorPlane( );
}

void hedetector::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector dimensions
    my_h = readbin( my_datapath+"height.dat" );
    my_m = readbin( my_datapath+"modrad.dat" );
    my_r = readbin( my_datapath+"refrad.dat" );
}

string hedetector::getTallyEnergyPath( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
}

string hedetector::getTallyPath( int ergIdx,
    int hidx, int midx, int ridx )
{
    return my_datapath+"erg"+str(ergIdx)+sep()+"height"+str(hidx)
        +sep()+"modrad"+str(midx)+sep()+"refrad"+str(ridx)+sep();
}

ntally hedetector::interpolateTally( int ergIdx )
{
    string talErgPath = getTallyEnergyPath(ergIdx);
    string talPath;

    ntally tal000;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx-1,my_refIdx-1);
    tal000.parse( talErgPath,talPath );

    ntally tal100;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx-1,my_refIdx-1);
    tal100.parse( talErgPath,talPath );

    ntally tal010;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx,my_refIdx-1);
    tal010.parse( talErgPath,talPath );

    ntally tal001;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx-1,my_refIdx);
    tal001.parse( talErgPath,talPath );

    ntally tal110;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx,my_refIdx-1);
    tal110.parse( talErgPath,talPath );

    ntally tal011;
    talPath = getTallyPath(ergIdx,my_heightIdx-1,
        my_modIdx,my_refIdx);
    tal011.parse( talErgPath,talPath );

    ntally tal101;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx-1,my_refIdx);
    tal101.parse( talErgPath,talPath );

    ntally tal111;
    talPath = getTallyPath(ergIdx,my_heightIdx,
        my_modIdx,my_refIdx);
    tal111.parse( talErgPath,talPath );

    // interpolate between dimensions
    // interpolate out height dimension
    ntally tal00 = ntally::interpolate( my_h[my_heightIdx-1],
        tal000,my_h[my_heightIdx],tal001,my_height );
    ntally tal10 = ntally::interpolate( my_h[my_heightIdx-1],
```

```
            tal100 ,my_h[my_heightIdx],tal101,my_height );
        ntally tal01 = ntally::interpolate( my_h[my_heightIdx-1],
            tal010 ,my_h[my_heightIdx],tal011,my_height );
        ntally tal11 = ntally::interpolate( my_h[my_heightIdx-1],
            tal110 ,my_h[my_heightIdx],tal111,my_height );

        // interpolate out moderator radius
        ntally tal0 = ntally::interpolate( my_m[my_modIdx-1],
            tal00 ,my_m[my_modIdx],tal01 ,my_modrad );
        ntally tal1 = ntally::interpolate( my_m[my_modIdx-1],
            tal10 ,my_m[my_modIdx],tal11 ,my_modrad );

        // interpolate out reflector radius
        ntally result = ntally::interpolate( my_r[my_refIdx-1],
            tal0 ,my_r[my_refIdx],tal1 ,my_refrad );

        return result;
    }

    void hedetector::buildResponse( )
    {
        //cout << "building helium detector response" << endl;
        const double yMin = my_detYPos-my_herad-my_modrad-my_refrad;
        const double yMax = my_detYPos+my_herad+my_modrad+my_refrad;

        double omega =0.0;
        if ( my_detPosType == side )
        {
            const double xMin = my_detXPos;
            const double xMax = my_detXPos+2*my_modrad
                +2*my_herad+my_refrad;
            const double zMin = my_detZPos - my_height/2.0
                - my_modrad - my_refrad;
            const double zMax = my_detZPos + my_height/2.0
                + my_modrad + my_refrad;
            omega = solidAngle( my_srcXPos,my_srcYPos,my_srcZPos,
                xMin,yMax,zMax,
                xMin,yMin,zMin,
                xMin,yMax,zMin )
                +
                solidAngle(my_srcXPos,my_srcYPos,my_srcZPos,
                xMin,yMax,zMax,
                xMin,yMin,zMin,
                xMin,yMin,zMax );
            my_area = (zMax-zMin)*(yMax-yMin);
            //my_area = (zMax-zMin)*my_actwidth;
        }
        else /* if ( my_detPosType == top ) */
        {
            const double zMin = my_detXPos;
            const double zMax = my_detXPos+2*my_modrad
                +2*my_herad+my_refrad;
            const double xMin = my_detZPos - my_height/2.0
                - my_modrad - my_refrad;
            const double xMax = my_detZPos + my_height/2.0
                + my_modrad + my_refrad;
            omega = solidAngle( my_srcXPos,my_srcYPos,my_srcZPos,
                xMax,yMax,zMin,
                xMin,yMin,zMin,
                xMax,yMin,zMin )
                +
                solidAngle(my_srcXPos,my_srcYPos,my_srcZPos,
                xMax,yMax,zMin,
                xMin,yMin,zMin,
                xMin,yMax,zMin );
            my_area = (xMax-xMin)*(yMax-yMin);
            //my_area = (xMax-xMin)*my_actwidth;
        }

        // need to interpolate between all three dimensions
        const int H = static_cast<int>(my_h.size());
        const int M = static_cast<int>(my_m.size());
        const int R = static_cast<int>(my_r.size());
        my_heightIdx = H-1;
        my_modIdx = M-1;
        my_refIdx = R-1;
```

```
    for ( int i=0;i<H-1;++i )
    {
        if ( my_height >= my_h[i] && my_height <= my_h[i+1] )
        {
            my_heightIdx = i+1;
            break;
        }
    }
    for ( int i=0;i<M-1;++i )
    {
        if ( my_modrad >= my_m[i] && my_modrad <= my_m[i+1] )
        {
            my_modIdx = i+1;
            break;
        }
    }
    for ( int i=0;i<R-1;++i )
    {
        if ( my_refrad >= my_r[i] && my_refrad <= my_r[i+1] )
        {
            my_refIdx = i+1;
            break;
        }
    }

    // build response function matrix
    genResponse( );

    // multiply by area because cargo
    // spectrum is in units of per unit area
    my_K = my_K * my_area;

    //cout << "finished building helium detector response" << endl;
}

tspectrum hedetector::K( const tspectrum& S_car )
{
    // because there is a void between detector face
    // and start of particle beam in simulation
    // need to shift time to account for this distance travelled
    return my_K * shiftDistance(S_car,my_gap);
    //return my_K * S_car;
}


ssdetector::ssdetector( const std::string& path,
    double detXPos,double detYPos,double detZPos,
    double modt,double area,double eff )
{
    initialize(path);
    my_detXPos0 = detXPos;
    my_detYPos0 = detYPos;
    my_detZPos0 = detZPos;
    my_modt = modt;
    my_area = area;
    my_eff = eff;
    determineDetectorPlane( );
}

void ssdetector::readDataFile()
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector dimensions
    my_m = readbin( my_datapath+"modt.dat" );
}

string ssdetector::getTallyEnergyPath( int ergIdx,int midx )
{
    return getTallyPath(ergIdx,midx);
}

string ssdetector::getTallyPath( int ergIdx, int midx )
{
    return my_datapath+"erg"+str(ergIdx)+sep()+"modt"+str(midx)+sep();
}
```

```
ntally ssdetector::interpolateTally( int ergIdx )
{
    string talErgPath;
    string talPath;

    ntally tal0;
    talPath = getTallyPath(ergIdx,my_modIdx-1);
    talErgPath = getTallyEnergyPath(ergIdx,my_modIdx-1);
    tal0.parse( talErgPath,talPath );

    ntally tal1;
    talPath = getTallyPath(ergIdx,my_modIdx);
    talErgPath = getTallyEnergyPath(ergIdx,my_modIdx);
    tal1.parse( talErgPath,talPath );

    // interpolate out moderator thickness
    ntally result = ntally::interpolate( my_m[my_modIdx-1],
        tal0,my_m[my_modIdx],tal1,my_modt );

    return result;
}

void ssdetector::buildResponse ( )
{
    //cout << "building solid state detector response" << endl;
    // need to interpolate between moderator thicknesses
    const int M = static_cast<int>(my_m.size());
    my_modIdx = M-1;
    for ( int i=0;i<M-1;++i )
    {
        if ( my_modt >= my_m[i] && my_modt <= my_m[i+1] )
        {
            my_modIdx = i+1;
            break;
        }
    }

    // build response function matrix
    genResponse( );

    // multiply by equivalent area because cargo spectrum is in units of per unit area
    my_K = my_K * my_area;

    //cout << "finished building solid state detector response" << endl;
}

tspectrum ssdetector::K( const tspectrum& S_car )
{
    return my_K * S_car;
}
```

Listing B.26: ndetector.hpp

```
#ifndef _ndetector_hpp_included_
#define _ndetector_hpp_included_
#include <vector>
#include <string>
#include <tr1/memory>
#include "nsubmodel.hpp"
#include "neutronbg.hpp"


class alarma;

class ndetectorbase : public nsubmodel
{
    public:

    typedef std::tr1::shared_ptr<alarma> alarmPtr;

    enum dpos { side,top };

    double getX0( ) { return my_detXPos0; };
    double getY0( ) { return my_detYPos0; };
    double getZ0( ) { return my_detZPos0; };

    void initialize ( const std::string& path );

    void buildResponse ( double srcXPos,double srcYPos,
        double srcZPos,double detYPos );

    virtual void buildResponse( ) = 0;

    ndetectorbase ( ) { };

    tspectrum operator() ( const tspectrum& );
    virtual tspectrum K( const tspectrum& ) = 0;

    void addAlarm( alarmPtr a )
    {
        my_alarm.push_back(a);
    }
    int numAlarm() const { return static_cast<int>(my_alarm.size()); };
    alarmPtr alarm(int a) { return my_alarm[a]; };
    alarmPtr lastAlarm() { return my_alarm.back(); };

    protected:

    virtual void readDataFile() = 0;
    virtual ntally interpolateTally( int ) = 0;
    void genResponse( );
    void determineDetectorPlane( );

    // area of detector
    double my_area;
    // user specified source position
    double my_srcXPos;
    double my_srcYPos;
    double my_srcZPos;
    // user specified detector position
    double my_detXPos;
    double my_detYPos;
    double my_detZPos;
    // detector planes
    double my_detXPlane;
    double my_detZPlane;
    // original user specified detector position
    double my_detXPos0;
    double my_detYPos0;
    double my_detZPos0;
    // extra space gap in simulation
    double my_gap;


    double my_eff; // collection efficiency

    dpos my_detPosType;
```

```
    std::vector<alarmPtr> my_alarm;

};


class hedetector : public ndetectorbase
{
    public:

    void buildResponse(  );

    hedetector( const std::string&,double,double,double,
        double,double,double,double );

    hedetector( ) { };

    tspectrum K( const tspectrum& );

    protected:

    void readDataFile( );
    std::string getTallyEnergyPath( int );
    std::string getTallyPath( int,int,int,int );
    ntally interpolateTally( int );

    // tube heights simulated
    std::vector<double> my_h;
    // moderator radii simulated
    std::vector<double> my_m;
    // reflector radii simulated
    std::vector<double> my_r;

    // tube height
    double my_height;
    // moderator radius
    double my_modrad;
    // reflector radius
    double my_refrad;
    // hard coded helium tube radius
    double my_herad;


    int my_heightIdx;
    int my_modIdx;
    int my_refIdx;

};



class ndetectorbgbase
{
    public:


    virtual void buildResponse( ) = 0;

    virtual spectrum operator() ( const neutronbg& nbg ) = 0;

    virtual spectrum R( const neutronbg& nbg ) = 0;

    protected:

};


class hedetectorbg : public ndetectorbgbase
{
    public:

    void initialize ( const std::string& path );

    hedetectorbg( const std::string& path,double height,
        double modrad,double refrad,double eff );

    void buildResponse( );
```

```
    spectrum operator() ( const neutronbg& nbg );

    spectrum R( const neutronbg& nbg );


    private:

    response my_R_side;
    response my_R_bottom;
    response my_R_top;
    response my_R_front;
    response my_R_back;

    std::string my_side;

    void genData( );
    void assignDataToResponse( response& );

    std::string my_datapath;
    std::vector<double> my_srcErg;
    std::vector<datapoint> my_data;

    void readDataFile( );
    std::string getTallyEnergyPath( int );
    std::string getTallyPath( int,int,int,int );
    datapoint interpolateTally( int );

    // tube heights simulated
    std::vector<double> my_h;
    // moderator radii simulated
    std::vector<double> my_m;
    // reflector radii simulated
    std::vector<double> my_r;


    // tube height
    double my_height;
    // moderator radius
    double my_modrad;
    // reflector radius
    double my_refrad;
    // hard coded helium tube radius
    double my_herad;
    // area of detector side
    double my_area;
    // collection efficiency
    double my_eff;

    int my_heightIdx;
    int my_modIdx;
    int my_refIdx;

};



class ssdetectorbg : public ndetectorbgbase
{
    public:

    void initialize ( const std::string& path );

    ssdetectorbg( const std::string& path,
        double modt,double area,double eff );

    void buildResponse( );

    spectrum operator() ( const neutronbg& nbg );

    spectrum R( const neutronbg& nbg );


    private:

    response my_R_side;
```

```
    response my_R_front;
    response my_R_back;

    std::string my_side;

    void genData( );
    void assignDataToResponse( response& );

    std::string my_datapath;
    std::vector<double> my_srcErg;
    std::vector<tally> my_data;

    void readDataFile( );
    std::string getTallyPath( int,int );
    tally interpolateTally( int );

    // moderator thicknesses simulated
    std::vector<double> my_m;

    // index of moderator thickness
    int my_modIdx;
    // moderator thickness
    double my_modt;
    // area of detector face
    double my_area;
    // collection efficiency
    double my_eff;

};


class ssdetector : public ndetectorbase
{
    public:

    void buildResponse( );

    ssdetector( const std::string&,double,double,
        double,double,double,double );

    ssdetector( ) { };

    tspectrum K( const tspectrum& );

    protected:

    void readDataFile( );
    std::string getTallyEnergyPath( int,int );
    std::string getTallyPath( int,int );
    ntally interpolateTally( int );

    // moderator thicknesses simulated
    std::vector<double> my_m;

    // index of moderator thickness
    int my_modIdx;

    // moderator thickness
    double my_modt;

};


#endif
```

Listing B.27: neutronbg.cpp

```
#include "neutronbg.hpp"

#include <cmath>
#include <vector>
#include <string>

#include "fileio.hpp"
#include "phys.hpp"
#include "response.hpp"

using namespace std;

void neutronbg::getDetectorPlane( )
{
    // hard-code in detector planes,
    // should read this in from file eventually
    const double tside = 129.54;
    const double ttop = 259.08;
    const double soff = 68.56;
    my_detXPlane = tside+soff; // side of truck + standoff
    my_detZPlane = ttop+soff; // side of truck + standoff

    // find out if this detector is a "side" or "top" detector
    if ( my_detZPos < my_detZPlane && my_detXPos > tside )
    {
        my_detPosType = side;
        if ( fabs( my_detXPlane-my_detXPos ) > 0.1 )
        {
            // in future, need to add 1/r^2 correction factor
            // and time correction factor
            throw fatal_error("side detector does not lie in x plane");
        }
    }
    else if ( my_detZPos > ttop && my_detXPos < my_detXPlane )
    {
        my_detPosType = top;
        if ( fabs( my_detZPlane-my_detZPos ) > 0.1 )
        {
            throw fatal_error("top detector does not lie in z plane");
        }
    }
    else
    {
        throw fatal_error("invalid detector position");
    }
}

void neutronbg::getDetectorIndices( )
{
    if ( my_detPosType == side )
    {
        my_detPosX = readbin(my_datapath+"side"+sep()+"detposx.dat");
        my_detPosY = readbin(my_datapath+"side"+sep()+"detposy.dat");
        my_detPosZ = readbin(my_datapath+"side"+sep()+"detposz.dat");

        my_detIdx2 = 1;
        for ( unsigned int x=1;x<my_detPosX.size();++x )
        {
            if ( my_detXPos >= my_detPosX[x-1]
                && my_detXPos <= my_detPosX[x] )
            {
                my_detIdx2 = x;
                break;
            }
        }
    }
    else /* if ( my_detPosType == top ) */
    {
        my_detPosX = readbin(my_datapath+"top"+sep()+"detposx.dat");
        my_detPosY = readbin(my_datapath+"top"+sep()+"detposy.dat");
        my_detPosZ = readbin(my_datapath+"top"+sep()+"detposz.dat");

        my_detIdx2 = 1;
        for ( unsigned int z=1;z<my_detPosZ.size();++z )
        {
```

```
            if ( my_detZPos >= my_detPosZ[z-1]
                && my_detZPos <= my_detPosZ[z] )
            {
                my_detIdx2 = z;
                break;
            }
        }
    }
    my_detIdx1 = 1;
    for ( unsigned int y=1;y<my_detPosY.size();++y )
    {
        if ( my_detYPos >= my_detPosY[y-1] && my_detYPos <= my_detPosY[y] )
        {
            my_detIdx1 = y;
            break;
        }
    }

}

void neutronbg::initialize( const std::string& datapath )
{
    my_datapath = datapath+sep();
}

std::string neutronbg::getTallyPath( int detPosIdx1,int detPosIdx2 )
{
    if ( my_detPosType == side )
    {
        return my_datapath+"side"+sep()+"dpos"+str(detPosIdx2)
            +"_"+str(detPosIdx1)+sep();
    }
    else /* if ( my_detPosType == top ) */
    {
        return my_datapath+"top"+sep()+"dpos"+str(detPosIdx2)
            +"_"+str(detPosIdx1)+sep();
    }
}

std::string neutronbg::getTallyErgPath( )
{
    return my_datapath;
}

void neutronbg::buildSource( double detXPos,double detYPos,
    double detZPos,double elevation,double solarLevel,
    double latitude,double longitude )
{
    my_detXPos = detXPos;
    my_detYPos = detYPos;
    my_detZPos = detZPos;
    my_elevation = elevation;
    my_solarLevel = solarLevel;
    my_latitude = latitude;
    my_longitude = longitude;

    getDetectorPlane( );

    getDetectorIndices( );

    //// use measured flux spectrum

    //// get flux
    //vector<double> phi
        //= readtxt<double>("data/nbackground/diffflux.dat");
    //vector<double> erg
        //= readtxt<double>("data/nbackground/difffluxerg.dat");
    //const double maxerg = 20;
    //int idx = 0;
    //for ( unsigned int e=1;e<erg.size();++e )
    //{
        //if ( erg[e] > maxerg )
        //{
            //idx = e;
            //break;
        //}
```

```
    //}
    //erg.erase( erg.begin()+idx,erg.end() );
    //phi.erase( phi.begin()+idx-1,phi.end() );
    //// integrate flux to get neutron density N
    //vector<double> N(phi.size());
    //double sum = 0.0;
    //for ( unsigned int e=0;e<erg.size()-1;++e )
    //{
        //const double meanerg = sqrt(erg[e]*erg[e+1]);
        //const double v = phys::velocity( meanerg,phys::mn );
        ////const double dE = erg[e+1]-erg[e];
        //N[e] = phi[e]*meanerg/v;
        ////N[e] = dE*phi[e]/v;
        //sum += N[e];
    //}
    //spectrum N2(erg);
    //for ( unsigned int e=0;e<erg.size()-1;++e )
    //{
        //N2(e) = datapoint(N[e]/sum,0.0);
    //}



    my_z0 = getTally("z0");
    my_z1 = getTally("z1");
    my_y0 = getTally("y0");
    my_y1 = getTally("y1");
    my_x0 = getTally("x0");
    my_x1 = getTally("x1");


    //response T(erg,my_z0.erg());
    //T.identity();
    //spectrum N3 = T*N2;
    //my_z0 = (N3)*(my_z0.sum());
    //my_z1 = (N3)*(my_z1.sum());
    //my_y0 = (N3)*(my_y0.sum());
    //my_y1 = (N3)*(my_y1.sum());
    //my_x0 = (N3)*(my_x0.sum());
    //my_x1 = (N3)*(my_x1.sum());

    const double scaler = getSourceScaler( my_elevation,
        my_solarLevel,my_latitude,my_longitude );

    my_z0 = my_z0*scaler;
    my_z1 = my_z1*scaler;
    my_y0 = my_y0*scaler;
    my_y1 = my_y1*scaler;
    my_x0 = my_x0*scaler;
    my_x1 = my_x1*scaler;

    //const double cut = 1e-6; // 1 eV
    //int ergidx = 0;
    //for ( int e=0;e<my_z0.numerg();++e )
    //{
        //if ( my_z0.erg(e) > cut )
        //{
            //ergidx = e;
            //break;
        //}
    //}
    //for ( int e=0;e<ergidx;++e )
    //{
        //my_z0(e) = datapoint(0.0,0.0);
        //my_z1(e) = datapoint(0.0,0.0);
        //my_y0(e) = datapoint(0.0,0.0);
        //my_y1(e) = datapoint(0.0,0.0);
        //my_x0(e) = datapoint(0.0,0.0);
        //my_x1(e) = datapoint(0.0,0.0);
    //}


    //my_x0.print();

}
```

```
tally neutronbg::getTally( const std::string& plane )
{

    std::string talPath;
    std::string talErgPath;

    talErgPath = getTallyErgPath( );


    talPath = getTallyPath( my_detIdx1-1,my_detIdx2-1 )+plane+sep();
    tally tal00;
    tal00.parse( talErgPath,talPath );
    tal00.integrateBinWidth();

    talPath = getTallyPath( my_detIdx1-1,my_detIdx2 )+plane+sep();
    tally tal01;
    tal01.parse( talErgPath,talPath );
    tal01.integrateBinWidth();

    talPath = getTallyPath( my_detIdx1,my_detIdx2-1 )+plane+sep();
    tally tal10;
    tal10.parse( talErgPath,talPath );
    tal10.integrateBinWidth();

    talPath = getTallyPath( my_detIdx1,my_detIdx2 )+plane+sep();
    tally tal11;
    tal11.parse( talErgPath,talPath );
    tal11.integrateBinWidth();


    tally tal0 = tally::interpolate( my_detPosY[my_detIdx1-1],tal00,
        my_detPosY[my_detIdx1],tal10,my_detYPos );
    tally tal1 = tally::interpolate( my_detPosY[my_detIdx1-1],tal01,
        my_detPosY[my_detIdx1],tal11,my_detYPos );


    tally tal;
    if ( my_detPosType == side )
    {
        tal = tally::interpolate( my_detPosX[my_detIdx2-1],tal0,
            my_detPosX[my_detIdx2],tal1,my_detXPos );
    }
    else /* if ( my_detPosType == top ) */
    {
        tal = tally::interpolate( my_detPosZ[my_detIdx2-1],tal0,
            my_detPosZ[my_detIdx2],tal1,my_detZPos );
    }



    return tal;
}

// Integrate flux
double neutronbg::getNYCNeutron( )
{
    // get flux
    std::vector<double> phi = readtxt<double>("data/nbackground/diffflux.dat");
    std::vector<double> erg = readtxt<double>("data/nbackground/difffluxerg.dat");

    // set max energy
    const double maxerg = 100; // MeV
    int idx = 0;
    for ( unsigned int e=1;e<erg.size();++e )
    {
        if ( erg[e] > maxerg )
        {
            idx = e;
            break;
        }
    }
    erg.erase( erg.begin()+idx,erg.end() );
    phi.erase( phi.begin()+idx-1,phi.end() );

    // integrate flux to get neutron density N
    std::vector<double> N(phi.size());
```

```
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        const double meanerg = sqrt(erg[e]*erg[e+1]);
        const double v = phys::velocity( meanerg,phys::mn );
        const double dE = erg[e+1]-erg[e];
        N[e] = dE*phi[e]/v;
    }

    // sum over all bins to get total # of neutrons per cubic cm
    double total = 0.0;
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        total += N[e];
    }

    // volume used in simulation
    const double vol = phys::pi*2500*2500*2300;

    return total*vol; // # of neutrons per second
}

// Elevation in Meters
double neutronbg::Pressure( double Elevation )
{
    return pow((44331.514-Elevation)/11880.516,5.255877);
}

// Pressure in hPa
double neutronbg::AtmosphericDepth( double Pressure )
{
    return Pressure/0.980665;
}

double neutronbg::getSourceScaler( double Elevation,
    double SolarLevel,double Latitude,double Longitude )
{
    const double nycnum = getNYCNeutron( );

    const double nycElevation = 10; // meters
    const double nycSolarLevel = 0.5; // average
    const double nycLatitude = 40.77;
    const double nycLongitude = 73.98;
    const double nycScale = getFluxScaler( nycElevation,
        nycSolarLevel,nycLatitude,nycLongitude );

    const double thisScale = getFluxScaler( Elevation,
        SolarLevel,Latitude,Longitude );

    return thisScale/nycScale * nycnum;
}

double neutronbg::getFluxScaler( double Elevation,
    double SolarLevel,double Latitude,double Longitude )
{
    std::string my_datapath = "data/nbackground/";

    const double pressure = Pressure(Elevation); // hPa
    const double depth = AtmosphericDepth(pressure); // cm^2/g
    const double FA = exp((1033.2-depth)/131.3); // unitless

    const double h = pressure/1000.0;
    const double a1 = exp(1.84+0.094*h-0.09*exp(-11.0*h));
    const double k1 = 1.4-0.56*h+0.24*exp(-8.8*h);
    const double a2 = exp(1.93+0.15*h-0.18*exp(-10.0*h));
    const double k2 = 1.32-0.49*h+0.18*exp(-9.5*h);


    std::vector<double> RcMap
        = readtxt<double>(my_datapath+"rigidity.dat");
    std::vector<double> lo
        = readtxt<double>(my_datapath+"longitude.dat");
    const int LO = static_cast<int>(lo.size());
    std::vector<double> la
        = readtxt<double>(my_datapath+"latitude.dat");
    const int LA = static_cast<int>(la.size());
    int loidx1 = 0;
```

```
    int loidx2 = 0;
    if ( Longitude > lo.back() )
    {
        loidx1 = LO-1;
        loidx2 = 0;
    }
    else
    {
        for ( int i=0;i<LO-1;++i )
        {
            if ( Longitude >= lo[i] && Longitude <= lo[i+1] )
            {
                loidx1 = i;
                loidx2 = i+1;
                break;
            }
        }
    }
    // latitude is descending
    int laidx1 = 0;
    int laidx2 = 1;
    for ( int i=0;i<LA-1;++i )
    {
        if ( Latitude <= la[i] && Latitude >= la[i+1] )
        {
            laidx1 = i;
            laidx2 = i+1;
            break;
        }
    }
    const double rclo11 = RcMap[loidx1+LO*laidx1];
    const double rclo12 = RcMap[loidx2+LO*laidx1];
    const double rclo1 = rclo11 + (Longitude-lo[loidx1])
        *(rclo12-rclo11)/(lo[loidx2]-lo[loidx1]);

    const double rclo21 = RcMap[loidx1+LO*laidx2];
    const double rclo22 = RcMap[loidx2+LO*laidx2];
    const double rclo2 = rclo11 + (Longitude-lo[loidx1])
        *(rclo22-rclo21)/(lo[loidx2]-lo[loidx1]);

    const double Rc = rclo1 + (Latitude-la[laidx1])*(rclo2-rclo1)/(la[laidx2]-la[laidx1]);

    const double FBmin = 1.098*(1.0 - exp(-a1/pow(Rc,k1)));
    const double FBmax = 1.098*(1.0 - exp(-a2/pow(Rc,k2)))
        *(1.0-exp(-a1/pow(50.0,k1)))/(1.0-exp(-a2/pow(50.0,k2)));


    const double FB = FBmin*(1.0-SolarLevel) + FBmax*SolarLevel;

    return FA*FB;

}
```

Listing B.28: neutronbg.hpp

```
#ifndef _neutronbg_hpp_included_
#define _neutronbg_hpp_included_

#include <string>
#include "spectrum.hpp"
#include "tally.hpp"

class neutronbg
{
    public:

    enum dpos { side,top };

    dpos getDetPos( ) const { return my_detPosType; };

    double getNYCNeutron( );

    double getSourceScaler( double Elevation,
        double SolarLevel,double Latitude,double Longitude );

    void initialize( const std::string& );

    void buildSource( double,double,double,
        double,double,double,double );

    spectrum getX0Current() const { return my_x0; };
    spectrum getX1Current() const { return my_x1; };
    spectrum getY0Current() const { return my_y0; };
    spectrum getY1Current() const { return my_y1; };
    spectrum getZ0Current() const { return my_z0; };
    spectrum getZ1Current() const { return my_z1; };

    void scaleBy( double scaler )
    {
        my_z0 = my_z0*scaler;
        my_z1 = my_z1*scaler;
        my_y0 = my_y0*scaler;
        my_y1 = my_y1*scaler;
        my_x0 = my_x0*scaler;
        my_x1 = my_x1*scaler;
    }

    void addResponse( const neutronbg& K,double frac )
    {
        my_z0 = my_z0 + K.getZ0Current()*frac;
        my_z1 = my_z1 + K.getZ1Current()*frac;
        my_y0 = my_y0 + K.getY0Current()*frac;
        my_y1 = my_y1 + K.getY1Current()*frac;
        my_x0 = my_x0 + K.getX0Current()*frac;
        my_x1 = my_x1 + K.getX1Current()*frac;
    }

    private:

    std::string my_datapath;

    void getDetectorPlane( );
    void getDetectorIndices( );
    std::string getTallyPath( int detPosIdx1,int detPosIdx2 );
    std::string getTallyErgPath( );
    tally getTally( const std::string& plane );



    spectrum my_z0;
    spectrum my_z1;
    spectrum my_y0;
    spectrum my_y1;
    spectrum my_x0;
    spectrum my_x1;

    // user specified detector positions
    double my_detXPos;
    double my_detYPos;
    double my_detZPos;
```

```
    double my_detXPlane;
    double my_detZPlane;

    // positions used in simulation
    std::vector<double> my_detPosX;
    std::vector<double> my_detPosY;
    std::vector<double> my_detPosZ;

    int my_detIdx1;
    int my_detIdx2;

    dpos my_detPosType;


    double Pressure( double Elevation );

    double AtmosphericDepth( double Pressure );

    double getFluxScaler( double Elevation,
        double SolarLevel,double Latitude,double Longitude );

    double my_elevation;
    double my_solarLevel;
    double my_latitude;
    double my_longitude;
};

#endif
```

Listing B.29: normsource.cpp

```cpp
#include "normsource.hpp"
#include <fstream>
#include <math.h>
#include "extras.hpp"
#include <iostream>
#include "fileio.hpp"

double normsource::getMaxErg( )
{
    //std::cout << my_source.erg() << std::endl;
    double max = std::numeric_limits<double>::min();
    for ( int e=0;e<my_source.numerg();++e )
    {
        if ( my_source.erg(e) > max )
        {
            max = my_source.erg(e);
        }
    }
    return max;
}

int normsource::numOutErgs ( )
{
    return my_source.numerg();
}

normsource::normsource ( const std::string& normpath,
    const std::string& k40normpath,
    const std::string& ra226normpath,
    const std::string& u238normpath,
    const std::string& th232normpath )
{
    initialize( normpath,k40normpath,ra226normpath,u238normpath,th232normpath );
}

void normsource::initialize ( const std::string& normpath,
    const std::string& k40normpath,
    const std::string& ra226normpath,
    const std::string& u238normpath,
    const std::string& th232normpath )
{
    my_normpath = normpath;
    my_k40normpath = k40normpath;
    my_ra226normpath = ra226normpath;
    my_u238normpath = u238normpath;
    my_th232normpath = th232normpath;

    readDataFile();
}

void normsource::readDataFile ( )
{
    // read monoenergetic lines
    std::string tmpstr;
    double tmpdbl;
    double nerg;

    std::ifstream k40normin ( my_k40normpath.c_str() );
    k40normin >> tmpstr;
    k40normin >> nerg;
    my_k40.resize( nerg );
    for ( int e=0;e<nerg;++e )
    {
        k40normin >> tmpdbl;
        my_k40.erg(e) = tmpdbl/1000.0;
        k40normin >> tmpdbl;
        my_k40(e).set( tmpdbl );
        my_k40(e).setErr( 0.0 );
    }
    k40normin.close();

    std::ifstream ra226normin ( my_ra226normpath.c_str() );
    ra226normin >> tmpstr;
    ra226normin >> nerg;
    my_ra226.resize( nerg );
```

```
    for ( int e=0;e<nerg;++e )
    {
        ra226normin >> tmpdbl;
        my_ra226.erg(e) = tmpdbl/1000.0;
        ra226normin >> tmpdbl;
        my_ra226(e).set( tmpdbl );
        my_ra226(e).setErr( 0.0 );
    }
    ra226normin.close();

    std::ifstream u238normin ( my_u238normpath.c_str() );
    u238normin >> tmpstr;
    u238normin >> nerg;
    my_u238.resize( nerg );
    for ( int e=0;e<nerg;++e )
    {
        u238normin >> tmpdbl;
        my_u238.erg(e) = tmpdbl/1000.0;
        u238normin >> tmpdbl;
        my_u238(e).set( tmpdbl );
        my_u238(e).setErr( 0.0 );
    }
    u238normin.close();

    std::ifstream th232normin ( my_th232normpath.c_str() );
    th232normin >> tmpstr;
    th232normin >> nerg;
    my_th232.resize( nerg );
    for ( int e=0;e<nerg;++e )
    {
        th232normin >> tmpdbl;
        //std::cout << tmpdbl << " ";
        my_th232.erg(e) = tmpdbl/1000.0;
        th232normin >> tmpdbl;
        //std::cout << tmpdbl << std::endl;
        my_th232(e).set( tmpdbl );
        my_th232(e).setErr( 0.0 );
    }
    th232normin.close();

}



void normsource::buildResponse( const std::string& normtype,
    double cargomass,double normfrac )
{
    // Get NORM data
    std::ifstream normin ( my_normpath.c_str() );
    if ( Find( normin,normtype,true ) )
    {
        // get activity (Bq/kg) for k40,ra226,u238,th232
        // from norm file in that order
        normin >> my_qk40;
        normin >> my_qra226;
        normin >> my_qu238;
        normin >> my_qth232;
    }
    else
    {
        throw fatal_error ( "invalid cargo NORM type \""
            + normtype + "\"" );
    }
    normin.close();

    // compute actual weight by multiplying by specified fraction
    const double normmass = cargomass*normfrac;

    // in the future, we should sort these energies
    my_source = ( my_k40*my_qk40 + my_ra226*my_qra226
        + my_u238*my_qu238 + my_th232*my_qth232 )*normmass;

    //std::cout << my_k40.erg() << std::endl;
    //std::cout << my_ra226.erg() << std::endl;
    //std::cout << my_u238.erg() << std::endl;
    //std::cout << my_th232.erg() << std::endl;
```

```
    return;
}

dspectrum normsource::operator() ( )
{
    return my_source;
}
```

Listing B.30: normsource.hpp

```
#ifndef _normsource_hpp_included_
#define _normsource_hpp_included_
#include <string>
#include <vector>
#include <map>

#include "submodel.hpp"
#include "dspectrum.hpp"
#include "datapoint.hpp"


class normsource
{
    public:

    void readDataFile();

    double getMaxErg( );

    void initialize ( const std::string&,
        const std::string&,
        const std::string&,
        const std::string&,
        const std::string& );

    void buildResponse ( const std::string&,
        double,
        double );

    normsource ( const std::string&,
        const std::string&,
        const std::string&,
        const std::string&,
        const std::string& );

    normsource ( ) { };

    dspectrum operator() ( );

    private:


    int numOutErgs ( );

    dspectrum my_k40;
    dspectrum my_ra226;
    dspectrum my_u238;
    dspectrum my_th232;
    dspectrum my_source;

    double my_qk40;
    double my_qra226;
    double my_qu238;
    double my_qth232;

    std::string my_normpath;
    std::string my_k40normpath;
    std::string my_ra226normpath;
    std::string my_u238normpath;
    std::string my_th232normpath;

};

#endif
```

Listing B.31: nshield.cpp

```cpp
#include "nshield.hpp"

#include "fileio.hpp"

void nshield::initialize( const std::string& datapath )
{
    my_datapath = datapath;
}

void nshield::getConvexityData( )
{
    // read convexity coefficient
    my_C = readtxt<double>(my_datapath+"c.dat");
    std::vector<double> temperg = readtxt<double>(my_datapath+"e.dat");
    my_Ce.resize(temperg.size()-1);
    for ( unsigned int i=0;i<temperg.size()-1;++i )
    {
        my_Ce[i] = sqrt(temperg[i]*temperg[i+1]);
    }
    //std::cout << "my_Ce = " << my_Ce << std::endl;
    my_Ct = readtxt<double>(my_datapath+"t.dat");
    //std::cout << "my_Ct = " << my_Ct << std::endl;
}

int nshield::getThicknessConvexityIndex( double thickness )
{
    // find convexity t index
    int contidx = 1;
    if ( thickness > my_Ct.back() )
    {
        contidx = my_Ct.size()-1;
    }
    else
    {
        for ( unsigned int i=0;i<my_Ct.size()-1;++i )
        {
            if ( thickness > my_Ct[i] )
            {
                contidx = i+1;
            }
        }
    }
    return contidx;
}

int nshield::getEnergyConvexityIndex( double erg )
{
    // find convexity t index
    int coneidx = 1;
    if ( erg > my_Ce.back() )
    {
        coneidx = my_Ce.size()-1;
    }
    else
    {
        for ( unsigned int i=0;i<my_Ce.size()-1;++i )
        {
            if ( erg > my_Ce[i] )
            {
                coneidx = i+1;
            }
        }
    }
    return coneidx;
}

double nshield::getConvexityCoefficient( double energy,double thickness )
{
    int contidx = getThicknessConvexityIndex(thickness);
    int coneidx = getEnergyConvexityIndex(energy);
    // calculate convexity coefficient
    //const double con1 = my_C[coneidx+my_Ce.size()*(contidx-1)];
    //const double con2 = my_C[coneidx+my_Ce.size()*(contidx)];
    const double e1 = my_Ce[coneidx-1];
    const double e2 = my_Ce[coneidx];
```

```
    const double t1 = my_Ct[contidx-1];
    const double t2 = my_Ct[contidx];
    const double cont1e1 = my_C[coneidx-1+my_Ce.size()*(contidx-1)];
    const double cont1e2 = my_C[coneidx+my_Ce.size()*(contidx-1)];
    const double cont2e1 = my_C[coneidx-1+my_Ce.size()*(contidx)];
    const double cont2e2 = my_C[coneidx+my_Ce.size()*(contidx)];
    const double cont1 = cont1e1 + (energy-e1)*(cont1e2-cont1e1)/(e2-e1);
    const double cont2 = cont2e1 + (energy-e1)*(cont2e2-cont2e1)/(e2-e1);
    const double con = cont1 + (thickness-t1)*(cont2-cont1)/(t2-t1);
    return con;
}

double nshield::getScaler( double c )
{
    double scaler = 1.0;
    const double ratio = plval/ptval;
    if ( ratio > std::numeric_limits<double>::min()
        && ! isnan(ratio)
        && ! isinf(ratio) )
    {
        scaler = exp(log(plval/ptval)*pow(my_R,c)) / (plval/ptval);
    }
    return scaler;
}

void nshield::buildData( )
{
    getConvexityData( );

    my_srcErg = readbin(my_datapath+"srcerg.dat");
    const int Ein = static_cast<int>(my_srcErg.size());

    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {
        // read in radii
        std::vector<double> rad = readbin(my_datapath+"erg"+str(e)+sep()+"rad.dat");
        const int R = static_cast<int>(rad.size());
        int ridx = 1;
        // find index one over this thickness
        for ( int r=1;r<R;++r )
        {
            if ( my_thickness >= rad[r-1] && my_thickness <= rad[r] )
            {
                ridx = r;
            }
        }

        std::string talPath;
        std::string talErgPath;
        talPath = my_datapath+"erg"+str(e)+sep()+"rad"+str(ridx-1)+sep();
        talErgPath = my_datapath+"erg"+str(e)+sep();
        ntally tal1;
        tal1.parse(talErgPath,talPath);
        double ptval1 = readVal<double>(talPath+"ptval.dat");

        talPath = my_datapath+"erg"+str(e)+sep()+"rad"+str(ridx)+sep();
        talErgPath = my_datapath+"erg"+str(e)+sep();
        ntally tal2;
        tal2.parse(talErgPath,talPath);

        double ptval2 = readVal<double>(talPath+"ptval.dat");
        ptval = logInterpolate(rad[ridx-1],ptval1,
            rad[ridx],ptval2,my_thickness);

        my_data[e] = ntally::loginterpolate(rad[ridx-1],
            tal1,rad[ridx],tal2,my_thickness);
        plval = my_data[e].sum().get();

        // get convexity coefficient
        const double c = getConvexityCoefficient(
            sqrt(my_srcErg[e]*my_srcErg[e+1]),my_thickness );
        const double scaler = getScaler(c);
        my_data[e].scaleBy( scaler );
    }
    assignDataToResponse( );
```

```
}

void nshield::buildResponse( double innerRadius,double thickness )
{
    my_thickness = thickness;
    my_ri = innerRadius;
    my_ro = innerRadius+my_thickness;
    my_R = my_ri/my_ro;

    buildData( );
}

tspectrum nshield::operator() ( const tspectrum& S )
{
    return my_K*S;
}

void nshieldrin::getConvexityData( )
{
    // read convexity coefficient
    my_C = readtxt<double>(my_datapath+"c-.dat");
    std::vector<double> temperg = readtxt<double>(my_datapath+"e-.dat");
    my_Ce.resize(temperg.size()-1);
    for ( unsigned int i=0;i<temperg.size()-1;++i )
    {
        my_Ce[i] = sqrt(temperg[i]*temperg[i+1]);
    }
    //std::cout << "my_Ce = " << my_Ce << std::endl;
    my_Ct = readtxt<double>(my_datapath+"t-.dat");
    //std::cout << "my_Ct = " << my_Ct << std::endl;
}


double nshieldrin::getScaler( double c )
{
    return (1.0 - pow(1.0-my_R,c));
}

void nshieldrin::buildData( )
{
    getConvexityData( );

    my_srcErg = readbin(my_datapath+"srcerg.dat");
    const int Ein = static_cast<int>(my_srcErg.size());

    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {

        // read in radii
        std::vector<double> rad = readbin(my_datapath+"erg"
            +str(e)+sep()+"rad.dat");
        const int R = static_cast<int>(rad.size());
        int ridx = 1;
        // find index one over this thickness
        for ( int r=1;r<R;++r )
        {
            if ( my_thickness >= rad[r-1] && my_thickness <= rad[r] )
            {
                ridx = r;
            }
        }

        std::string talPath;
        std::string talErgPath;
        talPath = my_datapath+"erg"+str(e)+sep()+"rad"+str(ridx-1)+sep();
        talErgPath = my_datapath+"erg"+str(e)+sep();
        ntally tal1;
        tal1.parse(talErgPath,talPath);

        talPath = my_datapath+"erg"+str(e)+sep()+"rad"+str(ridx)+sep();
        talErgPath = my_datapath+"erg"+str(e)+sep();
        ntally tal2;
        tal2.parse(talErgPath,talPath);

        my_data[e] = ntally::loginterpolate(rad[ridx-1],
```

```
            tal1 ,rad[ridx],tal2 ,my_thickness);

        // get convexity coefficient
        const double c = getConvexityCoefficient(
            sqrt(my_srcErg[e]*my_srcErg[e+1]),my_thickness );
        const double scaler = getScaler(c);
        my_data[e].scaleBy( scaler );
    }
    assignDataToResponse( );
}

void nshieldrout::getConvexityData( )
{
    // read convexity coefficient
    my_C = readtxt<double>(my_datapath+"c+.dat");
    std::vector<double> temperg =
        readtxt<double>(my_datapath+"e+.dat");
    my_Ce.resize(temperg.size()-1);
    for ( unsigned int i=0;i<temperg.size()-1;++i )
    {
        my_Ce[i] = sqrt(temperg[i]*temperg[i+1]);
    }
    //std::cout << "my_Ce = " << my_Ce << std::endl;
    my_Ct = readtxt<double>(my_datapath+"t+.dat");
    //std::cout << "my_Ct = " << my_Ct << std::endl;
}
```

Listing B.32: nshield.hpp

```
#ifndef _nshield_hpp_included_
#define _nshield_hpp_included_

#include "tresponse.hpp"
#include "nsubmodel.hpp"

class nshield : public nsubmodel
{
    public:

    void initialize( const std::string& );

    void buildResponse( double,double );

    tspectrum operator() ( const tspectrum& S );

    protected:


    std::string my_cfile;
    std::string my_cefile;
    std::string my_ctfile;
    double my_thickness;
    double my_ri;
    double my_ro;
    double my_R;
    std::vector<double> my_C;
    std::vector<double> my_Ce;
    std::vector<double> my_Ct;

    double getConvexityCoefficient( double energy,double thickness );
    int getEnergyConvexityIndex( double erg );
    int getThicknessConvexityIndex( double thickness );

    virtual void getConvexityData( );
    virtual void buildData( );
    virtual double getScaler( double c );

    private:

    double ptval;
    double plval;

};

class nshieldrin : public nshield
{
    public:

    private:

    void buildData( );
    double getScaler( double c );

    void getConvexityData( );

};

class nshieldrout : public nshield
{
    public:


    private:

    double ptval;
    double plval;

    void getConvexityData( );

};


#endif
```

Listing B.33: nsnm.cpp

```
#include "nsnm.hpp"
#include <limits>
#include "diagnostic.hpp"
#include "fileio.hpp"
#include "tally.hpp"
#include "phys.hpp"

using namespace std;

int nsnmbase::nIso( )
{
    return static_cast<int>(my_frac.size());
}

double nsnmbase::getMass( double mass,
    const vector<double>& nomfrac,int idx,double f )
{
    vector<double> isomass( nomfrac.size() );
    for ( unsigned int i=0;i<nomfrac.size();++i )
    {
        isomass[i] = mass*nomfrac[i]/100.0;
    }
    isomass[idx] = isomass[idx] + mass*(f-nomfrac[idx])/100.0;
    const double totmass = sum(isomass);
    return totmass;
}

void nsnmbase::getMassIdx( )
{
    if ( my_snmMass > my_mass.back() )
    {
        my_massIdx = my_mass.size()-1;
        global::warn.add("SNM mass exceeds simulated \
limits, extrapolating");
    }
    else if ( my_snmMass < my_mass.front() )
    {
        my_massIdx = 1;
        global::warn.add("SNM mass is below simulated \
limits, extrapolating");
    }
    else
    {
        // find mass index
        my_massIdx =1;
        for ( unsigned int i=1;i<my_mass.size();++i )
        {
            if ( my_snmMass > my_mass[i-1]
                && my_snmMass <= my_mass[i] )
            {
                my_massIdx = i;
                break;
            }
        }
    }
}

void nsnm::initialize( const std::string& datapath,
    const std::string& sfapath )
{
    my_datapath = datapath+sep();
    my_sfapath = sfapath+sep();;
}


double nsnm::getRadius( )
{
    return pow ( my_snmMass/my_rho * ( 3.0/4.0 )
        * ( 1.0/phys::pi ),1.0/3.0 );
}

void nsnm::buildResponse( double mass,
    const std::vector<std::string>& isoName,
    const std::vector<double>& frac )
{
```

```
    my_isoName = isoName;
    my_frac = frac*100.0; // frac is normalized, want it in %

    // see if any isoName's don't exist in the file, in which
    // case they weren't important enough
    for ( unsigned int i=0;i<my_isoName.size();++i )
    {
        if ( ! exists( my_datapath+my_isoName[i]+"frac.dat" ) )
        {
            my_isoName.erase(my_isoName.begin()+i);
            my_frac.erase(my_frac.begin()+i);
            i--;
            continue;
        }
    }
    if ( my_isoName.size() == 0 )
    {
        throw fatal_error("SNM type does not have any isotopes");
    }

    my_snmMass = mass;

    my_isoFrac.resize(nIso());
    for ( int i=0;i<nIso();++i )
    {
        my_isoFrac[i] = readbin(my_datapath
            +my_isoName[i]+"frac.dat");
    }
    my_mass = readbin(my_datapath+"mass.dat");
    my_nomfrac = readbin(my_datapath+"nomfrac.dat");
    my_rho = readVal<double>(my_datapath+"rho.dat");

    // find closest iso index to desired fraction
    my_isoIdx.resize(nIso());
    for ( int i=0;i<nIso();++i )
    {
        if ( my_frac[i] > my_nomfrac[i] )
        {
            my_isoIdx[i] = 1;
            if ( my_frac[i] > my_isoFrac[i][my_isoIdx[i]] )
            {
                global::warn.add("extrapolating isotope "
                    +my_isoName[i]+" above "
                    +str(my_isoFrac[i][my_isoIdx[i]]));
            }
        }
        else
        {
            my_isoIdx[i] = 0;
            if ( my_frac[i] < my_isoFrac[i][my_isoIdx[i]] )
            {
                global::warn.add("extrapolating isotope "
                    +my_isoName[i]+" below "
                    +str(my_isoFrac[i][my_isoIdx[i]]));
            }
        }

    }

    getMassIdx( );

    // calculate source strength
    my_sourceStrength = getSFSource( );

    generateNeutronSpectrum( );

    generateGammaSpectrum( );

    std::cout << "Spontaneous Fission Source  : "
        << std::setw(15) << my_sourceStrength
        << " fissions/s" << std::endl;
}


void nsnm::generateNeutronSpectrum( )
{
```

```
    std::string talPath;


    // read nominal fraction results
    ntally talnomm0;
    talPath = my_datapath+"mass"+str(my_massIdx-1)+sep()+"nom"+sep();
    talnomm0.parse(talPath,talPath);

    ntally talnomm1;
    talPath = my_datapath+"mass"+str(my_massIdx)+sep()+"nom"+sep();
    talnomm1.parse(talPath,talPath);

    // read results for each perturbation
    std::vector<ntally> talim0(nIso());
    std::vector<ntally> talim1(nIso());
    for ( int i=0;i<nIso();++i )
    {
        talPath = my_datapath+"mass"+str(my_massIdx-1)+sep()
            +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])+sep();
        talim0[i].parse(talPath,talPath);

        talPath = my_datapath+"mass"+str(my_massIdx)+sep()
            +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])+sep();
        talim1[i].parse(talPath,talPath);
    }

    // interpolate based on isotopic change
    ntally talm0 = talnomm0;
    ntally talm1 = talnomm1;
    for ( int i=0;i<nIso();++i )
    {
        talm0 = ntally::interpolate( my_nomfrac[i],talm0,
            my_isoFrac[i][my_isoIdx[i]],talim0[i],my_frac[i] );
        talm1 = ntally::interpolate( my_nomfrac[i],talm1,
            my_isoFrac[i][my_isoIdx[i]],talim1[i],my_frac[i] );
    }

    ntally tal = ntally::interpolate(my_mass[my_massIdx-1],
        talnomm0,my_mass[my_massIdx],talnomm1,my_snmMass);

    // assign tally to spectrum
    my_neutronSpectrum = tal;
    my_neutronSpectrum = my_neutronSpectrum*my_sourceStrength;
}


void nsnm::generateGammaSpectrum( )
{
    std::string talPath;

    // read nominal fraction results
    tally talnomm0;
    talPath = my_datapath+"mass"+str(my_massIdx-1)+sep()
        +"nom"+sep()+"gam"+sep();
    talnomm0.parse(talPath,talPath);
    tally talnomm1;
    talPath = my_datapath+"mass"+str(my_massIdx)+sep()
        +"nom"+sep()+"gam"+sep();
    talnomm1.parse(talPath,talPath);
    // read results for each perturbation
    std::vector<tally> talim0(nIso());
    std::vector<tally> talim1(nIso());
    for ( int i=0;i<nIso();++i )
    {
        talPath = my_datapath+"mass"+str(my_massIdx-1)+sep()
            +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])
            +sep()+"gam"+sep();
        talim0[i].parse(talPath,talPath);

        talPath = my_datapath+"mass"+str(my_massIdx)+sep()
            +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])
            +sep()+"gam"+sep();
        talim1[i].parse(talPath,talPath);
    }
    // interpolate based on isotopic change
    tally talm0 = talnomm0;
```

```
    tally talm1 = talnomm1;
    for ( int i=0;i<nIso();++i )
    {
        talm0 = tally::interpolate( my_nomfrac[i],talm0,
            my_isoFrac[i][my_isoIdx[i]],talim0[i],my_frac[i] );
        talm1 = tally::interpolate( my_nomfrac[i],talm1,
            my_isoFrac[i][my_isoIdx[i]],talim1[i],my_frac[i] );
    }
    // interpolate on total mass
    tally tal;
    tal = tally::interpolate(my_mass[my_massIdx-1],
        talm0,my_mass[my_massIdx],talm1,my_snmMass);

    my_gammaSpectrum = tal;
    my_gammaSpectrum = my_gammaSpectrum*my_sourceStrength;
}


int nsnm::getWeight( const std::string& iso )
{
    return fromstr<int>(iso.substr(iso.size()-3,3));
}

double nsnm::getSFSource( )
{
    double result = 0.0;
    for ( int i=0;i<nIso();++i )
    {
        //std::cout << "isotope " << my_isoName[i] << std::endl;
        const int weight = getWeight(my_isoName[i]);
        //std::cout << "weight = " << weight << std::endl;
        string filename = my_sfapath+my_isoName[i]+".dat";
        //std::cout << "filename = " << filename << std::endl;
        ifstream in(filename.c_str());
        double halflife; in >> halflife;
        //std::cout << "half life =  " << halflife << std::endl;
        double sffrac; in >> sffrac;
        //std::cout << "sf frac =  " << sffrac << std::endl;
        result += my_frac[i]/100.0*log(2)/halflife * sffrac
            * phys::avo / weight;
    }
    result = result*my_snmMass;
    return result;
}

void nsnmout::initialize( const std::string& datapath )
{
    my_datapath = datapath+sep();
}

void nsnmout::buildResponse( double mass,
    const std::vector<std::string>& isoName,
    const std::vector<double>& frac )
{

    my_isoName = isoName;
    my_frac = frac*100.0; // frac is normalized, want it in %

    // see if any isoName's don't exist in the file, in which
    // case they weren't important enough
    for ( unsigned int i=0;i<my_isoName.size();++i )
    {
        if ( ! exists( my_datapath+my_isoName[i]+"frac.dat" ) )
        {
            my_isoName.erase(my_isoName.begin()+i);
            my_frac.erase(my_frac.begin()+i);
            i--;
            continue;
        }
    }
    if ( my_isoName.size() == 0 )
    {
        throw fatal_error("SNM type does not have any isotopes");
    }
```

```
    my_snmMass = mass;

    my_isoFrac.resize(nIso());
    for ( int i=0;i<nIso();++i )
    {
        my_isoFrac[i] = readbin(my_datapath+my_isoName[i]+"frac.dat");
    }
    my_mass = readbin(my_datapath+"mass.dat");
    my_nomfrac = readbin(my_datapath+"nomfrac.dat");
    my_rho = readVal<double>(my_datapath+"rho.dat");



    // find closest iso index to desired fraction
    my_isoIdx.resize(nIso());
    for ( int i=0;i<nIso();++i )
    {
        if ( my_frac[i] > my_nomfrac[i] )
        {
            my_isoIdx[i] = 1;
            if ( my_frac[i] > my_isoFrac[i][my_isoIdx[i]] )
            {
                global::warn.add("extrapolating isotope "
                +my_isoName[i]+" above "
                +str(my_isoFrac[i][my_isoIdx[i]]));
            }
        }
        else
        {
            my_isoIdx[i] = 0;
            if ( my_frac[i] < my_isoFrac[i][my_isoIdx[i]] )
            {
                global::warn.add("extrapolating isotope "
                    +my_isoName[i]+" below "
                    +str(my_isoFrac[i][my_isoIdx[i]]));
            }
        }
    }

    getMassIdx( );

    generateResponse();

}


void nsnmout::generateResponse( )
{


    my_srcErg = readbin(my_datapath+"srcerg.dat");
    const int Ein = static_cast<int>(my_srcErg.size());

    my_data.resize(Ein);
    for ( int e=0;e<Ein-1;++e )
    {
        std::string talPath;

        // read nominal fraction results
        ntally talnomm0;
        talPath = my_datapath+"mass"+str(my_massIdx-1)
            +sep()+"nom"+sep()+"erg"+str(e)+sep();
        talnomm0.parse(talPath,talPath);
        ntally talnomm1;
        talPath = my_datapath+"mass"+str(my_massIdx)
            +sep()+"nom"+sep()+"erg"+str(e)+sep();
        talnomm1.parse(talPath,talPath);

        // read results for each perturbation
        std::vector<ntally> talim0(nIso());
        std::vector<ntally> talim1(nIso());
        for ( int i=0;i<nIso();++i )
        {
            talPath = my_datapath+"mass"+str(my_massIdx-1)+sep()
```

```
                +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])
                +sep()+"erg"+str(e)+sep();
            talim0[i].parse(talPath,talPath);

            talPath = my_datapath+"mass"+str(my_massIdx)+sep()
                +my_isoName[i]+sep()+"frac"+str(my_isoIdx[i])
                +sep()+"erg"+str(e)+sep();
            talim1[i].parse(talPath,talPath);
        }

        // interpolate based on isotopic change
        ntally talm0 = talnomm0;
        ntally talm1 = talnomm1;
        for ( int i=0;i<nIso();++i )
        {
            talm0 = ntally::interpolate( my_nomfrac[i],talm0,
                my_isoFrac[i][my_isoIdx[i]],talim0[i],my_frac[i] );
            talm1 = ntally::interpolate( my_nomfrac[i],talm1,
                my_isoFrac[i][my_isoIdx[i]],talim1[i],my_frac[i] );
        }
        my_data[e] = ntally::interpolate(my_mass[my_massIdx-1],
            talnomm0,my_mass[my_massIdx],talnomm1,my_snmMass);
    }


    assignDataToResponse( );


}


tspectrum nsnmout::operator() ( const tspectrum& S )
{
    return my_K*S;
}
```

Listing B.34: nsnm.hpp

```
#ifndef _nsnm_hpp_included_
#define _nsnm_hpp_included_

#include <string>
#include <vector>

#include "tspectrum.hpp"
#include "nsubmodel.hpp"

class nsnmbase
{
    public:



    protected:

    double getMass( double mass,
        const std::vector<double>& nomfrac,
        int idx,
        double f );
    int nIso( );

    void getMassIdx( );

    int my_massIdx;
    std::vector<std::string> my_isoName;
    std::vector< std::vector<double> > my_isoFrac;
    std::vector<int> my_isoIdx;
    std::vector<double> my_frac;
    std::vector<double> my_nomfrac;
    std::vector<double> my_mass;
    double my_snmMass;
    double my_rho;


};


class nsnm : public nsnmbase
{
    public:

    void initialize( const std::string& datapath,
        const std::string& sfapath );

    tspectrum getNeutronSpectrum() { return my_neutronSpectrum; };
    spectrum getGammaSpectrum() { return my_gammaSpectrum; };


    void buildResponse( double mass,
        const std::vector<std::string>& isoName,
        const std::vector<double>& frac );

    double getRadius( );

    private:

    std::string my_datapath;

    // spontaneous fission library path
    std::string my_sfapath;

    void generateNeutronSpectrum( );
    void generateGammaSpectrum( );

    tspectrum my_neutronSpectrum;
    spectrum my_gammaSpectrum;

    int getWeight( const std::string& iso );
    double getSFSource( );
    double my_sourceStrength;

};
```

```
class nsnmout : public nsnmbase, public nsubmodel
{
    public:

    void initialize( const std::string& datapath );

    void buildResponse( double mass,
        const std::vector<std::string>& isoName,
        const std::vector<double>& frac );

    tspectrum operator() ( const tspectrum& S );

    private:

    void generateResponse( );
};


#endif
```

Listing B.35: nsubmodel.cpp

```cpp
#include "nsubmodel.hpp"



void nsubmodel::assignDataToResponse( )
{
    //std::cout << "in assign data" << std::endl;
    const int T = my_data[0].nTime();
    //std::cout << "T = " << T << std::endl;
    const int Eout = my_data[0].nErg();
    //std::cout << "Eout = " << Eout << std::endl;
    const int Ein = static_cast<int>(my_srcErg.size());
    //std::cout << "Ein = " << Ein << std::endl;
    my_K.setTime( my_data[0].time() );
    for ( int t=0;t<T-1;++t )
    {
        my_K(t).initialize( my_srcErg,my_data[0].erg() );
        for ( int ei=0;ei<Ein-1;++ei )
        {
            for ( int eo=0;eo<Eout-1;++eo )
            {
                my_K(t)(eo,ei) = my_data[ei].val(eo+1,t+1);
            }
        }
        my_K(t).optimize();
    }
    //std::cout << "my_K.sum() = " << my_K.sum() << std::endl;
}
```

Listing B.36: nsubmodel.hpp

```
#ifndef _nsubmodel_included_hpp
#define _nsubmodel_included_hpp

#include <vector>
#include "tally.hpp"
#include "tresponse.hpp"
#include <string>

class nsubmodel
{
    public:

    tresponse getK( ) const { return my_K; };

    void scaleBy( double scaler ) { my_K.scaleBy(scaler); };

    protected:

    std::string my_datapath;
    tresponse my_K;
    std::vector<double> my_srcErg;
    std::vector<ntally> my_data;
    void assignDataToResponse( );

};

#endif
```

Listing B.37: nvehicle.cpp

```
#include "nvehicle.hpp"
#include "tally.hpp"
#include "fileio.hpp"
#include "extras.hpp"
#include "phys.hpp"


void nvehicle::readDataFile()
{
    // get list of source energies
    //my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // get detector positions/areas
    my_detSidePtX = readbin( my_datapath+"side/detposx.dat" );
    my_detSidePtY = readbin( my_datapath+"side/detposy.dat" );
    my_detSidePtZ = readbin( my_datapath+"side/detposz.dat" );
    my_detSideArea = readbin( my_datapath+"side/detarea.dat" );
    my_detTopPtX = readbin( my_datapath+"top/detposx.dat" );
    my_detTopPtY = readbin( my_datapath+"top/detposy.dat" );
    my_detTopPtZ = readbin( my_datapath+"top/detposz.dat" );
    my_detTopArea = readbin( my_datapath+"top/detarea.dat" );
    // source positions
    my_srcPtX = readbin( my_datapath+"srcposx.dat" );
    my_srcPtY = readbin( my_datapath+"srcposy.dat" );
    my_srcPtZ = readbin( my_datapath+"srcposz.dat" );

    return;
}

nvehicle::nvehicle ( const std::string& path )
{
    initialize( path );
}

void nvehicle::initialize( const std::string& path )
{
    my_datapath = path + sep();
    readDataFile();
}


std::string nvehicle::getTallyEnergyPath( )
{
    return my_datapath;
}

std::string nvehicle::getTallyPath( int detRowIdx,
    int detColIdx,int srcIdxX,int srcIdxY,int srcIdxZ )
{
    int detRow,detCol;
    if ( detRowIdx == 1 )
        detRow = my_detIdxRow1;
    else /*if ( detRowIdx == 2 )*/
        detRow = my_detIdxRow2;
    if ( detColIdx == 1 )
        detCol = my_detIdxCol1;
    else /*if ( detColIdx == 2 )*/
        detCol = my_detIdxCol2;
    int srcX,srcY,srcZ;
    if ( srcIdxX == 1 )
        srcX = my_srcIdxX1;
    else /*if ( srcIdxX == 2 )*/
        srcX = my_srcIdxX2;
    if ( srcIdxY == 1 )
        srcY = my_srcIdxY1;
    else /*if ( srcIdxY == 2 )*/
        srcY = my_srcIdxY2;
    if ( srcIdxZ == 1 )
        srcZ = my_srcIdxZ1;
    else /*if ( srcIdxZ == 2 )*/
        srcZ = my_srcIdxZ2;


    if ( my_detPosType == side )
    {
        return my_datapath + "side"
```

```
            + sep() + "dpos" + str(detRow) + "_" + str(detCol)
            + sep() + "spos" + str(srcX) + "_" + str(srcY) + "_"
            + str(srcZ) + sep();
    }
    else /*if ( my_detPosType == top )*/
    {
        return my_datapath + "top"
            + sep() + "dpos" + str(detRow) + "_" + str(detCol)
            + sep() + "spos" + str(srcX) + "_" + str(srcY) + "_"
            + str(srcZ) + sep();
    }
}

ntally2 nvehicle::interpolateSourceXZ( const ntally2& talx1z1,
    const ntally2& talx1z2,const ntally2& talx2z1,
    const ntally2& talx2z2 )
{
    ntally2 talx1 = ntally2::interpolate( my_srcPtZ[my_srcIdxZ1],talx1z1,
        my_srcPtZ[my_srcIdxZ2],talx1z2,my_srcPosZ );
    ntally2 talx2 = ntally2::interpolate( my_srcPtZ[my_srcIdxZ1],talx2z1,
        my_srcPtZ[my_srcIdxZ2],talx2z2,my_srcPosZ );
    ntally2 result = ntally2::interpolate( my_srcPtX[my_srcIdxX1],talx1,
        my_srcPtX[my_srcIdxX2],talx2,my_srcPosX );

    return result;
}

ntally2 nvehicle::interpolateDetectorRow( const ntally2& talRow1,
    const ntally2& talRow2 )
{
    ntally2 result;

    if ( my_detPosType == side )
    {
        result = ntally2::interpolate( my_detSidePtZ[my_detIdxRow1],
            talRow1,my_detSidePtZ[my_detIdxRow2],talRow2,my_detPosZ );
    }
    else /*if ( my_detPosType == top )*/
    {
        result = ntally2::interpolate( my_detTopPtX[my_detIdxRow1],
            talRow1,my_detTopPtX[my_detIdxRow2],talRow2,my_detPosX );
    }
    return result;
}

ntally2 nvehicle::interpolateDetectorsAndSources( )
{
    std::string talErgPath = getTallyEnergyPath();
    std::string talPath;

    ntally2 tal_r1c1_x1y1z1;
    talPath = getTallyPath( 1,1,1,1,1 );
    tal_r1c1_x1y1z1.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x1y1z2;
    talPath = getTallyPath( 1,1,1,1,2 );
    tal_r1c1_x1y1z2.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x1y2z1;
    talPath = getTallyPath( 1,1,1,2,1 );
    tal_r1c1_x1y2z1.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x1y2z2;
    talPath = getTallyPath( 1,1,1,2,2 );
    tal_r1c1_x1y2z2.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x2y1z1;
    talPath = getTallyPath( 1,1,2,1,1 );
    tal_r1c1_x2y1z1.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x2y1z2;
    talPath = getTallyPath( 1,1,2,1,2 );
    tal_r1c1_x2y1z2.parse( talErgPath,talPath );

    ntally2 tal_r1c1_x2y2z1;
    talPath = getTallyPath( 1,1,2,2,1 );
```

```
tal_r1c1_x2y2z1.parse( talErgPath,talPath );

ntally2 tal_r1c1_x2y2z2;
talPath = getTallyPath( 1,1,2,2,2 );
tal_r1c1_x2y2z2.parse( talErgPath,talPath );



ntally2 tal_r1c2_x1y1z1;
talPath = getTallyPath( 1,2,1,1,1 );
tal_r1c2_x1y1z1.parse( talErgPath,talPath );

ntally2 tal_r1c2_x1y1z2;
talPath = getTallyPath( 1,2,1,1,2 );
tal_r1c2_x1y1z2.parse( talErgPath,talPath );

ntally2 tal_r1c2_x1y2z1;
talPath = getTallyPath( 1,2,1,2,1 );
tal_r1c2_x1y2z1.parse( talErgPath,talPath );

ntally2 tal_r1c2_x1y2z2;
talPath = getTallyPath( 1,2,1,2,2 );
tal_r1c2_x1y2z2.parse( talErgPath,talPath );

ntally2 tal_r1c2_x2y1z1;
talPath = getTallyPath( 1,2,2,1,1 );
tal_r1c2_x2y1z1.parse( talErgPath,talPath );

ntally2 tal_r1c2_x2y1z2;
talPath = getTallyPath( 1,2,2,1,2 );
tal_r1c2_x2y1z2.parse( talErgPath,talPath );

ntally2 tal_r1c2_x2y2z1;
talPath = getTallyPath( 1,2,2,2,1 );
tal_r1c2_x2y2z1.parse( talErgPath,talPath );

ntally2 tal_r1c2_x2y2z2;
talPath = getTallyPath( 1,2,2,2,2 );
tal_r1c2_x2y2z2.parse( talErgPath,talPath );




ntally2 tal_r2c1_x1y1z1;
talPath = getTallyPath( 2,1,1,1,1 );
tal_r2c1_x1y1z1.parse( talErgPath,talPath );

ntally2 tal_r2c1_x1y1z2;
talPath = getTallyPath( 2,1,1,1,2 );
tal_r2c1_x1y1z2.parse( talErgPath,talPath );

ntally2 tal_r2c1_x1y2z1;
talPath = getTallyPath( 2,1,1,2,1 );
tal_r2c1_x1y2z1.parse( talErgPath,talPath );

ntally2 tal_r2c1_x1y2z2;
talPath = getTallyPath( 2,1,1,2,2 );
tal_r2c1_x1y2z2.parse( talErgPath,talPath );

ntally2 tal_r2c1_x2y1z1;
talPath = getTallyPath( 2,1,2,1,1 );
tal_r2c1_x2y1z1.parse( talErgPath,talPath );

ntally2 tal_r2c1_x2y1z2;
talPath = getTallyPath( 2,1,2,1,2 );
tal_r2c1_x2y1z2.parse( talErgPath,talPath );

ntally2 tal_r2c1_x2y2z1;
talPath = getTallyPath( 2,1,2,2,1 );
tal_r2c1_x2y2z1.parse( talErgPath,talPath );

ntally2 tal_r2c1_x2y2z2;
talPath = getTallyPath( 2,1,2,2,2 );
tal_r2c1_x2y2z2.parse( talErgPath,talPath );
```

```
ntally2 tal_r2c2_x1y1z1;
talPath = getTallyPath( 2,2,1,1,1 );
tal_r2c2_x1y1z1.parse( talErgPath,talPath );

ntally2 tal_r2c2_x1y1z2;
talPath = getTallyPath( 2,2,1,1,2 );
tal_r2c2_x1y1z2.parse( talErgPath,talPath );

ntally2 tal_r2c2_x1y2z1;
talPath = getTallyPath( 2,2,1,2,1 );
tal_r2c2_x1y2z1.parse( talErgPath,talPath );

ntally2 tal_r2c2_x1y2z2;
talPath = getTallyPath( 2,2,1,2,2 );
tal_r2c2_x1y2z2.parse( talErgPath,talPath );

ntally2 tal_r2c2_x2y1z1;
talPath = getTallyPath( 2,2,2,1,1 );
tal_r2c2_x2y1z1.parse( talErgPath,talPath );

ntally2 tal_r2c2_x2y1z2;
talPath = getTallyPath( 2,2,2,1,2 );
tal_r2c2_x2y1z2.parse( talErgPath,talPath );

ntally2 tal_r2c2_x2y2z1;
talPath = getTallyPath( 2,2,2,2,1 );
tal_r2c2_x2y2z1.parse( talErgPath,talPath );

ntally2 tal_r2c2_x2y2z2;
talPath = getTallyPath( 2,2,2,2,2 );
tal_r2c2_x2y2z2.parse( talErgPath,talPath );


ntally2 tal_r1c1_s1 = interpolateSourceXZ( tal_r1c1_x1y1z1,
    tal_r1c1_x1y1z2,tal_r1c1_x2y1z1,tal_r1c1_x2y1z2 );

ntally2 tal_r1c1_s2 = interpolateSourceXZ( tal_r1c1_x1y2z1,
    tal_r1c1_x1y2z2,tal_r1c1_x2y2z1,tal_r1c1_x2y2z2 );

ntally2 tal_r1c2_s1 = interpolateSourceXZ( tal_r1c2_x1y1z1,
    tal_r1c2_x1y1z2,tal_r1c2_x2y1z1,tal_r1c2_x2y1z2 );

ntally2 tal_r1c2_s2 = interpolateSourceXZ( tal_r1c2_x1y2z1,
    tal_r1c2_x1y2z2,tal_r1c2_x2y2z1,tal_r1c2_x2y2z2 );

ntally2 tal_r2c1_s1 = interpolateSourceXZ( tal_r2c1_x1y1z1,
    tal_r2c1_x1y1z2,tal_r2c1_x2y1z1,tal_r2c1_x2y1z2 );

ntally2 tal_r2c1_s2 = interpolateSourceXZ( tal_r2c1_x1y2z1,
    tal_r2c1_x1y2z2,tal_r2c1_x2y2z1,tal_r2c1_x2y2z2 );

ntally2 tal_r2c2_s1 = interpolateSourceXZ( tal_r2c2_x1y1z1,
    tal_r2c2_x1y1z2,tal_r2c2_x2y1z1,tal_r2c2_x2y1z2 );

ntally2 tal_r2c2_s2 = interpolateSourceXZ( tal_r2c2_x1y2z1,
    tal_r2c2_x1y2z2,tal_r2c2_x2y2z1,tal_r2c2_x2y2z2 );

ntally2 tal_d1s1 = interpolateDetectorRow( tal_r1c1_s1,
    tal_r2c1_s1 );

ntally2 tal_d1s2 = interpolateDetectorRow( tal_r1c1_s2,
    tal_r2c1_s2 );

ntally2 tal_d2s1 = interpolateDetectorRow( tal_r1c2_s1,
    tal_r2c2_s1 );

ntally2 tal_d2s2 = interpolateDetectorRow( tal_r1c2_s2,
    tal_r2c2_s2 );


ntally2 talds;
```

```
    if ( my_intrp == normal )
    {
        ntally2 tal_d1s = ntally2::interpolate( my_srcPtY[my_srcIdxY1],
            tal_d1s1,my_srcPtY[my_srcIdxY2],tal_d1s2,my_srcPosY );

        ntally2 tal_d2s = ntally2::interpolate( my_srcPtY[my_srcIdxY1],
            tal_d2s1,my_srcPtY[my_srcIdxY2],tal_d2s2,my_srcPosY );

        if ( my_detPosType == side )
        {
            talds = ntally2::interpolate( my_detSidePtY[my_detIdxCol1],
                tal_d1s,my_detSidePtY[my_detIdxCol2],tal_d2s,
                my_detPosY );
        }
        else /*if ( my_detPosType == top )*/
        {
            talds = ntally2::interpolate( my_detTopPtY[my_detIdxCol1],
                tal_d1s,my_detTopPtY[my_detIdxCol2],tal_d2s,
                my_detPosY );
        }
    }
    else /*if ( my_intrp == peak )*/
    {
        ntally2 tal_dsp = ntally2::interpolate( my_srcPtY[my_srcIdxY1],
            tal_d1s1,my_srcPtY[my_srcIdxY2],tal_d2s2,my_srcPosY );

        if ( my_srcPosY > my_detPosY )
        {
            if ( my_detPosType == side )
            {
                ntally2 tal_ds2 = ntally2::interpolate(
                    my_detSidePtY[my_detIdxCol1],tal_d1s2,
                    my_detSidePtY[my_detIdxCol2],tal_d2s2,
                    my_detPosY );

                talds = ntally2::interpolate( my_detPosY,tal_dsp,
                    my_srcPtY[my_srcIdxY2],tal_ds2,my_srcPosY );
            }
            else /*if ( my_detPosType == top )*/
            {
                ntally2 tal_ds2 = ntally2::interpolate(
                    my_detTopPtY[my_detIdxCol1],tal_d1s2,
                    my_detTopPtY[my_detIdxCol2],tal_d2s2,
                    my_detPosY );

                talds = ntally2::interpolate( my_detPosY,tal_dsp,
                    my_srcPtY[my_srcIdxY2],tal_ds2,my_srcPosY );
            }
        }
        else /*if ( my_srcPosY < my_detPosY )*/
        {
            if ( my_detPosType == side )
            {
                ntally2 tal_ds1 = ntally2::interpolate(
                    my_detSidePtY[my_detIdxCol1],tal_d1s1,
                    my_detSidePtY[my_detIdxCol2],tal_d2s1,
                    my_detPosY );

                talds = ntally2::interpolate( my_detPosY,tal_dsp,
                    my_srcPtY[my_srcIdxY1],tal_ds1,my_srcPosY );
            }
            else /*if ( my_detPosType == top )*/
            {
                ntally2 tal_ds1 = ntally2::interpolate(
                    my_detTopPtY[my_detIdxCol1],tal_d1s1,
                    my_detTopPtY[my_detIdxCol2],tal_d2s1,
                    my_detPosY );

                talds = ntally2::interpolate( my_detPosY,tal_dsp,
                    my_srcPtY[my_srcIdxY1],tal_ds1,my_srcPosY );
            }
        }
    }

    return talds;
}
```

```
void nvehicle::genResponse( )
{
    my_data = ntally2(0,0,0);
    //my_srcErg = readbin(my_datapath+"srcerg.dat");
    //const int Ein = static_cast<int>(my_srcErg.size());

    //my_data.resize(Ein);
    //for ( int e=0;e<Ein-1;++e )
    //{
        my_data = interpolateDetectorsAndSources(   );
    //}
    //assignDataToResponse( );


    const int T = my_data.nTime();
    const int Eout = my_data.nErg();
    const int Ein = my_data.nSrcErg();
    my_K.setTime( my_data.time() );
    for ( int t=0;t<T-1;++t )
    {
        my_K(t).initialize( my_data.srcErg(),my_data.erg() );
        for ( int ei=0;ei<Ein-1;++ei )
        {
            for ( int eo=0;eo<Eout-1;++eo )
            {
                my_K(t)(eo,ei) = my_data.val(ei,eo+1,t+1);
            }
        }
        //my_K(t).optimize();
    }
}

void nvehicle::getDetectorPlane( )
{
    // hard-code in detector planes, should
    // read this in from file eventually
    const double tside = 129.54;
    const double ttop = 259.08;
    const double soff = 68.56;
    my_detXPlane = tside+soff; // side of truck + standoff
    my_detZPlane = ttop+soff; // side of truck + standoff

    // find out if this detector is a "side" or "top" detector
    if ( my_detPosZ < my_detZPlane && my_detPosX > tside )
    {
        my_detPosType = side;
        if ( fabs( my_detXPlane-my_detPosX ) > 0.1 )
        {
            // in future, need to add 1/r^2 correction factor
            // and time correction factor
            throw fatal_error("side detector does not lie in x plane");
        }
    }
    else if ( my_detPosZ > ttop && my_detPosX < my_detXPlane )
    {
        my_detPosType = top;
        if ( fabs( my_detZPlane-my_detPosZ ) > 0.1 )
        {
            throw fatal_error("top detector does not lie in z plane");
        }
    }
    else
    {
        throw fatal_error("invalid detector position");
    }
}


void nvehicle::getDetectorIndices( )
{
    my_detIdxRow1 = 0;
    my_detIdxRow2 = 0;
    my_detIdxCol1 = 0;
    my_detIdxCol2 = 0;
```

```
    if ( my_detPosType == side )
    {
        const int Y = static_cast<int>( my_detSidePtY.size() );
        for ( int i=0;i<Y-1;++i )
        {
            if ( my_detPosY > my_detSidePtY[i]
                && my_detPosY < my_detSidePtY[i+1] )
            {
                my_detIdxCol1 = i;
                my_detIdxCol2 = i+1;
                break;
            }
        }
        const int Z = static_cast<int>( my_detSidePtZ.size() );
        for ( int i=0;i<Z-1;++i )
        {
            if ( my_detPosZ > my_detSidePtZ[i]
                && my_detPosZ < my_detSidePtZ[i+1] )
            {
                my_detIdxRow1 = i;
                my_detIdxRow2 = i+1;
                break;
            }
        }
    }
    else if ( my_detPosType == top )
    {
        const int Y = static_cast<int>( my_detTopPtY.size() );
        for ( int i=0;i<Y-1;++i )
        {
            if ( my_detPosY > my_detTopPtY[i]
                && my_detPosY < my_detTopPtY[i+1] )
            {
                my_detIdxCol1 = i;
                my_detIdxCol2 = i+1;
                break;
            }
        }
        const int X = static_cast<int>( my_detTopPtX.size() );
        for ( int i=0;i<X-1;++i )
        {
            if ( my_detPosX > my_detTopPtX[i]
                && my_detPosX < my_detTopPtX[i+1] )
            {
                my_detIdxRow1 = i;
                my_detIdxRow2 = i+1;
                break;
            }
        }
    }
}


void nvehicle::buildResponse ( double srcXPos,double srcYPos,
    double srcZPos,double detXPos,double detYPos,double detZPos )
{
    //std::cout << "building nvehicle response" << std::endl;

    // store detector positions
    my_detPosX = detXPos;
    my_detPosY = detYPos;
    my_detPosZ = detZPos;
    getDetectorPlane( );

    // record source position
    my_srcPosX = srcXPos;
    my_srcPosY = srcYPos;
    my_srcPosZ = srcZPos;

    // calculate detector-to-source distance (unused right now)
    //my_det2SrcDistance = sqrt( pow(my_detPosX-my_srcPosX,2)
    // + pow(my_detPosY-my_srcPosY,2) + pow(my_detPosZ-my_srcPosZ,2) );

    getDetectorIndices( );

    getSourceIndices( );
```

```
    // get detector indices which match source positions in y-direction
    getDetectorSourceIndices( );



    // determine interpolation type
    //
    // if the detector position is bounded by the
    // closest source positions, use peak method
    if ( my_detPosY < my_srcPtY[my_srcIdxY2]
        && my_detPosY > my_srcPtY[my_srcIdxY1] )
    {
        my_intrp = peak;
    }
    else
    {
        my_intrp = normal;
    }

    // build response function
    genResponse( );

    //my_I.setTime( my_K.getTime() );
    //my_I.setErg( my_K.getErg(),my_K.getErg() );
    //std::vector<double> d2s(3);
    //d2s[0] = srcXPos - detXPos;
    //d2s[1] = srcYPos - detYPos;
    //d2s[2] = srcZPos - detZPos;
    //my_I.shiftDistance(mag(d2s));
    //my_I = my_I * (1.0/(4*phys::pi*pow(mag(d2s),2.0)));
    //my_K = my_K * (1.0-omegaStream) + my_I
    // * ( (1.0/(4*phys::pi*pow(mag(d2s),2.0))) * omegaStream );

    //std::cout << "finished building nvehicle response" << std::endl;

    return;
}

void nvehicle::buildNoVehicle( double srcXPos,
                               double srcYPos,
                               double srcZPos,
                               double detXPos,
                               double detYPos,
                               double detZPos,
                               const tspectrum& S )
{
    //std::cout << "building no vehicle response" << std::endl;

    my_K.setTime( S.getTime() );
    my_K.setErg( S.getErg(),S.getErg() );

    std::vector<double> d2s(3);
    d2s[0] = srcXPos - detXPos;
    d2s[1] = srcYPos - detYPos;
    d2s[2] = srcZPos - detZPos;
    my_K.shiftDistance(mag(d2s));

    my_K = my_K * (1.0/(4*phys::pi*pow(mag(d2s),2.0)));

    //std::cout << "finished building no vehicle response" << std::endl;

    return;
}


void nvehicle::getSourceIndices( )
{
    my_srcIdxX1 = 0;
    my_srcIdxX2 = 0;
    my_srcIdxY1 = 0;
    my_srcIdxY2 = 0;
    my_srcIdxZ1 = 0;
    my_srcIdxZ2 = 0;

    const int X = static_cast<int>( my_srcPtX.size() );
```

```
    for ( int i=0;i<X-1;++i )
    {
        if ( my_srcPosX >= my_srcPtX[i] && my_srcPosX < my_srcPtX[i+1] )
        {
            my_srcIdxX1 = i;
            my_srcIdxX2 = i+1;
            break;
        }
    }
    const int Y = static_cast<int>( my_srcPtY.size() );
    for ( int i=0;i<Y-1;++i )
    {
        if ( my_srcPosY >= my_srcPtY[i] && my_srcPosY < my_srcPtY[i+1] )
        {
            my_srcIdxY1 = i;
            my_srcIdxY2 = i+1;
            break;
        }
    }
    const int Z = static_cast<int>( my_srcPtZ.size() );
    for ( int i=0;i<Z-1;++i )
    {
        if ( my_srcPosZ >= my_srcPtZ[i] && my_srcPosZ < my_srcPtZ[i+1] )
        {
            my_srcIdxZ1 = i;
            my_srcIdxZ2 = i+1;
            break;
        }
    }


}

void nvehicle::getDetectorSourceIndices( )
{
    // if we need to worry about the peak interpolation algorithm, use
    // only detector indices which line up with source
    if ( my_intrp == peak )
    {
        if ( my_detPosType == side )
        {
            const int Y = static_cast<int>( my_detSidePtY.size() );
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detSidePtY[i]
                    - my_srcPtY[my_srcIdxY1] ) < 0.1 )
                {
                    my_detIdxCol1 = i;
                    break;
                }
            }
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detSidePtY[i]
                    - my_srcPtY[my_srcIdxY2] ) < 0.1 )
                {
                    my_detIdxCol2 = i;
                    break;
                }
            }
        }
        else if ( my_detPosType == top )
        {
            const int Y = static_cast<int>( my_detTopPtY.size() );
            for ( int i=0;i<Y;++i )
            {
                if ( fabs( my_detTopPtY[i]
                    - my_srcPtY[my_srcIdxY1] ) < 0.1 )
                {
                    my_detIdxCol1 = i;
                    break;
                }
            }
            for ( int i=0;i<Y;++i )
            {
```

```
                if ( fabs( my_detTopPtY[i]
                    - my_srcPtY[my_srcIdxY2] ) < 0.1 )
                {
                    my_detIdxCol2 = i;
                    break;
                }
            }
        }
    }
}

tspectrum nvehicle::operator() ( const tspectrum& S )
{
    return my_K * S;
}
```

Listing B.38: nvehicle.hpp

```
#ifndef _nvehicle_hpp_included_
#define _nvehicle_hpp_included_

#include "nsubmodel.hpp"

class nvehicle : public nsubmodel
{
    public:

    enum dpos { side,top };
    enum interpolationType { peak,normal };

    nvehicle( ) { };
    nvehicle( const std::string& path );

    void initialize( const std::string& path );

    void buildResponse ( double srcXPos,double srcYPos,
        double srcZPos,double detXPos,double detYPos,
        double detZPos );

    void buildNoVehicle( double srcXPos,double srcYPos,
        double srcZPos,double detXPos,double detYPos,
        double detZPos,const tspectrum& S );

    tspectrum operator() ( const tspectrum& S );

    void addResponse( const tresponse& K,double frac )
    {
        my_K = my_K + K*frac;
    }

    private:

    tresponse my_I;

    void readDataFile();
    std::string getTallyEnergyPath(  );
    std::string getTallyPath( int detRowIdx,
        int detColIdx,int srcIdxX,int srcIdxY,int srcIdxZ );

    ntally2 interpolateSourceXZ( const ntally2& talx1z1,
        const ntally2& talx1z2,const ntally2& talx2z1,
        const ntally2& talx2z2 );

    ntally2 interpolateDetectorRow( const ntally2& talRow1,
        const ntally2& talRow2 );

    ntally2 interpolateDetectorsAndSources( );

    void genResponse( );

    void getDetectorPlane( );
    void getDetectorIndices( );
    void getSourceIndices( );
    void getDetectorSourceIndices( );


    double my_srcPosX;
    double my_srcPosY;
    double my_srcPosZ;

    double my_det2SrcDistance;

    std::vector< double > my_srcPtX;
    std::vector< double > my_srcPtY;
    std::vector< double > my_srcPtZ;

    double my_detPosX;
    double my_detPosY;
    double my_detPosZ;

    double my_detXPlane;
    double my_detZPlane;
```

```
    std::vector< double > my_detSidePtX;
    std::vector< double > my_detSidePtY;
    std::vector< double > my_detSidePtZ;
    std::vector< double > my_detSideArea;
    std::vector< double > my_detTopPtX;
    std::vector< double > my_detTopPtY;
    std::vector< double > my_detTopPtZ;
    std::vector< double > my_detTopArea;
    int my_detIdxRow1;
    int my_detIdxRow2;
    int my_detIdxCol1;
    int my_detIdxCol2;

    std::vector<int> my_detPosIdx;
    std::vector<double> my_detDistance;

    dpos my_detPosType;
    interpolationType my_intrp;


    ntally2 my_data;



    int my_srcIdxX1;
    int my_srcIdxX2;
    int my_srcIdxY1;
    int my_srcIdxY2;
    int my_srcIdxZ1;
    int my_srcIdxZ2;
};

#endif
```

Listing B.39: paths.cpp

```
#include "paths.hpp"
#include "extras.hpp"
#include "errh.hpp"
#include "fileio.hpp"

namespace input
{

    std::string paths::getPath( std::ifstream& in,const std::string& name )
    {
        Find ( in,name,true );
        std::string result;
        in >> result;
        if ( ! exists(result) )
        {
            throw fatal_error ( "required file/directory " + result
                                + " does not exist" );
        }
        return result;
    }

    void paths::parse ( )
    {
        using namespace std;

        // open data paths
        //
        ifstream in( "paths" );
        if ( ! in.good() )
        {
            throw fatal_error ( "error opening path file" );
        }

        radsrchome = getPath(in,"radsrchome");
        radsrcdata = getPath(in,"radsrcdata");

        psnm = getPath(in,"psnm");
        pshield = getPath(in,"pshield");
        pcargo = getPath(in,"pcargo");
        pbg = getPath(in,"pbg");
        pdet = getPath(in,"pdet");

        truckdim = getPath(in,"truckdim");

        uranium_soil = getPath(in,"uranium_soil");
        uranium_concrete = getPath(in,"uranium_concrete");
        thorium_soil = getPath(in,"thorium_soil");
        thorium_concrete = getPath(in,"thorium_concrete");
        potassium_soil = getPath(in,"potassium_soil");
        potassium_concrete = getPath(in,"potassium_concrete");

        nsnm = getPath(in,"nsnm");
        nshield = getPath(in,"nshield");
        ncargo = getPath(in,"ncargo");
        ndet = getPath(in,"ndet");
        sfa = getPath(in,"sfa");
        nbg = getPath(in,"nbg");

        norm = getPath(in,"norm");
        k40norm = getPath(in,"k40norm");
        u238norm = getPath(in,"u238norm");
        th232norm = getPath(in,"th232norm");
        ra226norm = getPath(in,"ra226norm");

        alarmtemp = getPath(in,"alarmtemp");

        savedir = getPath(in,"savedir");
    }

}
```

Listing B.40: paths.hpp

```
#ifndef _paths_hpp_included_
#define _paths_hpp_included_

#include <vector>
#include <string>
#include <fstream>

namespace input
{

class paths
{
    public:

    std::string radsrchome;
    std::string radsrcdata;

    std::string psnm;
    std::string pshield;
    std::string pcargo;
    std::string pdet;
    std::string pbg;

    std::string truckdim;

    std::string uranium_soil;
    std::string uranium_concrete;
    std::string thorium_soil;
    std::string thorium_concrete;
    std::string potassium_soil;
    std::string potassium_concrete;

    std::string norm;
    std::string k40norm;
    std::string u238norm;
    std::string th232norm;
    std::string ra226norm;


    std::string nsnm;
    std::string nshield;
    std::string ncargo;
    std::string ndet;
    std::string sfa;
    std::string nbg;

    std::string alarmtemp;

    std::string savedir;

    void parse ( );

    private:

    std::string getPath( std::ifstream& in,const std::string& name );

};


}
#endif
```

Listing B.41: response.cpp

```
#include "response.hpp"
#include <limits>
#include <iostream>
#include <iomanip>

int response::numergin ( )
{
    return static_cast<int>(my_energy_in.size());
}

int response::numergin ( ) const
{
    return static_cast<int>(my_energy_in.size());
}

int response::numergout ( )
{
    return static_cast<int>(my_energy_out.size());
}

int response::numergout ( ) const
{
    return static_cast<int>(my_energy_out.size());
}

std::vector < double >& response::ergin ( )
{
    return my_energy_in;
}

std::vector < double > response::ergin ( ) const
{
    return my_energy_in;
}

std::vector < double >& response::ergout ( )
{
    return my_energy_out;
}

std::vector < double > response::ergout ( ) const
{
    return my_energy_out;
}

double& response::ergin ( int a )
{
    return my_energy_in[a];
}

double response::ergin ( int a ) const
{
    return my_energy_in[a];
}

double& response::ergout ( int a )
{
    return my_energy_out[a];
}

double response::ergout ( int a ) const
{
    return my_energy_out[a];
}

matrix < datapoint >& response::data ()
{
    return my_data;
}

matrix < datapoint > response::data () const
{
    return my_data;
}
```

```
datapoint& response::data ( int i,int j )
{
    return my_data ( i,j );
}

datapoint response::data ( int i,int j ) const
{
    return my_data ( i,j );
}

datapoint& response::operator() ( int i,int j )
{
    return my_data ( i,j );
}

datapoint response::operator() ( int i,int j ) const
{
    return my_data ( i,j );
}

void response::identity ( )
{
    matrix < double > I = genTransform ( my_energy_in,my_energy_out );

    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            my_data ( i,j ).set ( I ( i,j ) );
            my_data ( i,j ).setErr ( 0.0 );
        }
    }
}

response response::inverse ( )
{
    response result( ergin(),ergout() );
    result.data() = my_data.inverse();
    return result;
}

void response::resize ( int numergout,int numergin )
{
    my_data.resize ( numergout-1,numergin-1 );
    my_energy_in.resize ( numergin );
    my_energy_out.resize ( numergout );
    my_T = respPtr( new response );
    return;
}

void response::initialize ( const std::vector<double>& energyin,
    const std::vector<double>& energyout )
{
    my_energy_in = energyin;
    my_energy_out = energyout;
    const int Eout = static_cast<int> ( energyout.size() );
    const int Ein = static_cast<int> ( energyin.size() );
    if ( Eout > 0 && Ein > 0 )
    {
        my_data.resize ( Eout-1,Ein-1 );
    }
    else
    {
        my_energy_in.resize(0);
        my_energy_out.resize(0);
    }
    my_T = respPtr( new response );
    return;
}

response::response( )
{

}

response::response ( int numergout,int numergin )
```

```
{
    resize ( numergout,numergin );
    my_T = respPtr( new response );
}

response::response ( const std::vector<double>& energyin,
    const std::vector<double>& energyout )
{
    initialize ( energyin,energyout );
}


void response::operator= ( const response& a )
{
    my_energy_in = a.ergin();
    my_energy_out = a.ergout();
    my_data = a.data();
    my_T = respPtr( new response );
    return;
}

void response::operator= ( double a )
{
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            data ( i,j ).set ( a );
            data ( i,j ).setErr ( 0.0 );
        }
    }
    return;
}

void response::operator= ( const datapoint& a )
{
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            data ( i,j ) = a;
        }
    }
    return;
}

spectrum operator * ( const response& resp,const spectrum& spec )
{
    if ( spec.numerg() == 0 )
    {
        return spec;
    }

    //typedef std::tr1::shared_ptr<response> respPtr;
    //bool reuseT = true;
    //if ( resp.getT()->numergout() == 0
    //   || resp.getT()->numergin() != spec.numerg() )
    //{
        //reuseT = false;
    //}
    //else
    //{
        //for ( int i=0;i<resp.getT()->numergin();++i )
        //{
            //if ( fabs(resp.getT()->ergin(i)-spec.erg(i)) > 1e-5 )
            //{
                //reuseT = false;
            //}
            //break;
        //}
    //}

    //if ( ! reuseT )
    //{
        //resp.getT()->initialize( spec.erg(),resp.ergin() );
        //resp.getT()->identity();
```

```
        //resp.getT()->optimize();
    //}

    response T(spec.erg(),resp.ergin());
    T.identity();


    spectrum result;
    //result.data() = resp.data() * ( resp.getT()->data() * spec.data() );
    result.data() = resp.data() * ( T.data() * spec.data() );
    //result.data() = resp.data() * spec.data();
    result.erg() = resp.ergout();

    return result;
}


spectrum operator * ( const response& resp,const dspectrum& spec )
{
    //std::cout << "in spectrum operator * " << std::endl;
    spectrum result ( resp.numergout() );
    result.erg() = resp.ergout();

    int startj = 0; // keep track of last response energy index used
    // index i is the discrete spectrum energy index
    for ( int i=0;i<spec.numerg();++i )
    {
        // search for appropriate bin in response function

        // make sure it's within the bounds
        // of the response function energies
        if ( spec.erg(i) >= resp.ergin(0)
            && spec.erg(i) <= resp.ergin(resp.numergin()-1) )
        {
            // iterate through all energies in
            // the response function by index j
            for ( int j=startj;j<resp.numergin()-1;++j )
            {
                // if the discrete energy falls within this bin (i know
                // this looks weird, but there are N data points and N+1
                // energy bins, so we want j -> j+1 bin instead of the
                // "normal" j-1 -> j scheme)
                if ( spec.erg(i) > resp.ergin(j)
                    && spec.erg(i) <= resp.ergin(j+1) )
                {
                    if ( spec.isSorted() )
                    {
                        // then we know this j index will be
                        // a good starting point for the next energy
                        startj = j;
                        // otherwise, just keep startj at 0
                    }
                    for ( int k=0;k<resp.numergout()-1;++k )
                    {
                        result(k) = result(k) + resp(k,j)*spec(i);
                    }
                }
            }
        }
    }
    //std::cout << "done in spectrum operator * " << std::endl;
    return result;
}


response operator + ( const response& a,const response& b )
{
    response result ( a.ergin(),a.ergout() );
    result.data() = a.data() + b.data();
    result.ergin() = a.ergin();
    result.ergout() = a.ergout();

    return result;
}

response operator - ( const response& a,const response& b )
```

```
{
    response result ( a.ergin(),a.ergout() );
    result.data() = a.data() - b.data();
    result.ergin() = a.ergin();
    result.ergout() = a.ergout();

    return result;
}

response operator * ( const response& a,const response& b )
{
    response result ( a.ergin(),a.ergout() );
    result.data() = a.data() * b.data();
    result.ergin() = a.ergin();
    result.ergout() = a.ergout();

    return result;
}

response operator * ( const response& resp,double a )
{
    response result ( resp.ergin(),resp.ergout() );
    result.data() = resp.data() * a;
    result.ergin() = resp.ergin();
    result.ergout() = resp.ergout();

    return result;
}

response operator * ( double a,const response& resp )
{
    return resp*a;
}

response operator / ( const response& resp,double a )
{
    response result ( resp.ergin(),resp.ergout() );
    result.data() = resp.data() / a;
    result.ergin() = resp.ergin();
    result.ergout() = resp.ergout();

    return result;
}


void response::print ( )
{
    std::cout << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 3 );
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            std::cout << std::setw ( 10 ) << my_data ( i,j ).get();
        }
        std::cout << std::endl;
    }
    return;
}

void response::printErr ( )
{
    std::cout << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 3 );
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            std::cout << std::setw ( 10 ) << my_data ( i,j ).getErr();
        }
        std::cout << std::endl;
    }
    return;
}

double response::sum ( )
```

```
{
    double result = 0.0;
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            result = result + my_data ( i,j ).get();
        }
    }
    return result;
}

void response::checkData ( )
{
    std::cout << std::resetiosflags( std::ios::scientific );
    int sumInf = 0;
    int sumZero = 0;
    int sumNan = 0;
    int sumNeg = 0;
    for ( int i=0;i<numergout()-1;++i )
    {
        for ( int j=0;j<numergin()-1;++j )
        {
            if ( isinf(my_data(i,j).get()) )
            {
                sumInf++;
            }
            else if ( isnan(my_data(i,j).get()) )
            {
                sumNan++;
            }
            else if ( my_data(i,j).get() < 0.0 )
            {
                sumNeg++;
            }
            else if ( my_data(i,j).get() < 1e-30 )
            {
                sumZero++;
            }
        }
    }
    const double tot = (numergout()-1)*(numergin()-1);
    std::cout << ((double)sumInf)/tot*100.0 << " % inf" << std::endl;
    std::cout << ((double)sumNan)/tot*100.0 << " % nan" << std::endl;
    std::cout << ((double)sumNeg)/tot*100.0 << " % negative" << std::endl;
    std::cout << ((double)sumZero)/tot*100.0 << " % zero" << std::endl;
}


void response::optimize( )
{
    my_data.optimize();
}
```

Listing B.42: response.hpp

```
#ifndef _response_hpp_included_
#define _response_hpp_included_
#include <vector>
#include <tr1/memory>
#include "errh.hpp"
#include "datapoint.hpp"
#include "spectrum.hpp"
#include "dspectrum.hpp"
#include "matrix.hpp"

class response
{
    public:

    typedef std::tr1::shared_ptr<response> respPtr;

    // return number of incoming energies
    int numergin ( );
    int numergin ( ) const;
    // return number of outgoing energies
    int numergout ( );
    int numergout ( ) const;

    std::vector < double >& ergin ( );
    std::vector < double > ergin ( ) const;
    std::vector < double >& ergout ( );
    std::vector < double > ergout ( ) const;

    double& ergin ( int );
    double ergin ( int ) const;
    double& ergout ( int );
    double ergout ( int ) const;

    matrix < datapoint >& data ();
    matrix < datapoint > data () const;

    datapoint& data ( int,int );
    datapoint data ( int,int ) const;

    void identity();

    response inverse();

    void resize ( int,int );

    void initialize ( const std::vector<double>& energyin,
        const std::vector<double>& energyout );

    datapoint& operator() ( int,int );
    datapoint operator() ( int,int ) const;

    void operator= ( const response& );
    void operator= ( double );
    void operator= ( const datapoint& );

    response ( );
    response ( int,int );
    response ( const std::vector<double>&,const std::vector<double>& );

    void print ( );
    void printErr ( );

    void clear() { my_data.clear(); };

    void scaleBy( double scaler )
    {
        my_data *= scaler;
    }

    double sum ( );

    void checkData( );

    void optimize( );
```

```
    respPtr getT( ) const { return my_T; };

    protected:

    std::vector < double > my_energy_in;
    std::vector < double > my_energy_out;

    respPtr my_T;

    private:

    matrix < datapoint > my_data;


};

spectrum operator * ( const response&,const spectrum& );

spectrum operator * ( const response&,const dspectrum& );

response operator + ( const response&,const response& );

response operator - ( const response&,const response& );

response operator * ( const response&,const response& );

response operator * ( const response&,double );

response operator * ( double,const response& );

response operator / ( const response&,double );

#endif
```

Listing B.43: shield.cpp

```
#include "shield.hpp"
#include <fstream>
#include <math.h>
#include <iostream>
#include <iomanip>
#include <limits>
#include "errh.hpp"
#include "fileio.hpp"
#include "extras.hpp"
#include "interpolation.hpp"
#include "phys.hpp"

double quadsolve( double A,double B,double C )
{
    const double x1 = (-B + sqrt(B*B-4*A*C))/(2*A);
    const double x2 = (-B - sqrt(B*B-4*A*C))/(2*A);
    if ( x1 > x2 )
    {
        return x1;
    }
    else
    {
        return x2;
    }
}

double chordLength( double r,double R,double theta )
{
    const double A = 1;
    const double B = -2*r*cos(theta);
    const double C = r*r-R*R;
    return quadsolve(A,B,C);
}

double aveLength( double sigma,double distance )
{
    const double x = sigma;
    const double d = distance;
    return (d*exp(-x*d)+1/x*exp(-x*d)-1/x)/(exp(-x*d)-1);
}

double scatprob( double Eo,double mu1,double mu2 )
{
    std::vector<double> cosb = linspace(mu1,mu2,10);
    std::vector<double> cost = linspace(-1.0,1.0,10);

    double prob = 0.0;
    for ( unsigned int c=0;c<cosb.size()-1;++c )
    {
        const double sig1 = phys::sigkn( Eo,cosb[c] )/phys::re;
        const double sig2 = phys::sigkn( Eo,cosb[c+1] )/phys::re;
        prob += (cosb[c+1]-cosb[c])*(1.0/2.0)*(sig1+sig2);
    }

    double tot = 0.0;
    for ( unsigned int c=0;c<cost.size()-1;++c )
    {
        const double sig1 = phys::sigkn( Eo,cost[c] )/phys::re;
        const double sig2 = phys::sigkn( Eo,cost[c+1] )/phys::re;
        tot += (cost[c+1]-cost[c])*(1.0/2.0)*(sig1+sig2);
    }
    return prob/tot;
}

double scaterg( double Eo,double mu1,double mu2 )
{
    std::vector<double> cosb = linspace(mu1,mu2,10);
    double Ebar = 0.0;
    for ( unsigned int c=0;c<cosb.size()-1;++c )
    {
        const double Ep1 = phys::scaterg( Eo,acos(cosb[c]) );
        const double Ep2 = phys::scaterg( Eo,acos(cosb[c+1]) );
        Ebar += (cosb[c+1]-cosb[c])*(1.0/2.0)*(Ep1+Ep2);
    }
    return Ebar/(mu2-mu1);
```

```
}

double shield::getEscapeProbability( double ri,
                              double r,
                              double ro,
                              double beta,
                              double xbar,
                              double Eo )
{
    const double pi = phys::pi;
    // iterate over all possible scattering angles and compute
    // probability of scattering in that angle times the probability
    // of escaping unscathed
    double p = 0.0;

    // possible angles
    std::vector<double> mu = linspace(-1.0,1.0,10);
    const int M = static_cast<int>(mu.size()) -1;

    // hard code in lead cross section
    //photonxsec xsec( my_Z,my_frac,my_rho );

    for ( int m=0;m<M;++m )
    {
        // calculate actual angles
        const double theta1 = acos(mu[m]);
        const double theta2 = acos(mu[m+1]);
        // calculate average chord length in this angle interval
        const double chord1 = chordLength( r,ro,pi-fabs(beta-theta1) );
        const double chord2 = chordLength( r,ro,pi-fabs(beta-theta2) );
        const double chord = (chord1+chord2)/2.0;
        // calculate probability of scattering into this angle bin
        const double pscat = scatprob( Eo,mu[m],mu[m+1] );
        // calculate average energy leaving this scatter into this angle
        const double Ep = scaterg( Eo,mu[m],mu[m+1] );
        // get total cross section for energy leaving
        const double Sigmat = my_xsec.tot(Ep);
        // probability of not interacting
        const double Pni = exp(-Sigmat*chord);
        // calculate angles which go back into inner sphere
        const double phi = asin( ri/r );
        const double alpha = acos( (xbar*xbar+r*r-ri*ri)/(2*xbar*r) );
        const double phi1 = phi+alpha;
        const double phi2 = phi-alpha;
        // if exiting angle is not going back into inner sphere
        // add it
        // need two angles because there is not azimuthal symmetry, half
        // for each pi/2pi azimuth
        //if ( theta1 < pi-phi1 && theta2 < pi-phi1 )
        //{
            //p = p + (1/2) * pscat * (1-Pi);
        //}
        //if ( theta1 < pi-phi2 && theta2 < pi-phi2 )
        //{
            //p = p + (1/2) * pscat * (1-Pi);
        //}

        // simple version
        p = p + pscat * Pni;
    }
    return p;
}


// this is for pair production
double shield::getEscapeProbability( double ri,
    double r,
    double ro,
    double xbar,
    double Eo )
{
    const double pi = phys::pi;
    // iterate over all possible scattering angles and compute
    // probability of scattering in that angle times the probability
    // of escaping unscathed
    double p = 0.0;
```

```
    // possible angles
    std::vector<double> mu = linspace(-1.0,1.0,10);
    const int M = static_cast<int>(mu.size()) -1;

    // hard code in lead cross section
    //photonxsec xsec( my_Z,my_frac,my_rho );

    for ( int m=0;m<M;++m )
    {
        // calculate actual angles
        const double theta1 = acos(mu[m]);
        const double theta2 = acos(mu[m+1]);
        // calculate average chord length in this angle interval
        // two sets of chord lengths, one for each direction
        const double chord11 = chordLength( r,ro,pi-theta1 );
        const double chord12 = chordLength( r,ro,pi-theta2 );
        const double chord1 = (chord11+chord12)/2.0;
        const double chord21 = chordLength( r,ro,theta1 );
        const double chord22 = chordLength( r,ro,theta2 );
        const double chord2 = (chord21+chord22)/2.0;
        // calculate probability of scattering into this angle bin
        // annihilation is isotropic
        const double pscat = (mu[m+1]-mu[m])/2.0;
        // annihilation photons are always 511 keV
        const double Ep = phys::me;
        // get total cross section for energy leaving
        const double Sigmat = my_xsec.tot(Ep);
        // probability of not interacting
        const double Pni1 = exp(-Sigmat*chord1);
        const double Pni2 = exp(-Sigmat*chord2);
        // calculate angles which go back into inner sphere
        const double phi = asin( ri/r );
        const double alpha = acos( (xbar*xbar+r*r-ri*ri)/(2*xbar*r) );
        const double phi1 = phi+alpha;
        const double phi2 = phi-alpha;
        // if exiting angle is not going back into inner sphere
        // add it
        // need two angles because there is not azimuthal symmetry, half
        // for each pi/2pi azimuth
        //if ( theta1 < pi-phi1 && theta2 < pi-phi1 )
        //{
            //p = p + (1/2) * pscat * ((1-Pi1)+(1-Pi2));
        //}
        //if ( theta1 < pi-phi2 && theta2 < pi-phi2 )
        //{
            //p = p + (1/2) * pscat * ((1-Pi1)+(1-Pi2));
        //}

        // simple version
        p = p + pscat * (Pni1+Pni2);
    }
    return p;
}

double chordLength2( double r,double R,double mu )
{
    return sqrt(R*R-r*r*(1-mu*mu))-r*mu;
}

double cosineChord( double r,double R,double c )
{
    return (c*c+R*R-r*r)/(2*c*R);
}

std::vector<double> shield::angularDistribution(
    double ri,double ro,double E )
{
    //photonxsec xsec(Z,my_frac,rho);
    const double sigmat = my_xsec.tot(E);
    const double sigmaa = my_xsec.abs(E);
    //std::cout << "sigmat = " << sigmat << std::endl;
    //const double sigmat = 0.0;

    std::vector<double> bulk = linspace(0.0,0.9*ri,50);
    std::vector<double> rlastmile = ri-ri*logspace(1e-15,1e-1,50);
```

```
std::vector<double> lastmile(rlastmile.size());
for (unsigned int i=0;i<rlastmile.size();++i )
{
    lastmile[i] = rlastmile[rlastmile.size()-i-1];
}
std::vector<double> r(bulk.size()+lastmile.size()+1);
for (unsigned int i=0;i<bulk.size();++i )
{
    r[i] = bulk[i];
}
for (unsigned int i=0;i<lastmile.size();++i )
{
    r[bulk.size()+i] = lastmile[i];
}
r.back() = ri;
std::vector<double> a = linspace(-1.0,1.0,100);
std::vector<double> mu = linspace(0.0,1.0,40);
std::vector<double> result(mu.size());

for ( unsigned int m=0;m<r.size()-1;++m )
{
    for ( unsigned int n=0;n<a.size()-1;++n )
    {
        const double dr1a1 = chordLength2(r[m],ri,a[n]);
        const double dr1a2 = chordLength2(r[m],ri,a[n+1]);
        const double dr2a1 = chordLength2(r[m+1],ri,a[n]);
        const double dr2a2 = chordLength2(r[m+1],ri,a[n+1]);

        double mur1a1 = sqrt(1.0
            - pow((r[m]/ri),2.0)*(1.0 -pow(a[n],2.0)));
        double mur1a2 = sqrt(1.0
            - pow((r[m]/ri),2.0)*(1.0 -pow(a[n+1],2.0)));
        double mur2a1 = sqrt(1.0
            - pow((r[m+1]/ri),2.0)*(1.0 -pow(a[n],2.0)));
        double mur2a2 = sqrt(1.0
            - pow((r[m+1]/ri),2.0)*(1.0 -pow(a[n+1],2.0)));

        // probability of non-interaction
        const double Pni = exp(-sigmat*dr1a1);
        // probability of interaction
        const double Pi = 1.0 - exp(-sigmat*dr1a1);
        // probability of absorption survival
        const double Pas = 1.0 - sigmaa/sigmat;

        // probability of escape
        const double Pe = (Pni+Pi*Pas);

        const double pr1a1 = (1.0/2.0)
            *(3.0*pow(r[m],2.0)/pow(ri,3.0))*Pe;
        const double pr1a2 = (1.0/2.0)
            *(3.0*pow(r[m],2.0)/pow(ri,3.0))*Pe;
        const double pr2a1 = (1.0/2.0)
            *(3.0*pow(r[m+1],2.0)/pow(ri,3.0))*Pe;
        const double pr2a2 = (1.0/2.0)
            *(3.0*pow(r[m+1],2.0)/pow(ri,3.0))*Pe;

        const double dalpha = a[n+1]-a[n];
        const double dr = r[m+1]-r[m];

        double f = (mur1a1+mur1a2+mur2a1+mur2a2)/4.0;
        //double f = mur2a2;
        const double angle = f;
        f = dr*( dalpha*(pr1a1+pr1a2)/2.0+dalpha
            *(pr2a1+pr2a2)/2.0   )/2.0;
        //f = pr2a2;
        const double prob = f;

        double mu1 = mur1a1;
        double mu2 = mur1a2;
        if ( mu1 > mu2 )
        {
            swap(mu1,mu2);
        }
        if ( mu1 != mu2 )
        {
            for ( unsigned int q=1;q<result.size();++q )
```

```
    {
        if ( mu1 < mu[q] )
        {
            double start = mu1;
            for ( unsigned y=q;y<result.size();++y )
            {
                if ( mu2 < mu[y] )
                {
                    result[y] += prob*(mu2-start)
                        /(mu2-mu1);
                    break;
                }
                else
                {
                    result[y] += prob*(mu[y]-start)
                        /(mu2-mu1);
                    start = mu[y];
                }
            }
            break;
        }

    }
}
else
{
    for ( unsigned int q=1;q<result.size();++q )
    {
        if ( mu1 >= mu[q-1] && mu1 <= mu[q] )
        {
            result[q] += prob;
            break;
        }
    }
}


mu1 = mur2a1;
mu2 = mur2a2;
if ( mu1 > mu2 )
{
    swap(mu1,mu2);
}
if ( mu1 != mu2 )
{
    for ( unsigned int q=1;q<result.size();++q )
    {
        if ( mu1 < mu[q] )
        {
            double start = mu1;
            for ( unsigned y=q;y<result.size();++y )
            {
                if ( mu2 < mu[y] )
                {
                    result[y] += prob*(mu2-start)
                        /(mu2-mu1);
                    break;
                }
                else
                {
                    result[y] += prob*(mu[y]-start)
                        /(mu2-mu1);
                    start = mu[y];
                }
            }
            break;
        }
    }
}
else
{
    for ( unsigned int q=1;q<result.size();++q )
    {
        if ( mu1 >= mu[q-1] && mu1 <= mu[q] )
        {
            result[q] += prob;
```

```
                        break;
                    }
                }
            }
        }
    }
    result = normalize(result);
    return result;
}


double shield::effectiveThickness( double ri,double ro,double E )
{
    //photonxsec xsec(Z,rho);
    const double sigmat = my_xsec.tot(E);

    std::vector<double> bulk = linspace(0.0,0.9*ri,10);
    std::vector<double> rlastmile = ri-logspace(1e-15,1e-1,50);
    std::vector<double> lastmile(rlastmile.size());
    for (unsigned int i=0;i<rlastmile.size();++i )
    {
        lastmile[i] = rlastmile[rlastmile.size()-i-1];
    }
    std::vector<double> r(bulk.size()+lastmile.size());
    for (unsigned int i=0;i<bulk.size();++i )
    {
        r[i] = bulk[i];
    }
    for (unsigned int i=0;i<lastmile.size();++i )
    {
        r[bulk.size()+i] = lastmile[i];
    }
    std::vector<double> mu = linspace(-1.0,1.0,100);
    double numer = 0.0;
    double denom = 0.0;
    for ( unsigned int m=0;m<r.size()-1;++m )
    {
        double r1top = 0.0;
        double r1bot = 0.0;
        for ( unsigned int n=0;n<mu.size()-1;++n )
        {
            const double c11 = chordLength2(r[m],ri,mu[n]);
            const double c12 = chordLength2(r[m],ri,mu[n+1]);
            double mu21 = cosineChord(r[m],ri,c11);
            double mu22 = cosineChord(r[m],ri,c12);
            if ( c11 < 1e-50 )
                mu21 = mu[n];
            if ( c12 < 1e-50 )
                mu22 = mu[n+1];
            const double c21 = chordLength2(ri,ro,mu21);
            const double c22 = chordLength2(ri,ro,mu22);

            const double p1 = exp(-sigmat*c11);
            const double p2 = exp(-sigmat*c12);

            double f1 = p1*c21;
            double f2 = p2*c22;
            r1top = r1top + (mu[n+1]-mu[n])*(1.0/2.0)*(f1+f2);

            f1 = p1;
            f2 = p2;
            r1bot = r1bot + (mu[n+1]-mu[n])*(1.0/2.0)*(f1+f2);
        }
        double r2top = 0.0;
        double  r2bot = 0.0;
        for ( unsigned int n=0;n<mu.size()-1;++n )
        {
            const double c11 = chordLength2(r[m+1],ri,mu[n]);
            const double c12 = chordLength2(r[m+1],ri,mu[n+1]);
            double mu21 = cosineChord(r[m+1],ri,c11);
            double mu22 = cosineChord(r[m+1],ri,c12);
            if ( c11 < 1e-50 )
                mu21 = mu[n];
            if ( c12 < 1e-50 )
                mu22 = mu[n+1];
            const double c21 = chordLength2(ri,ro,mu21);
```

```
            const double c22 = chordLength2(ri,ro,mu22);

            const double p1 = exp(-sigmat*c11);
            const double p2 = exp(-sigmat*c12);

            double f1 = p1*c21;
            double f2 = p2*c22;
            r2top = r2top + (mu[n+1]-mu[n])*(1.0/2.0)*(f1+f2);

            f1 = p1;
            f2 = p2;
            r2bot = r2bot + (mu[n+1]-mu[n])*(1.0/2.0)*(f1+f2);
        }
        double f1 = r[m]*r[m]*r1top;
        double f2 = r[m+1]*r[m+1]*r2top;
        numer = numer + (r[m+1]-r[m])*(1.0/2.0)*(f1+f2);

        f1 = r[m]*r[m]*r1bot;
        f2 = r[m+1]*r[m+1]*r2bot;
        denom = denom + (r[m+1]-r[m])*(1.0/2.0)*(f1+f2);
    }
    return numer/denom;
}

double effectiveThickness( double ri,double ro,int power )
{
    if ( power == 2 )
    {
        const double a = ri;
        const double b = ro;
        const double t2 = (3.0*(-2.0*pow(a,4.0)
            + a*b* (b*b + a*a) - pow(a*a - b*b,2.0)
            * log(2.0*(a + b))))/ (8.0*pow(a,3.0));
        const double t1 = (3.0*(-pow(a*a - b*b,2.0)
            * log(2.0*(sqrt(b*b - a*a)))))/ (8.0*pow(a,3.0));
        return t2-t1;
    }
    else if ( power == 1 )
    {
        return (2.0/3.0) * (pow(ro,3.0) - pow(pow(ro,2.0)
            -pow(ri,2.0),3.0/2.0) - pow(ri,3.0))/pow(ri,2.0);
    }
    else if ( power == 0 )
    {
        const double a = ri;
        const double b = ro;
        const double t2 = (b-a)/2.0
            + ((b*b-a*a)* log(2.0*(a*a + a*b)))/ (2.0*a);
        const double t1 = ((b*b-a*a)
            * log(2.0*(a*sqrt(b*b-a*a))))/ (2.0*a);
        return t2-t1;
    }
    else
    {
        return ro-ri;
    }
}

double invEffectiveThickness( double ri,double thickness )
{
    // iterate to find effective thickness
    double et1 = thickness*2.0;
    double et2 = thickness;
    const int maxIter = 100;
    for ( int i=0;i<maxIter;++i )
    {
        const double t1 = effectiveThickness( ri,ri+et1,1 );
        const double t2 = effectiveThickness( ri,ri+et2,1 );
        if ( fabs( t2-thickness ) < 1e-5 )
        {
            return et2;
            break;
        }
        const double m = (t2-t1)/(et2-et1);
        const double temp = et2;
        et2 = et1 - (t1-thickness)/m;
```

```
        et1 = temp;
    }
    return thickness;
}


double shield::transmissionProb( double ri,double ro,
    double Eo,double cospow )
{
    using namespace std;
    const double pi = phys::pi;
    // estimate transmission probability from first principles

    // starting at inner surface, assume cos^2 distribution
    // iterate over each possible emission angle
    vector<double> mu = linspace(0.0,1.0,10);
    const int M = static_cast<int>(mu.size()) -1;

    // estimate of transmission probability
    double est = 0.0;

    // hard code in lead cross section for now
    //photonxsec xsec( my_Z,rho );

    // iterate over emission angle bins
    for ( int m=0;m<M;++m )
    {
        // get actual angles of this bin
        const double theta1 = acos(mu[m]);
        const double theta2 = acos(mu[m+1]);
        // average chord length through material
        const double chord1 = chordLength( ri,ro,pi-theta1 );
        const double chord2 = chordLength( ri,ro,pi-theta2 );
        const double chord = (chord1+chord2)/2.0;
        // probability of source emission in this angle bin
        //const double Pe = mu[m+1]*mu[m+1] - mu[m]*mu[m];
        const double Pe = pow(mu[m+1],cospow+1) - pow(mu[m],cospow+1);
        // probability of interacting along this chord length
        const double Sigmat = my_xsec.tot(Eo);
        const double Pi = 1.0 - exp( -Sigmat*chord );
        // probability of not interacting
        const double Pni = exp( -Sigmat*chord );
        // probability of scattering given an interaction took place
        const double Sigmas = my_xsec.inc(Eo);
        const double Ps = Sigmas/Sigmat;
        // probability of pair production
        const double Sigmapp = my_xsec.ppe(Eo);
        const double Ppp = Sigmapp/Sigmat;
        // average interaction distance given an interaction took place
        const double xbar = aveLength( Sigmat,chord );
        // radial distance at first interaction
        const double thetabar = (theta1+theta2)/2;
        const double r = sqrt( ri*ri + xbar*xbar
            - 2*ri*xbar*cos(pi-thetabar) );
        // angle between radial direction and emission angle
        const double beta
            = acos((r*r + xbar*xbar - ri*ri)/(2*r*xbar));
        // probability of escaping after first scatter
        const double Pesc
            = getEscapeProbability( ri,r,ro,beta,xbar,Eo );
        // probability of pair production photons escaping
        // const double Pppesc
        //  = getEscapeProbability( ri,r,ro,xbar,Eo );
        // calculate tally estimate for this angle
        //est += Pe*( Pni + Pi*( Ps*Pesc + Ppp*Pppesc ) );
        est += Pe*( Pni + Pi*Ps*Pesc );

        // only uncollided
        //est += Pe*Pni ;
    }
    return est;
}

double shield::transmissionProb( double ri,double ro,double Eo,
    const std::vector<double>& prob )
{
```

```
    using namespace std;
    const double pi = phys::pi;
    // estimate transmission probability from first principles

    // starting at inner surface, assume cos^2 distribution
    // iterate over each possible emission angle
    vector<double> mu = linspace(0.0,1.0,40);
    const int M = static_cast<int>(mu.size()) -1;

    // estimate of transmission probability
    double est = 0.0;

    // hard code in lead cross section
    //photonxsec xsec( my_Z,my_rho );

    // iterate over emission angle bins
    for ( int m=0;m<M;++m )
    {
        // get actual angles of this bin
        const double theta1 = acos(mu[m]);
        const double theta2 = acos(mu[m+1]);
        // average chord length through material
        const double chord1 = chordLength( ri,ro,pi-theta1 );
        const double chord2 = chordLength( ri,ro,pi-theta2 );
        const double chord = (chord1+chord2)/2.0;
        // probability of source emission in this angle bin
        //const double Pe = mu[m+1]*mu[m+1] - mu[m]*mu[m];
        const double Pe = prob[m+1];
        // probability of interacting along this chord length
        const double Sigmat = my_xsec.tot(Eo);
        const double Pi = 1.0 - exp( -Sigmat*chord );
        // probability of not interacting
        const double Pni = exp( -Sigmat*chord );
        // probability of scattering given an interaction took place
        const double Sigmas = my_xsec.inc(Eo);
        const double Ps = Sigmas/Sigmat;
        // probability of pair production
        const double Sigmapp = my_xsec.ppe(Eo);
        const double Ppp = Sigmapp/Sigmat;
        // average interaction distance given an interaction took place
        const double xbar = aveLength( Sigmat,chord );
        // radial distance at first interaction
        const double thetabar = (theta1+theta2)/2;
        const double r = sqrt( ri*ri + xbar*xbar
            - 2*ri*xbar*cos(pi-thetabar) );
        // angle between radial direction and emission angle
        const double beta = acos((r*r + xbar*xbar
            - ri*ri)/(2*r*xbar));
        // probability of escaping after first scatter
        const double Pesc = getEscapeProbability( ri,r,
            ro,beta,xbar,Eo );
        // probability of pair production photons escaping
        // const double Pppesc = getEscapeProbability( ri,r,
        //  ro,xbar,Eo );
        // calculate tally estimate for this angle
        //est += Pe*( Pni + Pi*( Ps*Pesc + Ppp*Pppesc ) );
        est += Pe*( Pni + Pi*Ps*Pesc );

        // only uncollided
        //est += Pe*Pni ;
    }
    return est;
}


int shield::numInErgs ( )
{
    return my_R.numergin();
}

int shield::numInErgs ( ) const
{
    return my_R.numergin();
}
```

```
int shield::numOutErgs ( )
{
    return my_R.numergout();
}

int shield::numOutErgs ( ) const
{
    return my_R.numergout();
}


void shield::readDataFile()
{
    // get list of source energies
    //my_srcErg = readbin( my_datapath+"srcerg.dat" );

    return;
}

shield::shield ( const std::string& path )
{
    initialize( path );
}

void shield::initialize ( const std::string& path )
{
    my_datapath = path + sep();
    //readDataFile();
    // set all indices to -1
    my_srcIdx1 = -1;
    radIdx1_1 = -1;
    radIdx1_2 = -1;
    my_srcIdx3 = -1;
    radIdx3_1 = -1;
    radIdx3_2 = -1;

    my_pointResult1 = 1.0;
    my_pointResult3 = 1.0;
    my_planeResult1 = 1.0;
    my_planeResult3 = 1.0;

    my_geomScale1 = 1.0;
    my_geomScale3 = 1.0;
}


std::string shield::getTallyEnergyPath( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
}

std::vector<std::string> shield::getTallyPath( int ergIdx )
{
    std::vector<std::string> result(2);
    // check to see if this energy index has the thickness required
    if ( my_radMap.find(ergIdx) == my_radMap.end() )
    {
        // just return a blank string if nothing was found
        result[0] = "";
        result[1] = "";
        return result;
    }
    result[0] = my_datapath + "erg" + str(ergIdx) + sep()
        + "rad" + str(getRadiusIndex(ergIdx)[0]) + sep();
    result[1] = my_datapath + "erg" + str(ergIdx) + sep()
        + "rad" + str(getRadiusIndex(ergIdx)[1]) + sep();
    return result;
}

std::string shield::getTallyEnergyPath2( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
}

std::vector<std::string> shield::getTallyPath2( int ergIdx )
```

```
{
    std::vector<std::string> result(2);
    // check to see if this energy index has the thickness required
    if ( my_radMap.find(ergIdx) == my_radMap.end() )
    {
        // just return a blank string if nothing was found
        result[0] = "";
        result[1] = "";
        return result;
    }
    result[0] = my_datapath + "erg" + str(ergIdx) + sep()
        + "rad" + str(getRadiusIndex2(ergIdx)[0]) + sep();
    result[1] = my_datapath + "erg" + str(ergIdx) + sep()
        + "rad" + str(getRadiusIndex2(ergIdx)[1]) + sep();
    return result;
}

std::vector<int> shield::getRadiusIndex( int ergIdx )
{
    std::vector<int> result(2);
    result[0] = my_radMap[ergIdx];
    result[1] = my_radMap[ergIdx]+1;
    return result;
}

// return two closest interpolationpoints
std::vector<double> shield::getRadius( int ergIdx )
{
    std::vector<double> result(2);
    result[0] = radius[ergIdx][ my_radMap[ergIdx] ];
    result[1] = radius[ergIdx][ my_radMap[ergIdx]+1 ];
    return result;
}


std::vector<int> shield::getRadiusIndex2( int ergIdx )
{
    std::vector<int> result(2);
    result[0] = my_radMap2[ergIdx];
    result[1] = my_radMap2[ergIdx]+1;
    return result;
}

// return two closest interpolationpoints
std::vector<double> shield::getRadius2( int ergIdx )
{
    std::vector<double> result(2);
    result[0] = radius[ergIdx][ my_radMap2[ergIdx] ];
    result[1] = radius[ergIdx][ my_radMap2[ergIdx]+1 ];
    return result;
}

double shield::getPlaneResult( const std::string& talPath )
{
    std::ifstream in( (talPath+"plane.dat").c_str() );
    double planeResult;
    in >> planeResult;
    in.close();
    return planeResult;
}


void shield::scaleGeometry( tallyPtr tal1,
    const std::string& talPath, int idx )
{
    std::tr1::shared_ptr<tallytag> tal
        = std::tr1::dynamic_pointer_cast<tallytag>( tal1 );
    if ( ! tal )
    {
        throw fatal_error("could not dynamic cast ptally to tallytag");
    }

    const double ri = my_snmRadius;
    const double ro = my_snmRadius+my_thickness[idx];
    const double t = my_thickness[idx];
```

```
    std::vector<double> point = tal->sumParts();

    double cosPower = 1.5;
    for ( unsigned int i=1;i<my_angErg.size();++i )
    {
        if ( my_srcErg[idx] > my_angErg[i-1]
            && my_srcErg[idx] <= my_angErg[i] )
        {
            cosPower = my_angDist[i];
            break;
        }
    }
    const double scaler = transmissionProb( ri,ro,
        my_srcErg[idx],cosPower )
        /transmissionProb( 0.001*t,1.001*t,my_srcErg[idx],cosPower );
    tal->scaleBy( scaler );

    //std::cout << "scaler = " << scaler << std::endl;
}

std::vector<datapoint> shield::interpolateTallies(
    double newpeak,const std::vector< double >& newerg,
    int ergIdx1,int ergIdx3 )
{
    // get indices
    const int newSrcIdx1 = ergIdx1;
    const int newRadIdx1_1 = getRadiusIndex(ergIdx1)[0];
    const int newRadIdx1_2 = getRadiusIndex(ergIdx1)[1];
    const int newSrcIdx3 = ergIdx3;
    const int newRadIdx3_1 = getRadiusIndex(ergIdx3)[0];
    const int newRadIdx3_2 = getRadiusIndex(ergIdx3)[1];
    // parse tallies
    if ( newSrcIdx1 != my_srcIdx1 || newSrcIdx3 != my_srcIdx3 )
    {
        std::string talErgPath;
        std::string talPath;

        my_srcIdx1 = newSrcIdx1;
        talErgPath = getTallyEnergyPath(ergIdx1);
        talPath = getTallyPath(ergIdx1)[0];
        tal1_1->parse( talErgPath,talPath );
        tal1_1->setSourceEnergy( my_srcErg[ergIdx1] );
        scaleGeometry(tal1_1,talPath,ergIdx1);
        radIdx1_1 = newRadIdx1_1;

        //talErgPath = getTallyEnergyPath(ergIdx1);
        //talPath = getTallyPath2(ergIdx1)[0];
        //tal1_1_2->parse( talErgPath,talPath );
        //tal1_1_2->setSourceEnergy( my_srcErg[ergIdx1] );


        talErgPath = getTallyEnergyPath(ergIdx1);
        talPath = getTallyPath(ergIdx1)[1];
        tal1_2->parse( talErgPath,talPath );
        tal1_2->setSourceEnergy( my_srcErg[ergIdx1] );
        scaleGeometry(tal1_2,talPath,ergIdx1);
        radIdx1_2 = newRadIdx1_2;

        //talErgPath = getTallyEnergyPath(ergIdx1);
        //talPath = getTallyPath2(ergIdx1)[1];
        //tal1_2_2->parse( talErgPath,talPath );
        //tal1_2_2->setSourceEnergy( my_srcErg[ergIdx1] );


        my_srcIdx3 = newSrcIdx3;
        talErgPath = getTallyEnergyPath(ergIdx3);
        talPath = getTallyPath(ergIdx3)[0];
        tal3_1->parse( talErgPath,talPath );
        tal3_1->setSourceEnergy( my_srcErg[ergIdx3] );
        scaleGeometry(tal3_1,talPath,ergIdx3);
        radIdx3_1 = newRadIdx3_1;

        //talErgPath = getTallyEnergyPath(ergIdx3);
        //talPath = getTallyPath2(ergIdx3)[0];
        //tal3_1_2->parse( talErgPath,talPath );
```

```
        //tal3_1_2->setSourceEnergy( my_srcErg[ergIdx3] );

        talErgPath = getTallyEnergyPath(ergIdx3);
        talPath = getTallyPath(ergIdx3)[1];
        tal3_2->parse( talErgPath,talPath );
        tal3_2->setSourceEnergy( my_srcErg[ergIdx3] );
        scaleGeometry(tal3_2,talPath,ergIdx3);
        radIdx3_2 = newRadIdx3_2;

        //talErgPath = getTallyEnergyPath(ergIdx3);
        //talPath = getTallyPath2(ergIdx3)[1];
        //tal3_2_2->parse( talErgPath,talPath );
        //tal3_2_2->setSourceEnergy( my_srcErg[ergIdx3] );


        // check to see if thickness exceeds max radius
        // for that energy, then just make it a zero tally
        if ( my_thickness[ergIdx1] > getRadius(ergIdx1)[1]
            || my_thickness[ergIdx3] > getRadius(ergIdx3)[1] )
        {
            //std::cout << "making them zero" << std::endl;
            tal1->makeZero( tal1_1 );
            tal3->makeZero( tal3_1 );
        }
        else
        {
            tal1->interpolate( tal1_1,tal1_2,getRadius(ergIdx1)[0],
                getRadius(ergIdx1)[1],my_thickness[ergIdx1],"lin" );
            tal3->interpolate( tal3_1,tal3_2,getRadius(ergIdx3)[0],
                getRadius(ergIdx3)[1],my_thickness[ergIdx3],"lin" );
        }

    }

    // interpolate by source energy
    taginterpolator K;
    K.setSourceEnergies(my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);
    std::vector<datapoint> interpResult = K.interpolate(
        newerg,tal1,tal3 );
    //std::cout << interpResult << std::endl;
    //int dummy;
    //std::cin >> dummy;
    return interpResult;
}

void shield::setAngularDistribution(
    const std::vector<double>& erg,const std::vector<double>& ang )
{
    my_angDist = ang;
    my_angErg = erg;
}

void shield::buildResponse ( double thickness,
    double omegaStream,
    double maxErg,
    int redFact,
    snm* snmModel )
{
    //std::cout << "building shield response" << std::endl;

    my_snmModel = snmModel;

    const double snmRadius = my_snmModel->getRadius( );

    my_rho = readVal<double>( my_datapath+"rho.dat" );
    my_Z = readtxt<int>( my_datapath+"Z.dat" );
    my_frac = readtxt<double>( my_datapath+"frac.dat" );
    my_xsec.initialize(my_Z,my_frac,my_rho);

    my_srcErg = readbin( my_datapath+"srcerg.dat" );

    //const double ri = snmRadius;
    //const double ro = snmRadius+thickness;
    //const double shieldscale
    // = effectiveThickness(ri,ro,2)/effectiveThickness(ri,ro,1);
```

```
my_thickness.resize(my_srcErg.size());
for ( unsigned int i=0;i<my_srcErg.size();++i )
{
    //const double shieldscale = effectiveThickness( ri,ro,
    //  my_srcErg[i],92,15.75 )/effectiveThickness(ri,ro,1);
    //const double shieldscale = effectiveThickness(ri,ro,2)
    //  /effectiveThickness(ri,ro,1);
    //my_thickness[i] = effectiveThickness( ri,ro,
    //  my_srcErg[i],92,15.75 );
    //my_thickness[i] = effectiveThickness(ri,ro,2);
    //my_thickness[i] = thickness*shieldscale;
    my_thickness[i] = thickness;
    //std::cout << my_srcErg[i] << " "
    //  << my_thickness[i] << std::endl;
}

my_thickness2 = thickness;

truethickness = thickness;

my_redFact = redFact;
my_snmRadius = snmRadius;

// find which energies have the available thickness required
// and record which upper radius index we should use
radius.resize( my_srcErg.size() );
for ( unsigned int i=0;i<my_srcErg.size();++i )
{
    // read in radii
    radius[i] = readbin( my_datapath+"erg"+str(i)+sep()+"rad.dat" );
    // find radius that's just over the thickness required
    // don't want first radius because
    //  need to interpolate b/t j and j-1
    for ( unsigned int j=1;j<radius[i].size();++j )
    {
        if ( radius[i][j] >= my_thickness[i] )
        {
            //std::cout << "radius[" << i << "]["
            //  << j << "] = " << radius[i][j]
            //  << "  >  " << my_thickness << std::endl;
            // store the energy index -> radius index in our map
            my_radMap[i] = j-1;
            break;
        }
        if ( j == radius[i].size()-1 )
        {
            my_radMap[i] = j-1;
        }
    }


    for ( unsigned int j=1;j<radius[i].size();++j )
    {
        if ( radius[i][j] >= my_thickness2 )
        {
            my_radMap2[i] = j-1;
            break;
        }
    }
}

// keep tallies in scope outside of loop for efficiency
// don't have to read files as often
tal1 = tallyPtr( new tallytag(my_redFact) );
tal1_1 = tallyPtr( new tallytag(my_redFact) );
tal1_2 = tallyPtr( new tallytag(my_redFact) );
tal1_1_2 = tallyPtr( new tallytag(my_redFact) );
tal1_2_2 = tallyPtr( new tallytag(my_redFact) );
tal3 = tallyPtr( new tallytag(my_redFact) );
tal3_1 = tallyPtr( new tallytag(my_redFact) );
tal3_2 = tallyPtr( new tallytag(my_redFact) );
tal3_1_2 = tallyPtr( new tallytag(my_redFact) );
tal3_2_2 = tallyPtr( new tallytag(my_redFact) );

// build response function matrix
```

```
    computeResponse( maxErg );

    // make identity matrix
    //std::cout << "making shield identity matrix" << std::endl;
    my_I.resize( my_R.numergout(),my_R.numergin() );
    my_I.ergout() = my_R.ergout();
    my_I.ergin() = my_R.ergin();
    my_I.identity();

    //my_R.checkData( );
    // apply streaming factor
    //std::cout << "applying streaming factor" << std::endl;
    my_R = my_R * ( 1 - omegaStream ) + my_I * omegaStream;

    //my_R.checkData( );

    //std::cout << "finished building shield response" << std::endl;

    return;
}

spectrum shield::operator() ( const spectrum& S_src )
{
    //std::cout << "applying shield response function" << std::endl;
    return my_R * S_src;
}
```

Listing B.44: shield.hpp

```cpp
#ifndef _shield_hpp_included_
#define _shield_hpp_included_
#include "submodel.hpp"
#include "response.hpp"
#include "snm.hpp"
#include "xsec.hpp"
#include <map>

class shield : public submodel
{
    public:

    void buildResponse ( double,double,double,int,snm* );

    void initialize ( const std::string& path );

    shield ( const std::string& path );
    shield ( ) { };

    void setAngularDistribution( const std::vector<double>& erg,
        const std::vector<double>& ang );
    spectrum operator() ( const spectrum& );

    private:

    double my_rho;
    std::vector<int> my_Z;
    std::vector<double> my_frac;
    photonxsec my_xsec;

    snm* my_snmModel;
    std::vector<double> my_angDist;
    std::vector<double> my_angErg;

    std::string getTallyEnergyPath( int );
    std::vector<std::string> getTallyPath( int );
    std::string getTallyEnergyPath2( int );
    std::vector<std::string> getTallyPath2( int );
    std::vector<int> getRadiusIndex( int );
    std::vector<double> getRadius( int );
    std::vector<int> getRadiusIndex2( int );
    std::vector<double> getRadius2( int );
    std::vector<datapoint> interpolateTallies(
        double,const std::vector< double >&,int,int );
    double getPlaneResult( const std::string& );


    void scaleGeometry( tallyPtr tal1,const std::string& talPath,int idx );

    int numInErgs ( );
    int numInErgs ( ) const;
    int numOutErgs ( );
    int numOutErgs ( ) const;

    void readDataFile();

    int findShieldIndex ( double& );

    double truethickness;
    std::vector<double> my_thickness;
    double my_thickness2;
    double my_snmRadius;
    std::vector< std::vector<double> > radius;


    double effectiveThickness( double ri,double ro,double E );
    std::vector<double> angularDistribution( double ri,double ro,double E );
    double getEscapeProbability( double ri,double r,double ro,
        double beta,double xbar,double Eo );
    double getEscapeProbability( double ri,double r,double ro,
        double xbar,double Eo );
    double transmissionProb( double ri,double ro,double Eo,
        double cospow );
    double transmissionProb( double ri,double ro,double Eo,
        const std::vector<double>& prob );
```

```
    // identity matrix that transforms
    // incoming energies to outgoing energies
    response my_I;

    // map of source indices to radii
    std::map< int,int > my_radMap;
    std::map< int,int > my_radMap2;

    // parsed tallies
    tallyPtr tal1;
    tallyPtr tal1_1;
    tallyPtr tal1_2;
    tallyPtr tal1_1_2;
    tallyPtr tal1_2_2;
    tallyPtr tal3;
    tallyPtr tal3_1;
    tallyPtr tal3_2;
    tallyPtr tal3_1_2;
    tallyPtr tal3_2_2;
    // parsed tallies indices
    int my_srcIdx1;
    int radIdx1_1;
    int radIdx1_2;
    int my_srcIdx3;
    int radIdx3_1;
    int radIdx3_2;

    double my_pointResult1;
    double my_m312Result1;
    double my_m412Result1;
    double my_planeResult1;

    double my_pointResult3;
    double my_m312Result3;
    double my_m412Result3;
    double my_planeResult3;

    double my_geomScale1;
    double my_geomScale3;

};

#endif
```

Listing B.45: snm.cpp

```
#include "snm.hpp"
#include "fileio.hpp"
#include <fstream>
#include <math.h>
#include <iostream>
#include "gammalines.hpp"
#include "tally.hpp"
#include "extras.hpp"
#include "interpolation.hpp"
#include "errh.hpp"
#include "phys.hpp"


int snm::numOutErgs ( )
{
    return my_R.numergout();
}

snm::snm ( const std::string& path )
{
    initialize( path );
}

void snm::initialize ( const std::string& path )
{
    my_datapath = path + sep();
    readDataFile();
    srcIdx1 = -1;
    srcIdx3 = -1;
}

void snm::readDataFile ( )
{
    // get list of source energies
    my_srcErg = readbin( my_datapath+"srcerg.dat" );
    // hard-code in volume and surface area used in mcnp calculation for now (1 kg)
    my_volume = 52.77196;
    my_surfaceArea = 68.03822;
    return;
}

double cosinefit( const std::vector<double>& X, const std::vector<double>& Y )
{
    //assert(X.size() == Y.size());

    const int N = static_cast<int>(X.size());
    std::vector<double> x(N);
    for ( int i=0;i<N;++i )
    {
        x[i] = log(X[i]);
    }
    std::vector<double> y(N);
    std::vector<double> w(N);
    for ( int i=0;i<N;++i )
    {
        y[i] = log(Y[i]);
        if ( isinf(y[i]) || isnan(y[i]) )
        {
            return 1.5;
        }
        w[i] = pow(Y[i],2.0);
    }
    w = normalize(w);
    matrix<double> W(N,N);
    W.setDiagonal(w);

    double a = 1.0;
    for ( int k=0;k<100;++k )
    {
        matrix<double> J;
        J = 1.0/(1.0+a) + x;
        std::vector<double> dy = log(1.0+a) + a*x - y;
        matrix<double> Jt = J.transpose();

        matrix<double> JtWJ = Jt*W*J;
```

```
        std::vector<double> JtWdy = Jt*W*dy;
        std::vector<double> dbeta = LU_Solve(JtWJ,JtWdy);

        a = a - dbeta[0];
        //std::cout << "Iter " << k << std::endl;
        //std::cout << "\t" << "a  = " << a << std::endl;
        //std::cout << "\t" << "da = " << dbeta[0] << std::endl;
        if (fabs(dbeta[0]) < 1e-5 )
        {
            if ( a < 1.0 )
            {
                return 1.0;
            }
            else if ( a > 2.0 )
            {
                return 2.0;
            }
            else
            {
                return a;
            }
        }
    }
    return a;
}

std::vector<double> snm::getAngularEnergy( )
{
    return readbin(my_datapath+"atalerg.dat");
}

std::vector<double> snm::getAngularDistribution( dspectrum S,spectrum B )
{
    //mtally tal = parsemfile(my_datapath+"heu1kgangle1.m")[0];
    //std::vector<double> terg = tal.erg();
    //std::vector<double> tcos = tal.cos();
    std::vector<double> terg = readtxt<double>(my_datapath+"heu1kgangletalerg.dat");
    std::vector<double> tcos = readtxt<double>(my_datapath+"heu1kgangletalcos.dat");
    std::vector<double> rawdata = readtxt<double>(my_datapath+"heu1kgangle.dat");
    const int E = static_cast<int>(terg.size());
    const int C = static_cast<int>(tcos.size());
    // create spectrum with this group structure
    // add in discrete gammas
    std::vector<double> spec(E);
    for ( int s=0;s<S.numdat();++s )
    {
        for ( int e=1;e<E;++e )
        {
            if ( S.erg(s) >= terg[e-1] && S.erg(s) <= terg[e] )
            {
                spec[e] += S(s).get();
                break;
            }
        }
    }
    // add bremsstrahlung
    response I(B.erg(),terg);
    I.identity();
    std::vector<datapoint> bdata = (I*B).data();
    for ( unsigned int i=0;i<bdata.size();++i )
    {
        spec[i] = spec[i] + bdata[i].get();
    }
    // normalize
    spec = normalize(spec);
    //datapoint = rawdata(c+1+C*(e+E*(u-1)));
    std::vector<double> merg(E-1);
    for ( int e=0;e<E-1;++e )
    {
        merg[e] = (terg[e]+terg[e+1])/2.0;
    }
    std::vector<double> mcos(C-1);
    for ( int c=0;c<C-1;++c )
    {
        mcos[c] = (tcos[c]+tcos[c+1])/2.0;
    }
```

```
    std::vector<double> result(E-1);
    for ( int e=0;e<E-1;++e )
    {
        std::vector<double> newdata(C-1);
        for ( int c=0;c<C-1;++c )
        {
            for ( int u=0;u<E-1;++u )
            {
                //e+nerg*(c+ncos*(s+nseg*(u+nuser*(f+nflag*cs))))
                //int mtally::idx( int cs,int f,int u,int s,int c,int e )
                //newdata[c] += tal.val(0,0,u,0,c+1,e+1) * spec[u+1];
                //newdata[c] += rawdata[e+1+E*(c+1+C*u)] * spec[u+1];
                newdata[c] += rawdata[c+1+C*(e+1+E*u)] * spec[u+1];
            }

        }
        newdata = normalize(newdata);
        for ( int c=0;c<C-1;++c )
        {
            newdata[c] = newdata[c]/(tcos[c+1]-tcos[c]);
        }
        result[e] = cosinefit(mcos,newdata);
    }
    return result;
}

std::string snm::getTallyEnergyPath( int ergIdx )
{
    // for snm, the tally energies and
    // tally values are in the same directory
    return getTallyPath(ergIdx);
}

std::string snm::getTallyPath( int ergIdx )
{
    return my_datapath + "erg" + str(ergIdx) + sep();
}

std::vector<datapoint> snm::interpolateTallies(
    double newpeak,const std::vector< double >& newerg,
    int ergIdx1,int ergIdx3)
{
    const int newSrcIdx1 = ergIdx1;
    const int newSrcIdx3 = ergIdx3;
    // read tallies from file
    // see if we can reuse tallies
    taginterpolator K;

    if ( newSrcIdx1 != srcIdx1 || newSrcIdx3 != srcIdx3 )
    {
        // Tried to copy 3 to 1 before, couldn't get it working
        std::string talErgPath = getTallyEnergyPath(ergIdx1);
        std::string talPath = getTallyPath(ergIdx1);
        if ( talErgPath.empty() || talPath.empty() )
        {
            throw warning("could not parse file for snm");
        }
        tal1->parse( talErgPath,talPath );
        tal1->setSourceEnergy( my_srcErg[ergIdx1] );
        srcIdx1 = newSrcIdx1;

        talErgPath = getTallyEnergyPath(ergIdx3);
        talPath = getTallyPath(ergIdx3);
        if ( talErgPath.empty() || talPath.empty() )
        {
            throw warning("could not parse file for snm");
        }
        tal3->parse( talErgPath,talPath );
        tal3->setSourceEnergy( my_srcErg[ergIdx3] );
        srcIdx3 = newSrcIdx3;
    }

    K.setSourceEnergies(my_srcErg[ergIdx1],newpeak,my_srcErg[ergIdx3]);

    std::vector<datapoint> result = K.interpolate( newerg,tal1,tal3 );
```

```
    return result;
}

void snm::buildResponse ( double mass,
                          double density,
                          double maxErg,
                          int redFact )
{
    //std::cout << "building snm response" << std::endl;
    //std::cout << "mass    = " << mass << " g " << std::endl;
    //std::cout << "density = " << density << " g/cc " << std::endl;

    my_redFact = redFact;
    my_mass = mass;
    my_density = density;

    // keep tallies in scope outside of loop for efficiency
    // don't have to read files as often
    tal1 = tallyPtr( new tallytag(my_redFact) );
    tal3 = tallyPtr( new tallytag(my_redFact) );
    // build response function matrix
    computeResponse( maxErg );

    // Scale response matrix by surface area to volume ratios
    const double radius = getRadius( );
    //std::cout << "radius  = " << radius << " cm " << std::endl;
    const double new_volume = ( 4.0/3.0 ) *phys::pi*pow ( radius,3.0 );
    const double new_surfaceArea = 4.0*phys::pi*pow ( radius,2.0 );
    const double factor = (my_volume/my_surfaceArea)
        *(new_surfaceArea/new_volume);

    my_R = my_R * factor;

    //std::cout << "finished building snm response" << std::endl;
    return;
}

double snm::getRadius( )
{
    return pow ( my_mass/my_density * ( 3.0/4.0 )
        * ( 1.0/phys::pi ),1.0/3.0 );
}

spectrum snm::operator() ( dspectrum& S_src )
{
    //std::cout << "applying snm response" << std::endl;
    return my_R*S_src;
}

spectrum snm::operator() ( spectrum& S_src )
{
    //std::cout << "applying snm response" << std::endl;
    return my_R*S_src;
}
```

Listing B.46: snm.hpp

```
#ifndef _snm_hpp_included_
#define _snm_hpp_included_
#include <string>
#include <vector>
#include <map>

#include "submodel.hpp"
#include "spectrum.hpp"
#include "dspectrum.hpp"
#include "response.hpp"
#include "datapoint.hpp"
#include "data.hpp"


class snm : public submodel
{
    public:

    void readDataFile();

    void initialize ( const std::string& );

    void buildResponse ( double,
                         double,
                         double,
                         int );

    double getRadius( );

    snm ( const std::string& );
    snm ( ) { };

    spectrum operator() ( dspectrum& );
    spectrum operator() ( spectrum& );

    std::vector<double> getAngularDistribution( dspectrum S,spectrum B );
    std::vector<double> getAngularEnergy( );

    private:

    std::string getTallyEnergyPath( int );
    std::string getTallyPath( int );
    std::vector<datapoint> interpolateTallies(
        double,const std::vector< double >&,int,int );

    int numOutErgs ( );

    double my_mass;
    double my_density;
    double my_volume;
    double my_surfaceArea;

    // parsed tallies
    tallyPtr tal1;
    tallyPtr tal3;
    // parsed tallies indices
    int srcIdx1;
    int srcIdx3;

};

#endif
```

Listing B.47: source.cpp

```
#include "source.hpp"
#include "fileio.hpp"
#include <fstream>
#include <math.h>
#include <iostream>
#include <iomanip>
#include "gammalines.hpp"

int source::numOutErgs ( )
{
    return my_gammalines.numerg();
}

double source::getMaxErg( )
{
    if ( my_gammalines.numerg() == 0
        && my_bremsstrahlung.numerg() == 0 )
    {
        throw fatal_error("there are no gamma lines \
or bremsstrahlung, so no maximum energy");
    }
    double max = 0.0;
    for ( int i=0;i<my_gammalines.numerg();++i )
    {
        if ( my_gammalines.erg(i) > max )
        {
            max = my_gammalines.erg(i);
        }
    }
    for ( int i=0;i<my_bremsstrahlung.numerg();++i )
    {
        if ( my_bremsstrahlung.erg(i) > max )
        {
            max = my_bremsstrahlung.erg(i);
        }
    }
    return max;
}

void source::buildResponse ( double mass,
    const std::vector < std::string >& isotope,
    const std::vector < double >& fraction,
    double age )
{
    const int numDesiredIsos = static_cast<int> ( isotope.size() );

    // build radsrc input line
    std::stringstream radsrcinput;
    //std::cout << "generating radsrc input" << std::endl;
    for ( int i=0;i<numDesiredIsos;++i )
    {
        radsrcinput << isotope[i] + " " << fraction[i]*100.0 << " ";
    }
    radsrcinput << "Age " << age;
    // get gamma lines from radsrc
    my_gammalines = getGammaLines( radsrcinput.str(),0.001,4 );
    // get bremsstrahlung from radsrc
    my_bremsstrahlung = getBremsstrahlung( radsrcinput.str() );
    //my_bremsstrahlung.print();


    // units of gammalines is photons/sec/gram,
    // so multiply by mass to get photons/sec
    my_gammalines = my_gammalines * mass;
    my_bremsstrahlung = my_bremsstrahlung * mass;

    // print gamma lines
    //my_gammalines.print();

    std::cout << "Gamma Source                : "
        << std::setw(15) << my_gammalines.sum()
        << " g/s" << std::endl;
    std::cout << "Bremsstrahlung Source       : "
        << std::setw(15) << my_bremsstrahlung.sum()
        << " g/s" << std::endl;
```

```
    //std::cout << "done building source" << std::endl;
    return;
}

dspectrum source::getGamma ( )
{
    return my_gammalines;
}

spectrum source::getBrem ( )
{
    return my_bremsstrahlung;
}
```

Listing B.48: source.hpp

```
#ifndef _source_hpp_included_
#define _source_hpp_included_
#include <string>
#include <vector>
#include <map>
#include "submodel.hpp"
#include "spectrum.hpp"
#include "dspectrum.hpp"
#include "datapoint.hpp"
#include "data.hpp"


class source
{
    public:

    double getMaxErg( );

    void buildResponse ( double,
                         const std::vector < std::string >&,
                         const std::vector < double >&,
                         double );


    source ( ) { };

    dspectrum getGamma ( );
    spectrum getBrem ( );

    private:

    // radsrc gammalines
    dspectrum my_gammalines;
    // radsrc brem
    spectrum my_bremsstrahlung;

    int numOutErgs ( );

};

#endif
```

Listing B.49: spectrum.cpp

```
#include "spectrum.hpp"
#include <iostream>
#include <iomanip>
#include "response.hpp"
#include "matrix.hpp"

int spectrum::numdat ( )
{
    return static_cast<int> ( my_data.size() );
}

int spectrum::numdat ( ) const
{
    return static_cast<int> ( my_data.size() );
}

int spectrum::numerg ( )
{
    return static_cast<int> ( my_energy.size() );
}

int spectrum::numerg ( ) const
{
    return static_cast<int> ( my_energy.size() );
}

double& spectrum::erg ( int a )
{
    return my_energy[a];
}

double spectrum::erg ( int a ) const
{
    return my_energy[a];
}

std::vector < double >& spectrum::erg ( )
{
    return my_energy;
}

std::vector < double > spectrum::erg ( ) const
{
    return my_energy;
}

double spectrum::lasterg ( ) const
{
    return my_energy[my_energy.size()-1];
}

double spectrum::firsterg ( ) const
{
    return my_energy[0];
}

datapoint& spectrum::operator() ( int a )
{
    return my_data[a];
}

datapoint spectrum::operator() ( int a ) const
{
    return my_data[a];
}

std::vector < datapoint >& spectrum::data ( )
{
    return my_data;
}

std::vector < datapoint > spectrum::data ( ) const
{
    return my_data;
}
```

```
void spectrum::resize ( int numergs )
{
    my_data.resize ( numergs-1 );
    my_energy.resize ( numergs );
    return;
}

double spectrum::sum ( ) const
{
    double tot = 0.0;
    for ( int i=0;i<numdat();++i )
    {
        tot += my_data[i].get();
    }
    return tot;
}

void spectrum::unity( )
{
    for ( int i=0;i<numdat();++i )
    {
        my_data[i] = datapoint(1.0,0.0);
    }
}

void spectrum::print ( std::ofstream& outfile,
                       const std::string& header,
                       const std::vector<std::string>& title,
                       const std::vector<std::string>& unit )
{
    outfile << "NUMERGS_" << numerg() << std::endl;

    outfile << std::endl << std::endl;
    outfile << header << std::endl << std::endl;
    outfile << std::setw ( 15 ) << title[0];
    outfile << std::setw ( 25 ) << title[1];
    outfile << std::setw ( 25 ) << title[2] << std::endl;
    std::string newunit = "[" + unit[0] + "]";
    outfile << std::setw ( 15 ) << newunit;
    newunit = "[" + unit[1] + "]";
    outfile << std::setw ( 25 ) << newunit;
    newunit = "[" + unit[2] + "]";
    outfile << std::setw ( 25 ) << newunit << std::endl;

    // first bin is zero, always
    outfile << std::setw ( 15 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 )
        << my_energy[0];
    outfile << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 )
        << 0;
    outfile << std::setw ( 25 )
        << std::resetiosflags ( std::ios::scientific )
        << std::setprecision ( 6 )
        << 0 << std::endl;

    // sum up squares to report error for total
    double toterr = 0.0;
    // sum for energy range 40 keV - 3000 keV (for benchmarks)
    datapoint limsum(0.0,0.0);
    for ( int i=0;i<numdat();++i )
    {
        outfile << std::setw ( 15 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 )
            << my_energy[i+1];
        outfile << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 )
            << my_data[i].get();
        //outfile << std::setw ( 25 )
        //  << std::setiosflags ( std::ios::scientific )
        //  << std::setprecision ( 6 )
```

```
        //  << my_data[i].getErr() << std::endl;
        outfile << std::setw ( 25 )
            << std::setiosflags ( std::ios::scientific )
            << std::setprecision ( 6 )
            << my_data[i].getErr() << std::endl;
        toterr += my_data[i].getErr()
            *my_data[i].getErr()*my_data[i].get()
            *my_data[i].get();
        if ( my_energy[i+1] > 0.04 && my_energy[i+1] < 3.0 )
        {
            limsum = limsum + my_data[i];
        }
    }
    outfile << std::endl;
    outfile << std::setw ( 15 ) << "Total"
        << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 ) << sum();
    outfile << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 )
        << sqrt(toterr)/sum() << std::endl;
    outfile << std::setw ( 15 )
        << "Bench" << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 ) << limsum.get();
    outfile << std::setw ( 25 )
        << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 ) << limsum.getErr()
        << std::endl;
    return;
}

void spectrum::operator= ( const tally& tal )
{
    my_energy = tal.erg();
    my_data.resize(my_energy.size()-1);
    for ( unsigned int e=0;e<my_energy.size()-1;++e )
    {
        my_data[e] = tal.tal(e+1);
    }
    return;
}

void spectrum::operator= ( const spectrum& a )
{
    my_energy = a.erg();
    my_data = a.data();
    return;
}

void spectrum::operator= ( const datapoint& a )
{
    for ( int i=0;i<numdat();++i )
    {
        my_data[i] = a;
    }
    return;
}

void spectrum::operator= ( double a )
{
    for ( int i=0;i<numdat();++i )
    {
        my_data[i].set ( a );
        my_data[i].set ( 0.0 );
    }
    return;
}

spectrum::spectrum ( int numergs )
{
    initialize ( numergs );
}

spectrum::spectrum ( const std::vector<double>& erg )
```

```
{
    initialize ( erg );
}

void spectrum::initialize ( int numergs )
{
    my_data.resize ( numergs-1 );
    my_energy.resize ( numergs );
    return;
}

void spectrum::initialize ( const std::vector<double>& erg )
{
    my_energy = erg;
    my_data.resize ( erg.size()-1 );
    return;
}



spectrum operator+ ( const spectrum& a,const spectrum& b )
{
    //if ( a.numerg() != b.numerg() || a.numdat() != b.numdat() )
    //{
    //  throw fatal_error ( "size mismatch in spectrum addition" );
    //}
    if ( a.numerg() == 0 )
    {
        return b;
    }
    if ( b.numerg() == 0 )
    {
        return a;
    }


    if ( a.numerg() != b.numerg() )
    {
        bool match = true;
        int E = 0;
        if ( a.numerg() > b.numerg() )
        {
            E = b.numerg();
        }
        else
        {
            E = a.numerg();
        }
        for ( int i=0;i<E;++i )
        {
            if ( fabs( a.erg(i) - b.erg(i) ) > 1e-5 )
            {
                match = false;
                break;
            }
        }

        if ( match )
        {
            spectrum newa = a;
            spectrum newb = b;
            if ( newa.numerg() > newb.numerg() )
            {
                newb.erg() = newa.erg();
                const int n = newb.numdat();
                const int N = newa.numdat();
                newb.data().resize( N );
                for ( int i=n;i<N;++i )
                {
                    newb(i) = datapoint( 0.0,0.0 );
                }
            }
            else
            {
                newa.erg() = newb.erg();
```

```
                const int n = newa.numdat();
                const int N = newb.numdat();
                newa.data().resize( N );
                for ( int i=n;i<N;++i )
                {
                    newa(i) = datapoint( 0.0,0.0 );
                }
            }
            spectrum result;
            result.erg() = newa.erg();
            result.data() = newa.data() + newb.data();
            return result;
        }
        else
        {
            spectrum newa = a;
            spectrum newb = b;
            spectrum result;
            if ( a.numerg() > b.numerg() )
            {
                response I( b.erg(),a.erg() );
                I.identity();
                newb = I*b;
                result.erg() = a.erg();
            }
            else
            {
                response I( a.erg(),b.erg() );
                I.identity();
                newa = I*a;
                result.erg() = b.erg();
            }
            result.data() = newa.data() + newb.data();
            return result;
        }
    }
    else
    {
        spectrum result;
        result.erg() = a.erg();
        result.data() = a.data() + b.data();
        return result;
    }
}

spectrum operator* ( const spectrum& a,double b )
{
    spectrum result;
    result.erg() = a.erg();
    result.data() = a.data() *b;
    return result;
}

spectrum operator* ( double b,const spectrum& a )
{
    return a*b;
}

void spectrum::print ( )
{
    std::cout << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 );
    if ( numerg() > 0 )
    {
        std::cout << std::setw ( 13 ) << my_energy[0];
        std::cout << std::setw ( 13 ) << 0.0;
        std::cout << std::setw ( 13 ) << 0.0;
        std::cout << std::endl;
        for ( int i=0;i<numdat();++i )
        {
            std::cout << std::setw ( 13 ) << my_energy[i+1];
            std::cout << std::setw ( 13 ) << my_data[i].get();
            std::cout << std::setw ( 13 ) << my_data[i].getErr();
            std::cout << std::endl;
        }
    }
```

```
    return;
}

void spectrum::print ( ) const
{
    std::cout << std::setiosflags ( std::ios::scientific )
        << std::setprecision ( 6 );
    if ( numerg() > 0 )
    {
        std::cout << std::setw ( 13 ) << my_energy[0];
        std::cout << std::setw ( 13 ) << 0.0;
        std::cout << std::setw ( 13 ) << 0.0;
        std::cout << std::endl;
        for ( int i=0;i<numdat();++i )
        {
            std::cout << std::setw ( 13 ) << my_energy[i+1];
            std::cout << std::setw ( 13 ) << my_data[i].get();
            std::cout << std::setw ( 13 ) << my_data[i].getErr();
            std::cout << std::endl;
        }
    }
    return;
}
```

Listing B.50: spectrum.hpp

```
#ifndef _spectrum_hpp_included_
#define _spectrum_hpp_included_

#include <vector>
#include <string>
#include <fstream>

#include "errh.hpp"
#include "datapoint.hpp"
#include "tally.hpp"
#include "fileio.hpp"


class spectrum
{
    public:

    int numdat ( );
    int numdat ( ) const;

    int numerg ( );
    int numerg ( ) const;

    double& erg ( int );
    double erg ( int ) const;
    double lasterg ( ) const;
    double firsterg ( ) const;

    std::vector < double >& erg ( );
    std::vector < double > erg ( ) const;

    datapoint& operator() ( int );
    datapoint operator() ( int ) const;

    std::vector < datapoint >& data ( );
    std::vector < datapoint > data ( ) const;

    std::vector< double > get() const
    {
        std::vector<double> result(my_data.size());
        for ( unsigned int i=0;i<my_data.size();++i )
            result[i] = my_data[i].get();
        return result;
    }

    std::vector< double > getErr() const
    {
        std::vector<double> result(my_data.size());
        for ( unsigned int i=0;i<my_data.size();++i )
            result[i] = my_data[i].getErr();
        return result;
    }

    void read( const std::string& filename )
    {
        my_energy = readtxt<double>(filename+"_"+"energy"+".dat");
        std::vector<double> val
            = readtxt<double>(filename+"_"+"value"+".dat");
        std::vector<double> err
            = readtxt<double>(filename+"_"+"error"+".dat");
        if ( val.size() != my_energy.size()-1 ||
            err.size() != my_energy.size()-1 )
        {
            throw fatal_error("mismatched energy/value/error\
vectors in spectrum file read");
        }
        my_data.resize(val.size());
        for ( unsigned int i=0;i<val.size();++i )
        {
            my_data[i] = datapoint(val[i],err[i]);
        }
    }

    virtual void resize ( int );
```

```
    double sum ( ) const;

    void unity( );

    virtual void print ( std::ofstream&,const std::string&,
        const std::vector<std::string>&,
        const std::vector<std::string>& );
    void print ( );
    void print ( ) const;

//    void match ( const spectrum& );

    void operator= ( const spectrum& );
    void operator= ( const tally& );
    void operator= ( const datapoint& );
    void operator= ( double );

    virtual void initialize ( int );
    virtual void initialize ( const std::vector<double>& );

    spectrum() {};
    spectrum ( int );
    spectrum ( const std::vector<double>& );


    protected:

    std::vector < double > my_energy;
    std::vector < datapoint > my_data;

};


spectrum operator+ ( const spectrum&,const spectrum& );

spectrum operator* ( const spectrum&,double );

spectrum operator* ( double,const spectrum& );


#endif
```

Listing B.51: submodel.cpp

```cpp
#include "submodel.hpp"
#include "tally.hpp"
#include "extras.hpp"
#include "fileio.hpp"
#include "errh.hpp"
#include "diagnostic.hpp"

void submodel::computeResponse( double maxErg )
{
    // Find index of maximum point source energy required
    const int maxPtSrcIdx = find( my_srcErg,maxErg );

    // Get the energy vector for that point source energy
    //std::cout << "tally energy path is "
    //  << getTallyEnergyPath(maxPtSrcIdx) << std::endl;
    std::vector<double> erg = readbin(
        getTallyEnergyPath(maxPtSrcIdx)+"talerg.dat" );

    // Find maximum energy index
    const int nTempErg = static_cast<int>( erg.size() );
    int maxIdx = nTempErg-1;
    for ( int i=nTempErg-1;i>=0;--i )
    {
        if ( erg[i] < maxErg )
        {
            maxIdx = i+1;
            break;
        }
    }
    // erase extra elements
    erg.erase( erg.begin()+maxIdx+1,erg.end() );
    const int nerg = static_cast<int>( erg.size() );
    my_R.resize( nerg,nerg );
    //and make it the energy basis for our response
    my_R.ergin() = erg;
    my_R.ergout() = erg;

    // find index of smallest source energy,
    // we don't want to go below
    // that because then we would be extrapolating
    int minIdx = 1;
    for ( int i=0;i<nerg;++i )
    {
        if ( erg[i] > my_srcErg[0] )
        {
            minIdx = i;
            break;
        }
    }

    // build response by interpolating point source values
    // iterate over possible source energies
    // ( response matrix columns )
    // keep track of the last point source
    // index used for efficiency
    int lastidx = 0;
    // start on col=1 because col=0 is the energy cutoff
    global::init = 0.0;
    global::scat = 0.0;
    global::brem = 0.0;
    global::intrpscat = 0.0;
    global::intrpbrem = 0.0;
    global::intrpxray = 0.0;
    global::intrpannh = 0.0;
    global::intrpuncl = 0.0;
    global::intrp = 0.0;
    global::assign = 0.0;
    for ( int col=1;col<my_R.numergin();++col )
    {
        int ptSrcIdx1 = 0;
        int ptSrcIdx3 = 1;
        // find two closest point source energies
        for ( int e=lastidx;e<maxPtSrcIdx;++e )
        {
            if ( my_srcErg[e] < my_R.ergin(col)
```

```
                && my_srcErg[e+1] >= my_R.ergin(col) )
            {
                ptSrcIdx1 = e;
                ptSrcIdx3 = e+1;
                lastidx = e;
                break;
            }
        }

        // generate resulting interpolated spectrum
        //std::cout << "interpolating talles" << std::endl;
        std::vector< datapoint > intrpResult;
        try
        {
            // calculate geometric mean energy in this bin
            double meanErg = sqrt(my_R.ergin(col)*my_R.ergin(col-1));
            intrpResult = interpolateTallies( meanErg,
                my_R.ergout(),ptSrcIdx1,ptSrcIdx3 );
        }
        catch ( warning &w )
        {
            w.PrintError();
            continue;
        }



        // assign interpolated spectrum to every row in this column
        const int nErg = static_cast<int>( intrpResult.size() );
        timer assign;
        assign.start();
        for ( int row=1;row<nErg;++row )
        {
            my_R(row-1,col-1) = intrpResult[row];
        }
        assign.stop();
        global::assign += assign.getTime();
        //std::cout << "done in energy loop" << std::endl;
    }
    //std::cout << "leaving submodel" << std::endl;
    //my_R.optimize();

//std::cout << "done building response, time breakdown:" << std::endl;
//std::cout << "\t Initialization: " << global::init << std::endl;
//std::cout << "\t Scatter:        " << global::scat << std::endl;
//std::cout << "\t Bremsstrahlung: " << global::brem << std::endl;
//std::cout << "\t Interp Scatter: " << global::intrpscat << std::endl;
//std::cout << "\t Interp Brem:    " << global::intrpbrem << std::endl;
//std::cout << "\t Interp X-Ray:   " << global::intrpxray << std::endl;
//std::cout << "\t Interp Annih:   " << global::intrpannh << std::endl;
//std::cout << "\t Interp Uncoll:  " << global::intrpuncl << std::endl;
//std::cout << "\t Interp Total:   " << global::intrp << std::endl;
//std::cout << "\t Assignment:     " << global::assign << std::endl;
    return;
}
```

Listing B.52: submodel.hpp

```
#ifndef _submodel_hpp_included_
#define _submodel_hpp_included_
#include <string>
#include <vector>
#include "datapoint.hpp"
#include "response.hpp"
#include "tally.hpp"
#include <tr1/memory>

class submodel
{
    public:

    typedef std::tr1::shared_ptr<ptally> tallyPtr;

    response getR() const { return my_R; };
    void clear() { my_R.clear(); };
    void setR( const response& R ) { my_R = R; };
    void scaleBy( double scaler ) { my_R.scaleBy(scaler); };

    protected:

    virtual std::string getTallyEnergyPath( int ){ return ""; };
    virtual std::vector<datapoint> interpolateTallies(
        double,const std::vector< double >&,int,int )
    {
        std::vector<datapoint> a;
        return a;
    }


    void computeResponse( double );

    response my_R;

    std::vector< double > my_srcErg;


    std::vector < datapoint >& data ( );
    std::vector < datapoint > data ( ) const;

    std::vector < datapoint > my_data;

    std::string my_datapath;

    int my_redFact;


};

#endif
```

Listing B.53: tally.cpp

```
#include "tally.hpp"
#include "extras.hpp"
#include "fileio.hpp"
#include "phys.hpp"

double tallybase::erg( int e ) const
{
    return my_erg[e];
}

std::vector< double > tallybase::erg( ) const
{
    return my_erg;
}

std::vector< double >& tallybase::erg( )
{
    return my_erg;
}

double ntally::time( int t ) const
{
    return my_tim[t];
}

std::vector< double > ntally::time( ) const
{
    return my_tim;
}

std::vector< double >& ntally::time( )
{
    return my_tim;
}


std::vector< datapoint > ntally2::createData(
    const std::vector<double>& val,
    const std::vector<double>& err )
{
    if ( val.size() != err.size() )
    {
        std::cout << "val.size() = " << val.size() << std::endl;
        std::cout << "err.size() = " << err.size() << std::endl;
        throw fatal_error("non-matching tally value/error arrays");
    }

    std::vector< datapoint > result(nErg()*nSrcErg()*nTime());
    for ( int e=0;e<nErg();++e )
    {
        for ( int se=0;se<nSrcErg()-1;++se )
        {
            for ( int t=0;t<nTime()-1;++t )
            {
                const int idx1 = t+1+nTime()*(se+1+nSrcErg()*e);
                const int idx2 = t+(nTime()-1)*(se+(nSrcErg()-1)*e);
                result.at(idx1) = datapoint(val.at(idx2),err.at(idx2));
            }
        }
    }

    for ( int e=0;e<nErg();++e )
    {
        for ( int se=0;se<nSrcErg();++se )
        {
            // add zero time bin
            result[nTime()*(se+nSrcErg()*e)] = datapoint(0.0,0.0);
        }
    }

    for ( int e=0;e<nErg();++e )
    {
        for ( int t=0;t<nTime();++t )
        {
            // add zero src erg bin
```

```
            result[t+nTime()*(nSrcErg()*e)] = datapoint(0.0,0.0);
        }
    }

    return result;
}

void ntally2::parse( const std::string& talErgPath,
    const std::string& talPath )
{
    my_talErgPath = talErgPath;
    my_talPath = talPath;
    my_erg = readbin( talErgPath+"talerg.dat" );
    my_tim = readbin( talErgPath+"taltime.dat" );
    my_srcErg = readbin( talErgPath+"srcerg.dat" );

    // add 0 time bin
    my_tim.insert(my_tim.begin(),0.0);

    // get tally values/errors
    my_data = createData( readbin(talPath+"tal.dat"),
        readbin(talPath+"err.dat") );
}

ntally2 ntally2::interpolate(double x1,ntally2 y1,double x2,
    ntally2 y2,double x)
{
    const int TE = y1.nErg();
    const int TT = y1.nTime();
    const int SE = y1.nSrcErg();
    ntally2 result(SE,TE,TT);
    result.setErg(y1.erg());
    result.setTime(y1.time());
    result.setSrcErg(y1.srcErg());
    for ( int se=0;se<SE;++se )
    {
        for ( int te=0;te<TE;++te )
        {
            for ( int tt=0;tt<TT;++tt )
            {
                result.val(se,te,tt) = linearInterpolate( x1,
                    y1.val(se,te,tt),x2,y2.val(se,te,tt),x );
            }
        }
    }
    return result;
}

void ntally2::operator= ( const ntally2& t )
{
    my_erg = t.erg();
    my_srcErg = t.srcErg();
    my_tim = t.time();
    my_data = t.val();
}



matrix< datapoint > ntally::createData(
    const std::vector<double>& val,
    const std::vector<double>& err )
{
    if ( val.size() != err.size() )
    {
        std::cout << "val.size() = " << val.size() << std::endl;
        std::cout << "err.size() = " << err.size() << std::endl;
        throw fatal_error("non-matching tally value/error arrays");
    }

    matrix< datapoint > result(nErg(),nTime());
    for ( int e=0;e<nErg();++e )
    {
        result(e,0) = datapoint(0.0,0.0); // add zero time bin
        for ( int t=0;t<nTime()-1;++t )
        {
            const int idx = t+(nTime()-1)*e;
```

```
            result(e,t+1) = datapoint(val[idx],err[idx]);
        }
    }
    return result;
}

void ntally::parse( const std::string& talErgPath,
    const std::string& talPath )
{
    my_talErgPath = talErgPath;
    my_talPath = talPath;

    my_erg = readbin( talErgPath+"talerg.dat" );
    my_tim = readbin( talErgPath+"taltime.dat" );

    // add 0 time bin
    my_tim.insert(my_tim.begin(),0.0);

    // get tally values/errors
    my_val = createData( readbin(talPath+"tal.dat"),
        readbin(talPath+"err.dat") );
}

ntally ntally::loginterpolate(double x1,const ntally& y1,
    double x2,const ntally& y2,double x)
{
    const int E = y1.nErg();
    const int T = y1.nTime();
    ntally result(E,T);
    result.setErg(y1.erg());
    result.setTime(y1.time());
    for ( int e=0;e<E;++e )
    {
        for ( int t=0;t<T;++t )
        {
            result.setVal( e,t, logInterpolate(x1,y1.val(e,t),
                x2,y2.val(e,t),x) );
        }
    }
    return result;
}

void ntally::interpolate(double x1,ntally y1,double x2,
    ntally y2,double x,double thisx)
{
    const int E = y1.nErg();
    const int T = y1.nTime();

    for ( int e=0;e<E;++e )
    {
        // find peak time index
        int thismaxIdx = val().getColMaxIdx(e);
        int maxIdx1 = y1.val().getColMaxIdx(e);
        int maxIdx2 = y2.val().getColMaxIdx(e);
        // get max times
        double thismax = time(thismaxIdx);
        double max1 = y1.time(maxIdx1);
        double max2 = y2.time(maxIdx2);
        // find approximate new time
        double newmax = linearInterpolate(x1,max1,x2,max2,x);
        // find index of new max
        int newIdx = 0;
        for ( int t=0;t<T-1;++t )
        {
            if ( newmax > y1.time(t) && newmax <= y1.time(t+1) )
            {
                newIdx = t+1;
            }
        }
        // find shifts to newmax
        int dt = newIdx-thismaxIdx;
        int dt1 = newIdx-maxIdx1;
        int dt2 = newIdx-maxIdx2;
        // shift data
        if ( dt > 0 )
        {
```

```
            for ( int t=T-dt-1;t>=0;--t )
            {
                val(e,t+dt) = val(e,t);
            }
        }
        else if ( dt < 0 )
        {
            dt = dt*(-1);
            for ( int t=dt;t<T;++t )
            {
                val(e,t-dt) = val(e,t);
            }
        }
        if ( dt1 > 0 )
        {
            for ( int t=T-dt1-1;t>=0;--t )
            {
                y1.val(e,t+dt1) = y1.val(e,t);
            }
        }
        else if ( dt1 < 0 )
        {
            dt1 = dt1*(-1);
            for ( int t=dt1;t<T;++t )
            {
                y1.val(e,t-dt1) = y1.val(e,t);
            }
        }
        if ( dt2 > 0 )
        {
            for ( int t=T-dt2-1;t>=0;--t )
            {
                y2.val(e,t+dt2) = y2.val(e,t);
            }
        }
        else if ( dt2 < 0 )
        {
            dt2 = dt2*(-1);
            for ( int t=dt2;t<T;++t )
            {
                y2.val(e,t-dt2) = y2.val(e,t);
            }
        }

        for ( int t=0;t<T;++t )
        {
            val(e,t) = val(e,t) + (x-thisx)*(y2.val(e,t)
                -y1.val(e,t))/(x2-x1);
        }
    }
}


ntally ntally::interpolate(double x1,ntally y1,double x2,
    ntally y2,double x)
{
    const int E = y1.nErg();
    const int T = y1.nTime();
    ntally result(E,T);
    result.setErg(y1.erg());
    result.setTime(y1.time());

    for ( int e=0;e<E;++e )
    {
        for ( int t=0;t<T;++t )
        {
            result.setVal( e,t, linearInterpolate(x1,
                y1.val(e,t),x2,y2.val(e,t),x) );
        }
    }
    return result;
}

void ntally::operator= ( const ntally& t )
{
    my_erg = t.erg();
```

```
    my_tim = t.time();
    my_val = t.val();
}

datapoint ntally::sum( )
{
    datapoint result;
    for ( int e=0;e<nErg();++e )
    {
        for ( int t=0;t<nTime();++t )
        {
            result = result + val(e,t);
        }
    }
    return result;
}

void ntally::scaleBy( double a )
{
    my_val = my_val * a;
}

void ptally::setSourceEnergy( double serg )
{
    srcerg = serg;
    srcergidx = find( my_erg,srcerg );
    comperg = phys::scaterg( srcerg,phys::pi );
    compidx = find( my_erg,comperg );
}

std::vector< datapoint > ptally::createData(
    const std::vector<double>& erg,
    const std::vector<double>& val,
    const std::vector<double>& err )
{
    if ( val.size() != err.size() || val.size() != erg.size() )
    {
        throw fatal_error("non-matching tally \
energy/value/error arrays");
    }


    std::vector< datapoint > result;
    //
    // need to resize result
    // and change energy spectrum too for this to work
    //
    if ( my_redFact != 1 )
    {
        const int nFullBin = static_cast<int>(
            floor(static_cast<double>(val.size())/my_redFact));
        const int remainder = val.size() % my_redFact;
        const int nPartBin = remainder > 0 ? 1 : 0;
        result.resize( nFullBin+nPartBin );
        my_erg.resize( nFullBin+nPartBin );
        // first bin should always be zero
        result[0].set(0.0);
        result[0].setErr(0.0);
        my_erg[0] = erg[0];
        // do full bins
        const int N = my_redFact;
        for ( int i=1;i<nFullBin;++i )
        {
            my_erg.at(i) = erg.at(i*N);
            double t = 0;
            double e = 0;
            for ( int j=i*N;j>(i-1)*N;--j )
            {
                t += val.at(j);
                e += pow(val.at(j)*err.at(j),2.0);
            }
            result.at(i).set( t/(my_erg.at(i)-my_erg.at(i-1)) );
            result.at(i).setErr( sqrt(e)/t );
        }
        // do partial bin
        if ( nPartBin > 0 )
```

```
        {
            if ( nFullBin*N+remainder !=
                static_cast<int>(val.size()) || nPartBin > 1 )
            {
                throw fatal_error("something's wrong with \
reduction bin sizes");
            }
            my_erg.back() = erg.back();
            double t = 0;
            double e = 0;
            for ( unsigned int i=nFullBin*N;i<val.size();++i )
            {
                t += val.at(i);
                e += pow(val.at(i)*err.at(i),2.0);
            }
            result.back().set( t/(my_erg.back()
                -my_erg.at(my_erg.size()-2)) );
            result.back().setErr( sqrt(e)/t );
        }
    }
    else
    {
        my_erg = erg;
        result.resize( val.size() );
        // first bin should always be zero
        result[0].set(0.0);
        result[0].setErr(0.0);
        for ( unsigned int i=1;i<val.size();++i )
        {
            result[i].set( val[i]/(my_erg[i]
                -my_erg[i-1]) ); // make data per MeV
            result[i].setErr( err[i] );
        }
    }

    return result;

}

bool ptally::isDataOkay( const std::vector<datapoint>& vec )
{
    const unsigned int I = vec.size();
    for ( unsigned int i=0;i<I;++i )
    {
        if ( isnan( vec[i].get() ) )
        {
            throw fatal_error("bad data (NAN)");
        }
        if ( isinf( vec[i].get() ) )
        {
            throw fatal_error("bad data (INF)");
        }
    }
    return true;
}

ptally::ptally( )
{
    my_redFact = 1;
}

int ptally::getRedFact( )
{
    return my_redFact;
}

int ptally::getRedFact( ) const
{
    return my_redFact;
}

void ptally::print( )
{

}
```

```
void tally::print( )
{
    for ( unsigned int i=0;i<my_tal.size();++i )
    {
        std::cout << my_erg[i] << "_"
            << my_tal[i].get() << std::endl;
    }
}

void tallytag::parse( const std::string& talErgPath,
    const std::string& talPath )
{
    my_talErgPath = talErgPath;
    my_talPath = talPath;

    try
    {
        // get tally values/errors
        //std::cout << "\tparsing file "
        //<< talPath+"taluncl.dat" << std::endl;
        my_taluncl = createData(
            readbin( (talErgPath+"talerg.dat").c_str() ),
            readbin((talPath+"taluncl.dat").c_str()),
            readbin((talPath+"erruncl.dat").c_str()) );
        isDataOkay( my_taluncl );
        //std::cout << "\tparsing file "
        //<< talPath+"talbrem.dat" << std::endl;
        my_talbrem = createData(
            readbin( (talErgPath+"talerg.dat").c_str() ),
            readbin((talPath+"talbrem.dat").c_str()),
            readbin((talPath+"errbrem.dat").c_str()) );
        isDataOkay( my_talbrem );
        //std::cout << "\tparsing file "
        //<< talPath+"talxray.dat" << std::endl;
        my_talxray = createData(
            readbin( (talErgPath+"talerg.dat").c_str() ),
            readbin((talPath+"talxray.dat").c_str()),
            readbin((talPath+"errxray.dat").c_str()) );
        isDataOkay( my_talxray );
        //std::cout << "\tparsing file "
        //<< talPath+"talannh.dat" << std::endl;
        my_talannh = createData(
            readbin( (talErgPath+"talerg.dat").c_str() ),
            readbin((talPath+"talannh.dat").c_str()),
            readbin((talPath+"errannh.dat").c_str()) );
        isDataOkay( my_talannh );
        //std::cout << "\tparsing file "
        //<< talPath+"talscat.dat" << std::endl;
        my_talscat = createData(
            readbin( (talErgPath+"talerg.dat").c_str() ),
            readbin((talPath+"talscat.dat").c_str()),
            readbin((talPath+"errscat.dat").c_str()) );
        isDataOkay( my_talscat );
    }
    catch ( fatal_error& fe )
    {
        throw fatal_error( fe.getError()
            + "_in_file_" + my_talPath );
    }
}

tallytag::tallytag( const std::string& talErgPath,
    const std::string& talPath )
{
    parse( talErgPath,talPath );
}

void tallytag::operator= ( tallytag& tal )
{
    std::cout << "in_tallytag_operator=" << std::endl;
    my_erg = tal.erg();
    my_taluncl = tal.talUncl();
    my_talbrem = tal.talBrem();
    my_talxray = tal.talXray();
    my_talannh = tal.talAnnh();
    my_talscat = tal.talScat();
```

```
    srcerg = tal.srcerg;
    srcergidx = tal.srcergidx;
    comperg = tal.comperg;
    compidx = tal.compidx;
}

void tallytag::operator= ( std::tr1::shared_ptr<ptally>& _tal_ )
{
    std::cout << "in_tallytag_operator=" << std::endl;
    std::tr1::shared_ptr<tallytag> tal
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal_ );
    if ( ! tal )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tallytag");
    }
    my_erg = tal->erg();
    my_taluncl = tal->talUncl();
    my_talbrem = tal->talBrem();
    my_talxray = tal->talXray();
    my_talannh = tal->talAnnh();
    my_talscat = tal->talScat();

    srcerg = tal->srcerg;
    srcergidx = tal->srcergidx;
    comperg = tal->comperg;
    compidx = tal->compidx;
}

void tallytag::interpolate( tallyPtr _tal1_,
    tallyPtr _tal2_,double tal1val,double tal2val,
    double newval,std::string type )
{
    std::tr1::shared_ptr<tallytag> tal1
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_ );
    std::tr1::shared_ptr<tallytag> tal2
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal2_ );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could_not_dynamic_\
cast_ptally_to_tallytag");
    }
    if ( tal1->erg().size() != tal2->erg().size()
        || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching_tallytag_\
classes,_can't_interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_taluncl.resize( I );
    my_talbrem.resize( I );
    my_talxray.resize( I );
    my_talannh.resize( I );
    my_talscat.resize( I );
    srcerg = tal1->srcerg;
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    if ( type.compare("log") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_taluncl[i] = logInterpolate( tal1val,
                tal1->talUncl(i),tal2val,tal2->talUncl(i),newval );
            my_talbrem[i] = logInterpolate( tal1val,
                tal1->talBrem(i),tal2val,tal2->talBrem(i),newval );
            my_talxray[i] = logInterpolate( tal1val,
                tal1->talXray(i),tal2val,tal2->talXray(i),newval );
            my_talannh[i] = logInterpolate( tal1val,
                tal1->talAnnh(i),tal2val,tal2->talAnnh(i),newval );
            my_talscat[i] = logInterpolate( tal1val,
                tal1->talScat(i),tal2val,tal2->talScat(i),newval );
        }
    }
    else if ( type.compare("lin") == 0 )
```

```
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_taluncl[i] = linearInterpolate( tal1val,
                tal1->talUncl(i),tal2val,tal2->talUncl(i),newval );
            my_talbrem[i] = linearInterpolate( tal1val,
                tal1->talBrem(i),tal2val,tal2->talBrem(i),newval );
            my_talxray[i] = linearInterpolate( tal1val,
                tal1->talXray(i),tal2val,tal2->talXray(i),newval );
            my_talannh[i] = linearInterpolate( tal1val,
                tal1->talAnnh(i),tal2val,tal2->talAnnh(i),newval );
            my_talscat[i] = linearInterpolate( tal1val,
                tal1->talScat(i),tal2val,tal2->talScat(i),newval );
        }
    }
    else
    {
        throw fatal_error("unknown tally interpolation type "+type );
    }
}


void tallytag::interpolate( tallyPtr _tal1_,
    tallyPtr _tal2_,tallyPtr _tal1_2,tallyPtr _tal2_2,
    double tal1val,double tal2val,double tal1val2,
    double tal2val2,double newval,double newval2,std::string type )
{
    std::tr1::shared_ptr<tallytag> tal1
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_ );
    std::tr1::shared_ptr<tallytag> tal2
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal2_ );
    std::tr1::shared_ptr<tallytag> tal12
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_2 );
    std::tr1::shared_ptr<tallytag> tal22
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal2_2 );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could not dynamic cast \
ptally to tallytag");
    }
    if ( tal1->erg().size() != tal2->erg().size()
        || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching tallytag classes, \
can't interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_taluncl.resize( I );
    my_talbrem.resize( I );
    my_talxray.resize( I );
    my_talannh.resize( I );
    my_talscat.resize( I );
    srcerg = tal1->srcerg;
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    if ( type.compare("log") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_taluncl[i] = logInterpolate( tal1val,
                tal1->talUncl(i),tal2val,tal2->talUncl(i),newval );
            my_talbrem[i] = logInterpolate( tal1val2,
                tal12->talBrem(i),tal2val2,tal22->talBrem(i),newval2 );
            my_talxray[i] = logInterpolate( tal1val2,
                tal12->talXray(i),tal2val2,tal22->talXray(i),newval2 );
            my_talannh[i] = logInterpolate( tal1val2,
                tal12->talAnnh(i),tal2val2,tal22->talAnnh(i),newval2 );
            my_talscat[i] = logInterpolate( tal1val2,
                tal12->talScat(i),tal2val2,tal22->talScat(i),newval2 );
        }
    }
    else if ( type.compare("lin") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
```

```
        {
            my_taluncl[i] = linearInterpolate( tal1val,
                tal1->talUncl(i),tal2val,tal2->talUncl(i),newval );
            my_talbrem[i] = linearInterpolate( tal1val2,
                tal12->talBrem(i),tal2val2,tal22->talBrem(i),newval2 );
            my_talxray[i] = linearInterpolate( tal1val2,
                tal12->talXray(i),tal2val2,tal22->talXray(i),newval2 );
            my_talannh[i] = linearInterpolate( tal1val2,
                tal12->talAnnh(i),tal2val2,tal22->talAnnh(i),newval2 );
            my_talscat[i] = linearInterpolate( tal1val2,
                tal12->talScat(i),tal2val2,tal22->talScat(i),newval2 );
        }
    }
    else
    {
        throw fatal_error("unknown tally interpolation type " + type );
    }
}


void tallytag::interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
    double tal1x,double tal1y,double tal1z,double tal2x,
    double tal2y,double tal2z,double newx,double newy,double newz)
{
    std::tr1::shared_ptr<tallytag> tal1
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_ );
    std::tr1::shared_ptr<tallytag> tal2
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal2_ );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could not dynamic cast \
ptally to tallytag");
    }
    if ( tal1->erg().size() != tal2->erg().size()
         || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching tallytag classes, \
can't interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_taluncl.resize( I );
    my_talbrem.resize( I );
    my_talxray.resize( I );
    my_talannh.resize( I );
    my_talscat.resize( I );
    srcerg = tal1->srcerg;
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    for ( unsigned int i=0;i<I;++i )
    {
        const double dist = ( (tal1x-newx)/(tal2x-tal1x)
            + (tal1y-newy)/(tal2y-tal1y)
            + (tal1z-newz)/(tal2z-tal1z) );
         my_taluncl[i] = dist*(tal2->talUncl(i)-tal1->talUncl(i));
         my_talbrem[i] = dist*(tal2->talBrem(i)-tal1->talBrem(i));
         my_talxray[i] = dist*(tal2->talXray(i)-tal1->talXray(i));
         my_talannh[i] = dist*(tal2->talAnnh(i)-tal1->talAnnh(i));
         my_talscat[i] = dist*(tal2->talScat(i)-tal1->talScat(i));
    }
}

void tallytag::interpolate( tallyPtr _tal0_,
    tallyPtr _tal1_,tallyPtr _tal2_,tallyPtr _tal3_,
    double tal0y,double tal0z,
    double tal1y,double tal1z,
    double tal2y,double tal2z,
    double tal3y,double tal3z,
    double newy,double newz )
{
    std::tr1::shared_ptr<tallytag> tal0
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal0_ );
    std::tr1::shared_ptr<tallytag> tal1
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal1_ );
    std::tr1::shared_ptr<tallytag> tal2
```

```
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal2_ );
    std::tr1::shared_ptr<tallytag> tal3
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal3_ );
    if ( ! tal0 || ! tal1 || ! tal2 || ! tal3 )
    {
        throw fatal_error("could not dynamic cast \
ptally to tallytag");
    }
    if ( tal0->erg().size() != tal1->erg().size() ||
        tal1->erg().size() != tal2->erg().size() ||
        tal2->erg().size() != tal3->erg().size() )
    {
        throw fatal_error("non-matching tally classes, \
can't interpolate");
    }

    const unsigned int I = tal0->erg().size();
    my_erg = tal0->erg();
    my_taluncl.resize( I );
    my_talbrem.resize( I );
    my_talxray.resize( I );
    my_talannh.resize( I );
    my_talscat.resize( I );
    srcerg = tal0->srcerg;
    srcergidx = tal0->srcergidx;
    comperg = tal0->comperg;
    compidx = tal0->compidx;
    std::vector<datapoint> uncl01y(I);
    std::vector<datapoint> brem01y(I);
    std::vector<datapoint> xray01y(I);
    std::vector<datapoint> annh01y(I);
    std::vector<datapoint> scat01y(I);
    std::vector<datapoint> uncl23y(I);
    std::vector<datapoint> brem23y(I);
    std::vector<datapoint> xray23y(I);
    std::vector<datapoint> annh23y(I);
    std::vector<datapoint> scat23y(I);
    const double dist01y = ( newy-tal0y )/( tal1y-tal0y );
    const double dist23y = ( newy-tal2y )/( tal3y-tal2y );
    for ( unsigned int i=0;i<I;++i )
    {
        uncl01y[i] = tal0->talUncl(i)
            + dist01y*( tal1->talUncl(i)-tal0->talUncl(i) );
        brem01y[i] = tal0->talBrem(i)
            + dist01y*( tal1->talBrem(i)-tal0->talBrem(i) );
        xray01y[i] = tal0->talXray(i)
            + dist01y*( tal1->talXray(i)-tal0->talXray(i) );
        annh01y[i] = tal0->talAnnh(i)
            + dist01y*( tal1->talAnnh(i)-tal0->talAnnh(i) );
        scat01y[i] = tal0->talScat(i)
            + dist01y*( tal1->talScat(i)-tal0->talScat(i) );

        uncl23y[i] = tal2->talUncl(i)
            + dist23y*( tal3->talUncl(i)-tal2->talUncl(i) );
        brem23y[i] = tal2->talBrem(i)
            + dist23y*( tal3->talBrem(i)-tal2->talBrem(i) );
        xray23y[i] = tal2->talXray(i)
            + dist23y*( tal3->talXray(i)-tal2->talXray(i) );
        annh23y[i] = tal2->talAnnh(i)
            + dist23y*( tal3->talAnnh(i)-tal2->talAnnh(i) );
        scat23y[i] = tal2->talScat(i)
            + dist23y*( tal3->talScat(i)-tal2->talScat(i) );
    }
    const double distz = ( newz - tal0z )/( tal2z-tal0z );
    for ( unsigned int i=0;i<I;++i )
    {
        my_taluncl[i]
            = uncl01y[i] + distz * ( uncl23y[i]-uncl01y[i] );
        my_talbrem[i]
            = brem01y[i] + distz * ( brem23y[i]-brem01y[i] );
        my_talxray[i]
            = xray01y[i] + distz * ( xray23y[i]-xray01y[i] );
        my_talannh[i]
            = annh01y[i] + distz * ( annh23y[i]-annh01y[i] );
        my_talscat[i]
            = scat01y[i] + distz * ( scat23y[i]-scat01y[i] );
```

```
    }
}


void tallytag::makeZero( tallyPtr _tal_ )
{
    std::tr1::shared_ptr<tallytag> tal
        = std::tr1::dynamic_pointer_cast<tallytag>( _tal_ );
    if ( ! tal )
    {
        throw fatal_error("could not dynamic cast \
ptally to tallytag");
    }
    const unsigned int I = tal->erg().size();
    my_erg = tal->erg();
    my_taluncl.resize( I );
    my_talbrem.resize( I );
    my_talxray.resize( I );
    my_talannh.resize( I );
    my_talscat.resize( I );
    srcerg = tal->srcerg;
    srcergidx = tal->srcergidx;
    comperg = tal->comperg;
    compidx = tal->compidx;
    for ( unsigned int i=0;i<I;++i )
    {
        my_taluncl[i].zero();
        my_talbrem[i].zero();
        my_talxray[i].zero();
        my_talannh[i].zero();
        my_talscat[i].zero();
    }
}


void tallytag::scaleBy( double factor )
{
    const unsigned int I = my_erg.size();
    for ( unsigned int i=0;i<I;++i )
    {
        my_taluncl[i] = my_taluncl[i]*factor;
        my_talbrem[i] = my_talbrem[i]*factor;
        my_talxray[i] = my_talxray[i]*factor;
        my_talannh[i] = my_talannh[i]*factor;
        my_talscat[i] = my_talscat[i]*factor;
    }
}

datapoint tallytag::sum( )
{
    const unsigned int I = my_erg.size();
    datapoint sum(0.0,0.0);
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_taluncl[i] * Ebin;
        sum = sum + my_talbrem[i] * Ebin;
        sum = sum + my_talxray[i] * Ebin;
        sum = sum + my_talannh[i] * Ebin;
        sum = sum + my_talscat[i] * Ebin;
    }
    return sum;
}


datapoint tallytag::sumUncl( )
{
    const unsigned int I = my_erg.size();
    datapoint sum;
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_taluncl[i] * Ebin;
    }
    return sum;
```

```
}

datapoint tallytag::sumBrem( )
{
    const unsigned int I = my_erg.size();
    datapoint sum;
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_talbrem[i] * Ebin;
    }
    return sum;
}

datapoint tallytag::sumXray( )
{
    const unsigned int I = my_erg.size();
    datapoint sum;
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_talxray[i] * Ebin;
    }
    return sum;
}

datapoint tallytag::sumAnnh( )
{
    const unsigned int I = my_erg.size();
    datapoint sum;
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_talannh[i] * Ebin;
    }
    return sum;
}

datapoint tallytag::sumScat( )
{
    const unsigned int I = my_erg.size();
    datapoint sum;
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_talscat[i] * Ebin;
    }
    return sum;
}

void tallytag::scaleByParts(
    const std::vector<double>& factor )
{
    const unsigned int I = my_erg.size();
    for ( unsigned int i=0;i<I;++i )
    {
        my_taluncl[i] = my_taluncl[i] * factor[0];
        my_talbrem[i] = my_talbrem[i] * factor[1];
        my_talxray[i] = my_talxray[i] * factor[2];
        my_talannh[i] = my_talannh[i] * factor[3];
        my_talscat[i] = my_talscat[i] * factor[4];
    }
}

std::vector<double> tallytag::sumParts( )
{
    const unsigned int I = my_erg.size();
    std::vector<double> sum(5,0.0);
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum[0] += my_taluncl[i].get() * Ebin;
        sum[1] += my_talbrem[i].get() * Ebin;
        sum[2] += my_talxray[i].get() * Ebin;
        sum[3] += my_talannh[i].get() * Ebin;
        sum[4] += my_talscat[i].get() * Ebin;
```

```
    }
    return sum;
}

void tally::parse( const std::string& talErgPath,
    const std::string& talPath )
{
    my_talErgPath = talErgPath;
    my_talPath = talPath;

    // get tally values/errors
    my_tal = createData(
        readbin( (talErgPath+"talerg.dat").c_str() ),
        readbin((talPath+"tal.dat").c_str()),
        readbin((talPath+"err.dat").c_str()) );
    checkData();
}

tally::tally( const std::string& talErgPath,
    const std::string& talPath )
{
    parse( talErgPath,talPath );
}

void tally::operator= ( const tally& tal )
{
    my_erg = tal.erg();
    my_tal = tal.tal();

    srcerg = tal.srcerg;
    srcergidx = tal.srcergidx;
    comperg = tal.comperg;
    compidx = tal.compidx;
    my_redFact = tal.getRedFact( );
}

void tally::operator= ( std::tr1::shared_ptr<ptally>& _tal_ )
{
    std::tr1::shared_ptr<tally> tal
        = std::tr1::dynamic_pointer_cast<tally>( _tal_ );
    if ( ! tal )
    {
        throw fatal_error("could not dynamic cast ptally to tally");
    }
    my_erg = tal->erg();
    my_tal = tal->tal();

    srcerg = tal->srcerg;
    srcergidx = tal->srcergidx;
    comperg = tal->comperg;
    compidx = tal->compidx;
    my_redFact = tal->getRedFact( );
}


void tally::interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
    double tal1val,double tal2val,double newval,std::string type )
{
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal2
        = std::tr1::dynamic_pointer_cast<tally>( _tal2_ );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could not dynamic cast ptally to tally");
    }
    if ( tal1->erg().size() != tal2->erg().size()
        || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching tallytag classes, \
can't interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_tal.resize( I );
    srcerg = tal1->srcerg;
```

```
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    if ( type.compare("log") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_tal[i] = logInterpolate( tal1val,
                tal1->tal(i),tal2val,tal2->tal(i),newval );
        }
    }
    else if ( type.compare("lin") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_tal[i] = linearInterpolate( tal1val,
                tal1->tal(i),tal2val,tal2->tal(i),newval );
        }
    }
    else
    {
        throw fatal_error("unknown_interpolation_type_" + type);
    }
}


void tally::interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
    tallyPtr _tal1_2,tallyPtr _tal2_2,double tal1val,
        double tal2val,double tal1val2,double tal2val2,
        double newval,double newval2,std::string type )
{
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal2
        = std::tr1::dynamic_pointer_cast<tally>( _tal2_ );
    std::tr1::shared_ptr<tally> tal1_2
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_2 );
    std::tr1::shared_ptr<tally> tal2_2
        = std::tr1::dynamic_pointer_cast<tally>( _tal2_2 );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tally");
    }
    if ( tal1->erg().size() != tal2->erg().size()
        || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching_tallytag_classes,_\
can't_interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_tal.resize( I );
    srcerg = tal1->srcerg;
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    if ( type.compare("log") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_tal[i] = logInterpolate( tal1val,
                tal1->tal(i),tal2val,tal2->tal(i),newval );
        }
    }
    else if ( type.compare("lin") == 0 )
    {
        for ( unsigned int i=0;i<I;++i )
        {
            my_tal[i] = linearInterpolate( tal1val,
                tal1->tal(i),tal2val,tal2->tal(i),newval );
        }
    }
    else
    {
        throw fatal_error("unknown_interpolation_type_" + type);
    }
```

```
}


void tally::interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
    double tal1x,double tal1y,double tal1z,double tal2x,
    double tal2y,double tal2z,double newx,double newy,double newz)
{
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal2
        = std::tr1::dynamic_pointer_cast<tally>( _tal2_ );
    if ( ! tal1 || ! tal2 )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tally");
    }
    if ( tal1->erg().size() != tal2->erg().size()
        || fabs(tal1->srcerg - tal2->srcerg) > 1e-20 )
    {
        throw fatal_error("non-matching_tallytag_classes,_\
can't_interpolate");
    }
    const unsigned int I = tal1->erg().size();
    my_erg = tal1->erg();
    my_tal.resize( I );
    srcerg = tal1->srcerg;
    srcergidx = tal1->srcergidx;
    comperg = tal1->comperg;
    compidx = tal1->compidx;
    for ( unsigned int i=0;i<I;++i )
    {
        double dist = 0;
        if ( fabs(tal2x-tal1x) > 1e-5 )
        {
            dist+= (tal1x-newx)/(tal2x-tal1x);
        }
        if ( fabs(tal2y-tal1y) > 1e-5 )
        {
            dist+= (tal1y-newy)/(tal2y-tal1y);
        }
        if ( fabs(tal2z-tal1z) > 1e-5 )
        {
            dist+= (tal1z-newz)/(tal2z-tal1z);
        }
        my_tal[i] = tal1->tal(i)
            + ( dist )*(tal2->tal(i)-tal1->tal(i));
    }
}


// note that the z-component can actually be the
// x-direction, it's just here to help
// comprehend how the interpolation works
void tally::interpolate( tallyPtr _tal0_,
    tallyPtr _tal1_,tallyPtr _tal2_,tallyPtr _tal3_,
    double tal0y,double tal0z,
    double tal1y,double tal1z,
    double tal2y,double tal2z,
    double tal3y,double tal3z,
    double newy,double newz )
{
    std::tr1::shared_ptr<tally> tal0
        = std::tr1::dynamic_pointer_cast<tally>( _tal0_ );
    std::tr1::shared_ptr<tally> tal1
        = std::tr1::dynamic_pointer_cast<tally>( _tal1_ );
    std::tr1::shared_ptr<tally> tal2
        = std::tr1::dynamic_pointer_cast<tally>( _tal2_ );
    std::tr1::shared_ptr<tally> tal3
        = std::tr1::dynamic_pointer_cast<tally>( _tal3_ );
    if ( ! tal0 || ! tal1 || ! tal2 || ! tal3 )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tally");
    }
    if ( tal0->erg().size() != tal1->erg().size() ||
        tal1->erg().size() != tal2->erg().size() ||
        tal2->erg().size() != tal3->erg().size() )
```

```
    {
        throw fatal_error("non-matching_tally_classes,_\
can't_interpolate");
    }

    const unsigned int I = tal0->erg().size();
    my_erg = tal0->erg();
    my_tal.resize( I );
    srcerg = tal0->srcerg;
    srcergidx = tal0->srcergidx;
    comperg = tal0->comperg;
    compidx = tal0->compidx;
    std::vector<datapoint> tal01y(I);
    std::vector<datapoint> tal23y(I);
    const double dist01y = ( newy-tal0y )/( tal1y-tal0y );
    const double dist23y = ( newy-tal2y )/( tal3y-tal2y );
    for ( unsigned int i=0;i<I;++i )
    {
        tal01y[i] = tal0->tal(i)
            + dist01y*( tal1->tal(i)-tal0->tal(i) );
        tal23y[i] = tal2->tal(i)
            + dist23y*( tal3->tal(i)-tal2->tal(i) );
    }
    const double distz = ( newz - tal0z )/( tal2z-tal0z );
    for ( unsigned int i=0;i<I;++i )
    {
        my_tal[i] = tal01y[i] + distz * ( tal23y[i]-tal01y[i] );
    }
}


tally tally::interpolate(double x1,const tally& y1,
    double x2,const tally& y2,double x)
{
    const int E = y1.nErg();
    tally result;
    result.erg() = y1.erg();
    result.tal().resize(E);
    result.srcerg = y1.srcerg;
    result.srcergidx = y1.srcergidx;
    result.comperg = y1.comperg;
    result.compidx = y1.compidx;
    for ( int e=0;e<E;++e )
    {
        result.tal(e)
            = linearInterpolate(x1,y1.tal(e),x2,y2.tal(e),x);
    }
    return result;
}


void tally::scaleBy( double factor )
{
    const unsigned int I = my_tal.size();
    for ( unsigned int i=0;i<I;++i )
    {
        my_tal[i] = my_tal[i]*factor;
    }
}

datapoint tally::sum( )
{
    const unsigned int I = my_tal.size();
    datapoint sum(0.0,0.0);
    for ( unsigned int i=1;i<I;++i )
    {
        const double Ebin = erg(i)-erg(i-1);
        sum = sum + my_tal[i]*Ebin;
    }
    return sum;
}

void tally::makeZero( tallyPtr _tal_ )
{
    std::tr1::shared_ptr<tally> tal
        = std::tr1::dynamic_pointer_cast<tally>( _tal_ );
```

```
    if ( ! tal )
    {
        throw fatal_error("could_not_dynamic_cast_ptally_to_tally");
    }
    const unsigned int I = tal->erg().size();
    my_erg = tal->erg();
    my_tal.resize( I );
    srcerg = tal->srcerg;
    srcergidx = tal->srcergidx;
    comperg = tal->comperg;
    compidx = tal->compidx;
    for ( unsigned int i=0;i<I;++i )
    {
        my_tal[i].zero();
    }
}

void tallytag::checkData( )
{
    const unsigned int I = my_erg.size();
    for ( unsigned int i=0;i<I;++i )
    {
        double x[5] = {my_taluncl[i].get(),
            my_talbrem[i].get(),
            my_talxray[i].get(),
            my_talannh[i].get(),
            my_talscat[i].get()};
        for ( int j=0;j<5;++j )
        {
            if ( x[j] < 0.0 || isinf(x[j]) || isnan(x[j]) )
            {
                throw fatal_error("in_checkData_\
unreasonably_data_("+str(x[j])+")");
                break;
            }
        }
    }
}

void tally::checkData( )
{
    const unsigned int I = my_erg.size();
    for ( unsigned int i=0;i<I;++i )
    {
        if ( my_tal[i].get() < 0.0
            || isinf(my_tal[i].get())
            || isnan(my_tal[i].get()) )
        {
            throw fatal_error("in_checkData_unreasonable_\
data_("+str(my_tal[i].get())+")");
            break;
        }
    }
    //int zerosum = 0;
    //for ( unsigned int i=0;i<I;++i )
    //{
        //if ( my_tal[i].get() < std::numeric_limits<double>::min() )
        //{
            //zerosum++;
        //}
    //}
    //std::cout << " for energy " << srcerg << "    "
    //  << ((double)zerosum)/((double)I)
    //  << "%  out of " << I << std::endl;
}

void tally::integrateBinWidth( )
{
    for ( unsigned int i=1;i<my_erg.size();++i )
    {
        my_tal[i] = my_tal[i]*(my_erg[i]-my_erg[i-1]);
    }
}
```

Listing B.54: tally.hpp

```
#ifndef _tally_hpp_
#define _tally_hpp_

#include <cstdlib>
#include <vector>
#include <string>
#include <fstream>
#include <tr1/memory>

#include "errh.hpp"
#include "datapoint.hpp"
#include "matrix.hpp"

class tallybase
{
    public:

    double erg( int ) const;
    std::vector< double > erg( ) const;
    std::vector< double >& erg( );
    int nErg( ) { return static_cast<int>(my_erg.size()); };
    int nErg( ) const { return static_cast<int>(my_erg.size()); };


    protected:

    std::vector< double > my_erg;
    std::string my_talErgPath;
    std::string my_talPath;

};


class ntally : public tallybase
{
    public:

    double time( int t ) const;
    std::vector< double > time( ) const;
    std::vector< double >& time( );

    matrix< datapoint >& val() { return my_val; };
    matrix< datapoint > val() const { return my_val; };
    datapoint val(int e,int t) const { return my_val(e,t); };
    datapoint& val(int e,int t) { return my_val(e,t); };

    void setVal( int i,int j,const datapoint& a ) { my_val(i,j) = a; };
    void setErg( int idx,double a ) { my_erg[idx] = a; };
    void setErg( const std::vector<double>& a ) { my_erg = a; };
    void setTime( int idx,double a ) { my_tim[idx] = a; };
    void setTime( const std::vector<double>& a ) { my_tim = a; };

    int nTime( ) { return static_cast<int>(my_tim.size()); };
    int nTime( ) const { return static_cast<int>(my_tim.size()); };

    virtual void parse( const std::string& talErgPath,
        const std::string& talPath );

    void interpolate(double x1,ntally y1,double x2,
        ntally y2,double x,double thisx);
    static ntally interpolate( double,ntally,
        double x1,ntally,double );
    static ntally loginterpolate( double,const ntally&,
        double x1,const ntally&,double );

    datapoint sum( );

    void scaleBy( double );

    void operator= ( const ntally& );

    ntally ( ) { };

    ntally( int nErg,int nTime )
    {
```

```
        my_val.resize(nErg,nTime);
        my_erg.resize(nErg);
        my_tim.resize(nTime);
    }

    protected:

    matrix< datapoint > my_val;
    std::vector< double > my_tim;

    private:

    matrix< datapoint > createData(
        const std::vector<double>& val,
        const std::vector<double>& err );

};


class ntally2 : public ntally
{
    public:

    datapoint val( int se,int te,int tt ) const
    {
        return my_data[ tt+nTime()*(se+nSrcErg()*te) ];
    }

    datapoint& val( int se,int te,int tt )
    {
        return my_data[ tt+nTime()*(se+nSrcErg()*te) ];
    }

    int nSrcErg() const
    {
        return static_cast<int>(my_srcErg.size());
    }

    std::vector<double> srcErg() const
    {
        return my_srcErg;
    }

    void setSrcErg( const std::vector<double>& src )
    {
        my_srcErg = src;
    }

    std::vector<datapoint> val() const
    {
        return my_data;
    }

    static ntally2 interpolate( double,ntally2,
        double x1,ntally2,double );

    ntally2( int srcErg,int talErg,int talTime )
    {
        my_data.resize(srcErg*talErg*talTime);
        my_erg.resize(talErg);
        my_srcErg.resize(srcErg);
        my_tim.resize(talTime);
    }

    void parse( const std::string& talErgPath,
        const std::string& talPath );

    ntally2() {};

    void operator= ( const ntally2& );


    private:
    std::vector< datapoint > my_data;
    std::vector< double > my_srcErg;
```

```
    std::vector< datapoint > createData(
        const std::vector<double>& val,
        const std::vector<double>& err );

};




class ptally : public tallybase
{
public:
    typedef std::tr1::shared_ptr<ptally> tallyPtr;

    int getRedFact( );
    int getRedFact( ) const;

    virtual void parse( const std::string&,const std::string& ) = 0;
    virtual void setSourceEnergy( double );
    virtual void interpolate( tallyPtr,tallyPtr,
        double,double,double,std::string type="log" ) = 0;
    virtual void interpolate( tallyPtr _tal1_,
        tallyPtr _tal2_,tallyPtr _tal1_2,tallyPtr _tal2_2,
        double tal1val,double tal2val,double,
        double,double newval,double newval2,
        std::string type="log") = 0;
    virtual void interpolate( tallyPtr,tallyPtr,double,
        double,double,double,double,double,
        double,double,double ) = 0;
    virtual void interpolate( tallyPtr _tal0_,
        tallyPtr _tal1_,tallyPtr _tal2_,tallyPtr _tal3_,
        double tal0y,double tal0z,
        double tal1y,double tal1z,
        double tal2y,double tal2z,
        double tal3y,double tal3z,
        double newy,double newz ) = 0;

    virtual void scaleBy( double ) = 0;
    virtual datapoint sum( ) = 0;

    virtual void makeZero( tallyPtr ) = 0;
    virtual void checkData( ) = 0;

    bool isDataOkay( const std::vector<datapoint>& );

    double srcerg;
    int srcergidx;
    double comperg;
    int compidx;

    //virtual void operator= ( std::tr1::shared_ptr<ptally>& );

    virtual void print( );

    virtual std::string who( ) { return "ptally"; };

    ptally( );
    ptally( int redFact ) { my_redFact = redFact; };

protected:

    int my_redFact;

    std::vector< datapoint > createData(
        const std::vector<double>& erg,
        const std::vector<double>& val,
        const std::vector<double>& err );

};

class tallytag : public ptally
{
public:
    std::vector< datapoint > talUncl() const { return my_taluncl; };
    datapoint talUncl(int a) const { return my_taluncl[a]; };
```

```
    std::vector< datapoint >& talUncl() { return my_taluncl; };
    datapoint& talUncl(int a) { return my_taluncl[a]; };
    datapoint sumUncl( );

    std::vector< datapoint > talBrem() const { return my_talbrem; };
    datapoint talBrem(int a) const { return my_talbrem[a]; };
    std::vector< datapoint >& talBrem() { return my_talbrem; };
    datapoint& talBrem(int a) { return my_talbrem[a]; };
    datapoint sumBrem( );

    std::vector< datapoint > talXray() const { return my_talxray; };
    datapoint talXray(int a) const { return my_talxray[a]; };
    std::vector< datapoint >& talXray() { return my_talxray; };
    datapoint& talXray(int a) { return my_talxray[a]; };
    datapoint sumXray( );

    std::vector< datapoint > talAnnh() const { return my_talannh; };
    datapoint talAnnh(int a) const { return my_talannh[a]; };
    std::vector< datapoint >& talAnnh() { return my_talannh; };
    datapoint& talAnnh(int a) { return my_talannh[a]; };
    datapoint sumAnnh( );

    std::vector< datapoint > talScat() const { return my_talscat; };
    datapoint talScat(int a) const { return my_talscat[a]; };
    std::vector< datapoint >& talScat() { return my_talscat; };
    datapoint& talScat(int a) { return my_talscat[a]; };
    datapoint sumScat( );


    void parse( const std::string&,const std::string& );

    void operator= ( tallytag& );
    void operator= ( std::tr1::shared_ptr<ptally>& );
    void interpolate( tallyPtr,tallyPtr,double,double,
        double,std::string type="log" );
    void interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
        tallyPtr _tal1_2,tallyPtr _tal2_2,double tal1val,
        double tal2val,double,double,double newval,
        double newval2,std::string type="log");
    void interpolate( tallyPtr,tallyPtr,double,double,
        double,double,double,double,double,double,double );
    void interpolate( tallyPtr _tal0_,tallyPtr _tal1_,
        tallyPtr _tal2_,tallyPtr _tal3_,
    double tal0y,double tal0z,
    double tal1y,double tal1z,
    double tal2y,double tal2z,
    double tal3y,double tal3z,
    double newy,double newz );

    void scaleBy( double );
    void checkData( );
    datapoint sum( );
    void scaleByParts( const std::vector<double>& factor );
    std::vector<double> sumParts( );

    void makeZero( tallyPtr );

    virtual std::string who( ) { return "tallytag"; };


    tallytag( const std::string&,const std::string& );
    tallytag( int redFact ) : ptally( redFact ) { };
    tallytag( ) : ptally( ) {};
    ~tallytag( ) {};

protected:
    std::vector< datapoint > my_taluncl;
    std::vector< datapoint > my_talbrem;
    std::vector< datapoint > my_talxray;
    std::vector< datapoint > my_talannh;
    std::vector< datapoint > my_talscat;
};



class tally : public ptally
```

```
{
    public:

    std::vector< datapoint > tal() const { return my_tal; };
    datapoint tal(int a) const { return my_tal[a]; };
    std::vector< datapoint >& tal() { return my_tal; };
    datapoint& tal(int a) { return my_tal[a]; };

    virtual void parse( const std::string&,const std::string& );

    void operator= ( const tally& );
    void operator= ( std::tr1::shared_ptr<ptally>& );
    void interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
        double tal1val,double tal2val,double newval,
        std::string type="log");
    void interpolate( tallyPtr _tal1_,tallyPtr _tal2_,
        tallyPtr _tal1_2,tallyPtr _tal2_2,double tal1val,
        double tal2val,double,double,double newval,
        double newval2,std::string type="log");
    void interpolate( tallyPtr,tallyPtr,double,double,
        double,double,double,double,double,double,double );
    void interpolate( tallyPtr _tal0_,tallyPtr _tal1_,
        tallyPtr _tal2_,tallyPtr _tal3_,
    double tal0y,double tal0z,
    double tal1y,double tal1z,
    double tal2y,double tal2z,
    double tal3y,double tal3z,
    double newy,double newz );

    static tally interpolate( double,const tally&,
        double x1,const tally&,double );


    void scaleBy( double );
    datapoint sum( );

    void makeZero( tallyPtr );

    void checkData( );


    void integrateBinWidth( );

    void print( );

    virtual std::string who( ) { return "tally"; };

    tally( const std::string&,const std::string& );
    tally( int redFact ) : ptally ( redFact ) { };
    tally( ) : ptally( ) {};
    ~tally( ) {};

private:
    std::vector<datapoint> my_tal;

};

#endif
```

Listing B.55: terrestrial.cpp

```
#include "terrestrial.hpp"
#include <fstream>
#include <iostream>
#include "extras.hpp"

#include <phys.hpp>

double terrestrial::getMaxErg( )
{
    return my_source.lasterg();
}

spectrum terrestrial::readDataFile ( const std::string& path )
{
    std::ifstream infile ( path.c_str() );

    double density,depth,radius,mass,specificactivity,gammaactivity;
    int numergs;

    std::string tempString;
    infile >> tempString; infile >> density;
    infile >> tempString; infile >> depth;
    infile >> tempString; infile >> radius;
    infile >> tempString; infile >> mass;
    my_mass.push_back ( mass );
    infile >> tempString; infile >> specificactivity;
    my_specact.push_back ( specificactivity );
    infile >> tempString; infile >> gammaactivity;
    my_gamact.push_back ( gammaactivity );
    infile >> tempString; infile >> numergs;

    spectrum result ( numergs );
//std::cout << "reading data file " << path << std::endl;
    double energy,tally,error;
    infile >> energy;
    result.erg(0) = energy;
    for ( int i=1;i<numergs;++i )
    {
        //std::cout << i << std::endl;
        infile >> energy;
        infile >> tally;
        infile >> error;

        result.erg ( i ) = energy;
        result ( i-1 ).set ( tally );
        result ( i-1 ).setErr ( error );
    }
    //std::cout << "done" << std::endl;
    return result;
}



void terrestrial::readDataFiles ( )
{
    my_usoil = readDataFile ( my_usoildatapath );
    my_uconc = readDataFile ( my_uconcdatapath );
    my_ksoil = readDataFile ( my_ksoildatapath );
    my_kconc = readDataFile ( my_kconcdatapath );
    my_thsoil = readDataFile ( my_thsoildatapath );
    my_thconc = readDataFile ( my_thconcdatapath );

    return;
}


void terrestrial::buildResponse ( double usoil,
                                  double uconc,
                                  double ksoil,
                                  double kconc,
                                  double thsoil,
                                  double thconc )

{
```

```
    const double usoilfac = my_mass[0] * usoil * my_gamact[0] / my_specact[0];
    const double uconcfac = my_mass[1] * uconc * my_gamact[1] / my_specact[1];
    const double ksoilfac = my_mass[2] * ksoil * my_gamact[2] / my_specact[2];
    const double kconcfac = my_mass[3] * kconc * my_gamact[3] / my_specact[3];
    const double thsoilfac = my_mass[4] * thsoil * my_gamact[4] / my_specact[4];
    const double thconcfac = my_mass[5] * thconc * my_gamact[5] / my_specact[5];

    my_source = my_usoil*usoilfac + my_uconc*uconcfac
              + my_ksoil*ksoilfac + my_kconc*kconcfac
              + my_thsoil*thsoilfac + my_thconc*thconcfac;

    // this is still per unit area
    // hard code in surface area used in ground-to-detector calculations
    const double area = phys::pi*4000*4000;

    my_source = my_source * area;


    return;
}

spectrum terrestrial::operator() ( )
{
    return my_source;
}

terrestrial::terrestrial ( const std::string& usoilpath,
                           const std::string& uconcpath,
                           const std::string& ksoilpath,
                           const std::string& kconcpath,
                           const std::string& thsoilpath,
                           const std::string& thconcpath )
{
    initialize ( usoilpath,uconcpath,ksoilpath,kconcpath,thsoilpath,thconcpath );
}

void terrestrial::initialize ( const std::string& usoilpath,
                               const std::string& uconcpath,
                               const std::string& ksoilpath,
                               const std::string& kconcpath,
                               const std::string& thsoilpath,
                               const std::string& thconcpath )
{
    my_usoildatapath = usoilpath;
    my_uconcdatapath = uconcpath;
    my_ksoildatapath = ksoilpath;
    my_kconcdatapath = kconcpath;
    my_thsoildatapath = thsoilpath;
    my_thconcdatapath = thconcpath;

    readDataFiles ( );
    return;
}
```

Listing B.56: terrestrial.hpp

```
#ifndef _terrestrial_hpp_included_
#define _terrestrial_hpp_included_
#include <string>
#include <vector>

#include "spectrum.hpp"
#include "submodel.hpp"


class terrestrial : public submodel
{
    public:

    double getMaxErg( );

    void buildResponse ( double,double,double,double,double,double );

    spectrum operator() ( );

    terrestrial ( const std::string&,
                  const std::string&,
                  const std::string&,
                  const std::string&,
                  const std::string&,
                  const std::string& );

    terrestrial ( ) { };

    void initialize ( const std::string&,
                      const std::string&,
                      const std::string&,
                      const std::string&,
                      const std::string&,
                      const std::string& );

    private:


    spectrum readDataFile ( const std::string& path );

    void readDataFiles ( );

    spectrum my_usoil;
    spectrum my_uconc;
    spectrum my_ksoil;
    spectrum my_kconc;
    spectrum my_thsoil;
    spectrum my_thconc;

    std::string my_usoildatapath;
    std::string my_uconcdatapath;
    std::string my_ksoildatapath;
    std::string my_kconcdatapath;
    std::string my_thsoildatapath;
    std::string my_thconcdatapath;

    spectrum my_source;


    std::vector < double > my_mass;
    std::vector < double > my_specact;
    std::vector < double > my_gamact;


};


#endif
```

Listing B.57: tresponse.cpp

```
#include "tresponse.hpp"
#include "phys.hpp"



int tresponse::nTime( ) const
{
    return static_cast<int>(my_time.size());
}

double tresponse::getTime(int t) const
{
    return my_time[t];
}

std::vector<double> tresponse::getTime( ) const
{
    return my_time;
}

std::vector<double> tresponse::getErg( ) const
{
    return my_response[0].ergout();
}

std::vector<double> tresponse::getErgIn( ) const
{
    return my_response[0].ergin();
}

std::vector<double> tresponse::getErgOut( ) const
{
    return my_response[0].ergout();
}

int tresponse::nErg( ) const
{
    return my_response[0].numergout();
}

response tresponse::getResponse(int t) const
{
    return my_response[t];
}

void tresponse::setResponse( int t,const response& a )
{
    my_response[t]=a;
}

void tresponse::addResponse( int t,const response& a )
{
    my_response[t]=my_response[t]+a;
}

response& tresponse::operator() (int t)
{
    return my_response[t];
}

void tresponse::setTime( const std::vector<double>& time )
{
    my_time = time; my_response.resize(time.size()-1);
}

void tresponse::setErg( const std::vector<double>& ergin,
    const std::vector<double>& ergout )
{
    for ( int i=0;i<nTime()-1;++i )
    {
        my_response[i].initialize(ergin,ergout);
    }
}
```

```
double tresponse::sum( )
{
    double mysum = 0.0;
    for ( int i=0;i<nTime()-1;++i )
    {
        mysum = mysum + my_response[i].sum();
    }
    return mysum;
}

void tresponse::scaleBy( double scaler )
{
    for ( int i=0;i<nTime()-1;++i )
    {
        my_response[i].scaleBy(scaler);
    }
}



void tresponse::identity( )
{
    for ( int t=0;t<nTime()-1;++t )
    {
        my_time[t] = 0.0;
        my_response[t].identity();
    }
}

tresponse tresponse::inverse( )
{
    tresponse result;
    std::vector<double> time(nTime());
    for ( int t=0;t<nTime();++t )
    {
        time[t] = (-1.0)*my_time[t];
    }
    result.setTime( time );
    for ( int t=0;t<nTime()-1;++t )
    {
        result.setResponse(t,my_response[t].inverse());
    }
    return result;
}


void tresponse::shiftDistance( double dist )
{
    // assumes ergin/ergout are same for each response matrix
    //
    // compute time shift for each energy bin
    //
    std::vector<double> erg = getErg();
    // compute velocity for each bin
    std::vector<double> v( erg.size()-1 );
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        v[e] = phys::velocity( sqrt(erg[e]*erg[e+1]),phys::mn );
        //std::cout << "v[" << e << "] = " << v[e] << std::endl;
    }
    // compute time to traverse the distance in shakes
    std::vector<double> dt( erg.size()-1 );
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        dt[e] = dist/v[e]*1e8;
        //std::cout << "dt[" << e << "] = " << dt[e] << std::endl;
    }
    for ( int e=0;e<nErg()-1;++e )
    {
        int idx = 0;
        for ( int t=0;t<nTime()-1;++t )
        {
            if ( dt[e] > getTime(t) && dt[e] <= getTime(t+1) )
            {
                idx = t;
```

```
                break;
            }
        }

        my_response[idx](e,e) = my_response[idx](e,e) + 1.0;
    }
}


tspectrum operator * ( const tresponse& resp,const tspectrum& spec )
{
    // time bins for total spectrum is arbitrary, but no information
    // is gained over the resolution of the response functions
    tspectrum total(resp.getTime(),resp.getErg());
    // loop over spectrum times
    // st = spectrum time
    const double eps = 1e-20;
    for ( int st=0;st<spec.nTime()-1;++st )
    {
        if ( spec.getSpectrum(st).sum() > eps )
        {
            // get min/max times of original spectrum
            const double minSTime = spec.getTime(st);
            const double maxSTime = spec.getTime(st+1);

            // loop over response times
            // rt = response time
            for ( int rt=0;rt<resp.nTime()-1;++rt )
            {
                if ( resp.getResponse(rt).sum() > eps )
                {
                    spectrum temp = resp.getResponse(rt)
                        * spec.getSpectrum(st);

                    // get min/max times this response puts it in
                    const double minRTime = resp.getTime(rt);
                    const double maxRTime = resp.getTime(rt+1);

                    // can only assume uniformity
                    // over time bin, new min/max is
                    const double minTime = minSTime+minRTime;
                    const double maxTime = maxSTime+maxRTime;

                    // average time over bin
                    const double avgTime = (minTime+maxTime)/2.0;
                    for ( int t=0;t<total.nTime()-1;++t )
                    {
                        if ( avgTime > total.getTime(t)
                            && avgTime <= total.getTime(t+1) )
                        {
                            total.addSpectrum(t,temp);
                            break;
                        }
                    }

                    //// calculate weights for each of
                    //// the total spectrum's
                    //// time bins corresponding to
                    //// the fraction of the
                    //// outgoing time bin they cover
                    //std::vector<int> idx;
                    //std::vector<double> w;
                    //for ( int t=0;t<total.nTime()-1;++t )
                    //{
                        //if ( minTime > total.getTime(t)
                            //&& minTime <= total.getTime(t+1) )
                        //{
                            //double lb = minTime;
                            //for ( int tt=t;tt<total.nTime()-1;++tt )
                            //{
                                //if ( maxTime
                                //  <= total.getTime(tt+1) )
                                //{
                                    //idx.push_back(tt);
                                    //w.push_back((maxTime-lb)
```

```
                                //   /(maxTime-minTime));
                                //break;
                            //}
                            //else
                            //{
                                //idx.push_back(tt);
                                //w.push_back(
                                //   (total.getTime(tt+1)-lb)
                                //   /(maxTime-minTime));
                                //lb = total.getTime(tt+1);
                            //}
                        //}
                        //break;
                    //}
                //}
                ////std::cout << " w    idx " << std::endl;
                //for ( unsigned int t=0;t<idx.size();++t )
                //{
                    ////std::cout << w[t]
                    //  << "    " << idx[t] << std::endl;
                    //total.addSpectrum(idx[t],w[t]*temp);
                //}
                ////std::cout << std::endl;
            }
        }
    }
    }
    return total;
}


tspectrum operator * ( const tresponse& resp,const spectrum& spec )
{
    // time bins for total spectrum is arbitrary, but no information
    // is gained over the resolution of the response functions
    tspectrum total(resp.getTime(),resp.getErg());
    // loop over response times
    // rt = response time
    for ( int rt=0;rt<resp.nTime()-1;++rt )
    {
        spectrum temp = resp.getResponse(rt)*spec;

        // get min/max times this response puts it in
        const double minTime = resp.getTime(rt);
        const double maxTime = resp.getTime(rt+1);
        // calculate weights for each of the total spectrum's
        // time bins corresponding to the fraction of the
        // outgoing time bin they cover
        std::vector<int> idx;
        std::vector<double> w;
        for ( int t=0;t<total.nTime()-1;++t )
        {
            if ( minTime > total.getTime(t)
                && minTime <= total.getTime(t+1) )
            {
                double lb = minTime; // lower bound of bin
                for ( int tt=t;tt<total.nTime();++tt )
                {
                    if ( maxTime <= total.getTime(tt) )
                    {
                        idx.push_back(tt);
                        w.push_back((maxTime-lb)/(maxTime-minTime));
                        break;
                    }
                    else
                    {
                        idx.push_back(tt);
                        w.push_back((total.getTime(tt)-lb)
                            /(maxTime-minTime));
                        lb = total.getTime(tt);
                    }
                }
                break;
            }
        }
        for ( unsigned int t=0;t<idx.size();++t )
```

```
        {
            total.addSpectrum(idx[t],w[t]*temp);
        }
    }

    return total;
}


tresponse operator * ( const tresponse& tresp,double scale )
{
    tresponse result;
    result.setTime( tresp.getTime() );
    for ( int t=0;t<tresp.nTime()-1;++t )
    {
        result.setResponse(t,tresp.getResponse(t)*scale);
    }
    return result;
}

tresponse operator + ( const tresponse& a,const tresponse& b )
{
    tresponse result;
    result.setTime( a.getTime() );
    for ( int t=0;t<a.nTime()-1;++t )
    {
        result.setResponse(t,a.getResponse(t)+b.getResponse(t));
    }
    return result;
}

tresponse operator - ( const tresponse& a,const tresponse& b )
{
    tresponse result;
    result.setTime( a.getTime() );
    for ( int t=0;t<a.nTime()-1;++t )
    {
        result.setResponse(t,a.getResponse(t)-b.getResponse(t));
    }
    return result;
}


tresponse operator * ( const tresponse& a,const tresponse& b )
{
    tresponse result;
    result.setTime( a.getTime() );


    //// loop over spectrum times
    //// st = spectrum time
    //const double eps = 1e-20;
    //for ( int st=0;st<spec.nTime()-1;++st )
    //{
        //if ( spec.getSpectrum(st).sum() > eps )
        //{
            //// get min/max times of original spectrum
            //const double minSTime = spec.getTime(st);
            //const double maxSTime = spec.getTime(st+1);

            //// loop over response times
            //// rt = response time
            //for ( int rt=0;rt<resp.nTime()-1;++rt )
            //{
                //if ( resp.getResponse(rt).sum() > eps )
                //{
                    //spectrum temp = resp.getResponse(rt)
                        //* spec.getSpectrum(st);

                    //// get min/max times this response puts it in
                    //const double minRTime = resp.getTime(rt);
                    //const double maxRTime = resp.getTime(rt+1);

                    //// can only assume uniformity
                    //// over time bin, new min/max is
                    //const double minTime = minSTime+minRTime;
```

```
            //const double maxTime = maxSTime+maxRTime;

            //// average time over bin
            //const double avgTime = (minTime+maxTime)/2.0;
            //for ( int t=0;t<total.nTime()-1;++t )
            //{
                //if ( avgTime > total.getTime(t)
                    //&& avgTime <= total.getTime(t+1) )
                //{
                    //total.addSpectrum(t,temp);
                //}
            //}
          //}
        //}
      //}
    //}
    //return total;

    result.setErg( a.getErgIn(),a.getErgOut() );

    for ( int at=0;at<a.nTime()-1;++at )
    {
        for ( int bt=0;bt<b.nTime()-1;++bt )
        {
            // average time over bin
            const double minTime = a.getTime(at)+b.getTime(bt);
            const double maxTime = a.getTime(at+1)+b.getTime(bt+1);
            const double avgTime = (minTime+maxTime)/2.0;
            for ( int t=0;t<result.nTime()-1;++t )
            {
                if ( avgTime > result.getTime(t)
                    && avgTime <= result.getTime(t+1) )
                {
                    //total.addSpectrum(t,temp);
                    result.addResponse(t,a.getResponse(at)
                        *b.getResponse(bt));
                    break;
                }
            }
            //result.setResponse(a.getTime(t)+b.getTime(t),
            //    a.getResponse(t)*b.getResponse(t));
        }
    }
    return result;
}
```

Listing B.58: tresponse.hpp

```
#ifndef _tresponse_hpp_included_
#define _tresponse_hpp_included_

#include "response.hpp"
#include "tspectrum.hpp"

class tresponse
{
    public:


    int nTime( ) const;
    double getTime(int t) const;
    std::vector<double> getTime( ) const;
    std::vector<double> getErg( ) const;
    std::vector<double> getErgIn( ) const;
    std::vector<double> getErgOut( ) const;
    int nErg( ) const;
    response getResponse(int t) const;
    void setResponse( int t,const response& a );
    void addResponse( int t,const response& a );

    response& operator() (int t);

    void shiftDistance( double dist );

    void identity();

    tresponse inverse();

    void setTime( const std::vector<double>& time );
    void setErg( const std::vector<double>& ergin,
        const std::vector<double>& ergout );

    double sum( );

    void scaleBy( double scaler );

    private:

    std::vector<double> my_time;
    std::vector< response > my_response;

};


tspectrum operator * ( const tresponse&,const tspectrum& );

tspectrum operator * ( const tresponse&,const spectrum& );

tresponse operator * ( const tresponse&,double );

tresponse operator + ( const tresponse&,const tresponse& );

tresponse operator - ( const tresponse&,const tresponse& );

tresponse operator * ( const tresponse&,const tresponse& );

#endif
```

Listing B.59: tspectrum.cpp

```
#include "tspectrum.hpp"
#include "phys.hpp"

int tspectrum::nTime( ) const
{
    return static_cast<int>(my_time.size());
}

double tspectrum::getTime( int t ) const
{
    return my_time[t];
}

std::vector<double> tspectrum::getTime( ) const
{
    return my_time;
}

std::vector<double> tspectrum::getErg( ) const
{
    if ( nTime() > 0 )
    {
        return my_spectrum[0].erg();
    }
    else
    {
        return std::vector<double>(0);
    }
}

int tspectrum::nErg( ) const
{
    if ( nTime() > 0 )
    {
        return static_cast<int>(my_spectrum[0].numerg());
    }
    else
    {
        return 0;
    }
}

std::vector<double> tspectrum::getData( ) const
{
    // make sure all energies are the same
    for ( int i=0;i<nTime()-2;++i )
    {
        if ( my_spectrum[i].numerg() != my_spectrum[i+1].numerg() )
        {
            return std::vector<double>(0);
        }
    }

    const int E = my_spectrum[0].numerg();
    const int T = nTime();
    std::vector<double> result( (E-1)*(T-1) );
    for ( int e=0;e<E-1;++e )
    {
        for ( int t=0;t<T-1;++t )
        {
            result[t+(T-1)*e] = my_spectrum[t](e).get();
        }
    }
    return result;
}

std::vector<double> tspectrum::getErr( ) const
{
    // make sure all energies are the same
    for ( int i=0;i<nTime()-2;++i )
    {
        if ( my_spectrum[i].numerg() != my_spectrum[i+1].numerg() )
        {
            return std::vector<double>(0);
        }
```

```
    }
    const int E = my_spectrum[0].numerg();
    const int T = nTime();
    std::vector<double> result( (E-1)*(T-1) );
    for ( int e=0;e<E-1;++e )
    {
        for ( int t=0;t<T-1;++t )
        {
            result[t+(T-1)*e] = my_spectrum[t](e).getErr();
        }
    }
    return result;
}

spectrum tspectrum::getSpectrum( int t ) const
{
    return my_spectrum[t];
}

void tspectrum::addSpectrum(int idx,const spectrum& S)
{
    my_spectrum[idx] = my_spectrum[idx] + S;
}

void tspectrum::setSpectrum(int idx,const spectrum& S)
{
    my_spectrum[idx] = S;
}

tspectrum::tspectrum( int t )
{
    my_time.resize(t);
    my_spectrum.resize(t-1);
}

tspectrum::tspectrum( const std::vector<double>& time )
{
    my_time = time;
    my_spectrum.resize(time.size()-1);
}

tspectrum::tspectrum( const std::vector<double>& time,
    const std::vector<double>& erg )
{
    my_time = time;
    my_spectrum.resize(time.size()-1);
    for ( unsigned int i=0;i<time.size()-1;++i )
    {
        my_spectrum[i].initialize(erg);
    }
}

void tspectrum::operator= ( const ntally& tal )
{
    my_time = tal.time();
    my_spectrum.resize(tal.nTime()-1);
    // want to ignore first time and energy bins (they are zero)
    for ( int t=0;t<tal.nTime()-1;++t )
    {
        my_spectrum[t].resize(tal.nErg());
        my_spectrum[t].erg() = tal.erg();
        for ( int e=0;e<tal.nErg()-1;++e )
        {
            my_spectrum[t](e) = tal.val(e+1,t+1);
        }
    }
}


tspectrum::tspectrum( const spectrum& a,const std::vector<double>& time )
{
    my_time = time;
    my_spectrum.resize(time.size()-1);
    spectrum tempspec = a;
    tempspec = tempspec * (1.0/static_cast<double>(time.size()-1));
    for ( unsigned int t=0;t<time.size()-1;++t )
```

```
    {
        my_spectrum[t] = a*( (time[t+1]-time[t])/(time.back()-time[0]) );
    }
}

spectrum tspectrum::toSpectrum() const
{
    spectrum result( getErg() );
    for ( int t=0;t<nTime()-1;++t )
    {
        result = result + my_spectrum[t];
    }
    return result;
}


tspectrum operator+(const tspectrum& a,const tspectrum& b )
{
    tspectrum result(a.getTime(),a.getErg());
    for ( int t=0;t<a.nTime()-1;++t )
    {
        spectrum temp = a.getSpectrum(t)+b.getSpectrum(t);
        result.setSpectrum(t,temp);
    }
    return result;
}

tspectrum operator*(const tspectrum& a,double b )
{
    tspectrum result(a.getTime(),a.getErg());
    for ( int t=0;t<a.nTime()-1;++t )
    {
        spectrum temp = a.getSpectrum(t)*b;
        result.setSpectrum(t,temp);
    }
    return result;
}

tspectrum shiftDistance( const tspectrum& a,double dist )
{
    // compute time shift for each energy bin
    //
    std::vector<double> erg = a.getErg();
    // compute velocity for each bin
    std::vector<double> v( erg.size()-1 );
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        v[e] = phys::velocity( sqrt(erg[e]*erg[e+1]),phys::mn );
        //std::cout << "v[" << e << "] = " << v[e] << std::endl;
    }
    // compute time to traverse the distance in shakes
    std::vector<double> dt( erg.size()-1 );
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        dt[e] = dist/v[e]*1e8;
        //std::cout << "dt[" << e << "] = " << dt[e] << std::endl;
    }
    tspectrum result(a.getTime(),a.getErg());
    for ( int e=0;e<a.nErg()-1;++e )
    {
        for ( int at=0;at<a.nTime()-1;++at )
        {
            const double minTime = a.getTime(at) + dt[e];
            const double maxTime = a.getTime(at+1) + dt[e];

            std::vector<int> idx;
            std::vector<double> w;
            for ( int t=0;t<result.nTime()-1;++t )
            {
                if ( minTime > result.getTime(t)
                   && minTime <= result.getTime(t+1) )
                {
                    double lb = minTime; // lower bound of bin
                    for ( int tt=t;tt<result.nTime()-1;++tt )
                    {
                        if ( maxTime <= result.getTime(tt+1) )
```

```
                        {
                            idx.push_back(tt);
                            w.push_back((maxTime-lb)/(maxTime-minTime));
                            break;
                        }
                        else
                        {
                            idx.push_back(tt);
                            w.push_back((result.getTime(tt+1)-lb)
                                /(maxTime-minTime));
                            lb = result.getTime(tt+1);
                        }
                    }
                    break;
                }
            }
            for ( unsigned int t=0;t<idx.size();++t )
            {
                result(e,idx[t]) += w[t]*a(e,at);
            }
        }

    }
    return result;
}
```

Listing B.60: tspectrum.hpp

```
#ifndef _tspectrum_hpp_included_
#define _tspectrum_hpp_included_

#include <vector>
#include "spectrum.hpp"


class tspectrum
{
    public:

    int nTime( ) const;
    double getTime(int t) const;
    std::vector<double> getTime( ) const;
    std::vector<double> getErg( ) const;
    int nErg( ) const;
    std::vector<double> getData( ) const;
    std::vector<double> getErr( ) const;
    spectrum getSpectrum(int t) const;

    void setSpectrum(int idx,const spectrum& S);

    void addSpectrum(int idx,const spectrum& S);

    tspectrum( int t );
    tspectrum( const std::vector<double>& time );
    tspectrum( const std::vector<double>& time,
        const std::vector<double>& erg );

    void operator= ( const ntally& tal );

    spectrum toSpectrum() const;

    datapoint operator() (int e,int t) const
    {
        return my_spectrum[t](e);
    }
    datapoint& operator() (int e,int t)
    {
        return my_spectrum[t](e);
    }

    tspectrum( ) { };

    tspectrum( const spectrum&,const std::vector<double>& );

    double sum( ) const
    {
        double mysum = 0.0;
        for ( int i=0;i<nTime()-1;++i )
        {
            mysum = mysum + my_spectrum[i].sum();
        }
        return mysum;
    }



    private:

    std::vector<double> my_time;
    std::vector<spectrum> my_spectrum;

};

tspectrum operator+(const tspectrum& a,const tspectrum& b );

tspectrum shiftDistance( const tspectrum& a,double dist );

tspectrum operator*(const tspectrum& a,double b );

#endif
```

Listing B.61: xpass.cpp

```cpp
//! @mainpage eXpedited Parametric Analysis of Smuggling Scenarios (XPASS)
//! @version 3.0
//! @author Gregory G. Thoreson ( thoreson.greg@gmail.com )
//! @date 05/2011
//!
//! This software package is designed to allow fast evaluation of
//! threat scenarios by use of decomposition and parameterization.
//! A threat scenario is broken up into well-defined subspaces called
//! submodels. The radiation transport within these submodels is
//! simulated with some code package (in this case, MCNPX). XPASS
//! reads the results from these simulations, generates response
//! functions based on the data, and convolves the functions to
//! re-build the full threat scenario. It also uses parameterization,
//! or interpolation, to sample variables continuously.

#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <math.h>
#include <ctime>
#include <memory>
#include "parser.hpp"
#include "inpgen.hpp"
#include "errh.hpp"
#include "data.hpp"
#include "model.hpp"
#include "alarm.hpp"
#include "fileio.hpp"
#include "extras.hpp"
#include "phys.hpp"
#include "nsnm.hpp"
#include "diagnostic.hpp"
#include "benchmark.hpp"

#include <time.h>

//! @brief the main function called at runtime
//!
//! calls the parser to read the input file, then
//! builds a model based on the input, passes the
//! information to an alarm algorithm to compute
//! evasion probabilities
int main ( int argc, const char* argv[] )
{
    clock_t start2 = std::clock();

    try
    {
        std::stringstream errorstream;
        errorstream << "incorrect format. arguments are:"
        << std::endl
        << "      [options]  [inputfile]  " << std::endl
        << std::endl
        << "  options: " << std::endl
        << "    -s   suppress output to console " << std::endl
        << "    -d   dump most data to file " << std::endl
        << "    -g   generate mcnp input decks \
(no input file specified)" << std::endl
        << "    -p   parse mctal files (no input file specified)"
        << std::endl
        << "    -t   benchmark SNM and detector \
models (no input file specified)" << std::endl
        << std::endl;

        std::string inputpath;
        bool consoleOutput = true;
        bool dataDump = false;
        bool quickRun = false;
        bool binaryInput = false;
        bool checkError = false;
        bool generateInput = false;
        bool parseOutput = false;
```

```
bool benchmark = false;

bool inputset = false;
for ( int i=1;i<argc;++i )
{
    if ( argv[i][0] == '-' )
    {
        if ( argv[i][1] == 's' )
            consoleOutput = false;
        else if ( argv[i][1] == 'd' )
            dataDump = true;
        else if ( argv[i][1] == 'q' )
            quickRun = true;
        else if ( argv[i][1] == 'g' )
            generateInput = true;
        else if ( argv[i][1] == 'p' )
            parseOutput = true;
        else if ( argv[i][1] == 't' )
            benchmark = true;
        else
            throw fatal_error ( "unknown option" );
    }
    else if ( ! inputset )
    {
        inputset = true;
        inputpath = argv[i];
    }
}

if ( generateInput )
{
    return generateInputDecks( );
}
else if ( parseOutput )
{
    return parseOutputFiles( );
}
else if ( benchmark )
{
    return benchmarkData( );
}

std::string outputPath = "data_dump/" + inputpath + "/";
if ( dataDump)
{
    mkdir( outputPath );
}

// parse input file
//
input::data data;
data.parse ( inputpath );

// integrate submodels by superposition
//
model mymodel ( data );
mymodel.setConsoleOutput( consoleOutput );
mymodel.setDataDump ( dataDump );
mymodel.setDataPath ( outputPath );
mymodel.setQuickRun ( quickRun );
mymodel.build ( );

// print statistics and evasion probability
//
if ( consoleOutput )
{
    std::cout << std::endl << std::endl
    << "  Total computation time: " << std::setw ( 20 )
    << std::setiosflags ( std::ios::scientific )
    << std::setprecision ( 3 )
    << ( ( std::clock() - start2 ) / ( double ) CLOCKS_PER_SEC )
    << " seconds " << std::endl << std::endl << std::endl;

    global::warn.flush();
    global::fatal.flush();
}
```

```
    }
    catch ( fileiowarn& w )
    {
        w.PrintError();
        return 1;
    }
    catch ( fatal_error& e )
    {
        e.PrintError();
        return 1;
    }
    catch ( std::bad_alloc )
    {
        std::cout << "fatal error: memory allocation failed" << std::endl;
        return 1;
    }
    return 0;
}
```

Listing B.62: xsec.cpp

```
#include "xsec.hpp"
#include "fileio.hpp"
#include "extras.hpp"
#include <phys.hpp>

// MT Number Cheat Sheet:
//
// Photons:
// MT   Reaction
// 501  Total
// 504  Incoherent Scattering
// 502  Coherent Scattering
// 522  Photoelectric Absorption
// 516  Pair Production (in nuclear field)
//
// Neutrons:
// MT   Reaction
// 1    Total
// 2    Elastic Scattering
// 102  Radiative Capture
//

neutronxsec::neutronxsec( int Z,int A,double rho )
{
    initialize(Z,A,rho);
}

neutronxsec::neutronxsec( const std::vector<int>& Z,
    const std::vector<int>& A,const std::vector<double>& wo,
    double rho )
{
    initialize(Z,A,wo,rho);
}

void neutronxsec::initialize( int Z,int A,double rho )
{
    my_Z.resize(1);
    my_A.resize(1);
    my_wo.resize(1);
    my_Z[0] = Z;
    my_A[0] = A;
    my_wo[0] = 1.0;
    my_rho = rho;
    my_datapath = "data/xsec/neutron/";
    readData( );
}

void neutronxsec::initialize( const std::vector<int>& Z,
    const std::vector<int>& A,const std::vector<double>& wo,
    double rho )
{
    my_Z = Z;
    my_A = A;
    my_wo = wo;
    my_rho = rho;
    my_datapath = "data/xsec/neutron/";
    readData( );
}

std::vector<double> neutronxsec::extractEnergy(
    const std::vector<double>& rawdata )
{
    std::vector<double> result( rawdata.size()/2 );
    for ( unsigned int i=0;i<result.size();++i )
    {
        result[i] = rawdata[2*i]/1e6; // convert eV to MeV
        //std::cout << result[i] << std::endl;
    }
    return result;
}

std::vector<double> neutronxsec::extractXSec(
    const std::vector<double>& rawdata )
{
    std::vector<double> result( rawdata.size()/2 );
```

```
    for ( unsigned int i=0;i<result.size();++i )
    {
        result[i] = rawdata[2*i+1];
    }
    return result;
}

void neutronxsec::readData( )
{
    std::vector< std::vector<double> > tot(my_wo.size());
    std::vector< std::vector<double> > esc(my_wo.size());
    std::vector< std::vector<double> > cap(my_wo.size());
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        tot[i] = readtxtbadformat(
            my_datapath+str(my_Z[i])+"_"+str(my_A[i])+"_1.dat" );
        esc[i] = readtxtbadformat(
            my_datapath+str(my_Z[i])+"_"+str(my_A[i])+"_2.dat" );
        cap[i] = readtxtbadformat(
            my_datapath+str(my_Z[i])+"_"+str(my_A[i])+"_102.dat" );
    }

    my_tot_erg.resize(my_wo.size());
    my_tot_xsec.resize(my_wo.size());
    my_esc_erg.resize(my_wo.size());
    my_esc_xsec.resize(my_wo.size());
    my_cap_erg.resize(my_wo.size());
    my_cap_xsec.resize(my_wo.size());
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        my_tot_erg[i] = extractEnergy( tot[i] );
        my_tot_xsec[i] = extractXSec( tot[i] );
        my_esc_erg[i] = extractEnergy( esc[i] );
        my_esc_xsec[i] = extractXSec( esc[i] );
        my_cap_erg[i] = extractEnergy( cap[i] );
        my_cap_xsec[i] = extractXSec( cap[i] );
    }
}

double neutronxsec::getXSec( double E,
    const std::vector<double>& rx,
    const std::vector<double>& erg )
{
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        if ( E >= erg[e] && E <= erg[e+1] )
        {
            const double sigma = logInterpolate(
                erg[e],rx[e],erg[e+1],rx[e+1],E );
            return sigma * 1e-24 * phys::avo / my_rho;
        }
    }
    return 0.0;
}

double neutronxsec::tot( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i]/my_A[i] * getXSec(
            E,my_tot_xsec[i],my_tot_erg[i] );
    }
    return result;
}

double neutronxsec::esc( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i]/my_A[i] * getXSec(
            E,my_esc_xsec[i],my_esc_erg[i] );
    }
    return result;
}
```

```
double neutronxsec::cap( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i]/my_A[i] * getXSec(
            E,my_cap_xsec[i],my_cap_erg[i] );
    }
    return result;
}




photonxsec::photonxsec( int Z,double rho )
{
    initialize(Z,rho);
}

photonxsec::photonxsec( const std::vector<int>& Z,
    const std::vector<double>& wo,double rho )
{
    initialize(Z,wo,rho);
}

void photonxsec::initialize( int Z,double rho )
{
    my_Z.resize(1);
    my_wo.resize(1);
    my_Z[0] = Z;
    my_wo[0] = 1.0;
    my_rho = rho;
    my_datapath = "data/xsec/photon/";
    readData( );
}

void photonxsec::initialize( const std::vector<int>& Z,
    const std::vector<double>& wo,double rho )
{
    my_Z = Z;
    my_wo = wo;
    my_rho = rho;
    my_datapath = "data/xsec/photon/";
    readData( );
}

void photonxsec::readData( )
{
    my_erg.resize(my_wo.size());
    my_tot.resize(my_wo.size());
    my_inc.resize(my_wo.size());
    my_coh.resize(my_wo.size());
    my_abs.resize(my_wo.size());
    my_ppe.resize(my_wo.size());
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        my_erg[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_erg.dat" );
        my_tot[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_501.dat" );
        my_inc[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_504.dat" );
        my_coh[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_502.dat" );
        my_abs[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_522.dat" );
        my_ppe[i] = readtxt<double>(
            my_datapath+str(my_Z[i])+"_516.dat" );
    }
}

double photonxsec::getXSec(
    double E,const std::vector<double>& rx,
    const std::vector<double>& erg )
```

```
{
    for ( unsigned int e=0;e<erg.size()-1;++e )
    {
        if ( E >= erg[e] && E <= erg[e+1] )
        {
            return my_rho * logInterpolate(
                erg[e],rx[e],erg[e+1],rx[e+1],E );
        }
    }
    return 0.0;
}

double photonxsec::tot( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i] * getXSec( E,my_tot[i],my_erg[i] );
    }
    return result;
}

double photonxsec::inc( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i] * getXSec( E,my_inc[i],my_erg[i] );
    }
    return result;
}

double photonxsec::coh( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i] * getXSec( E,my_coh[i],my_erg[i] );
    }
    return result;
}

double photonxsec::abs( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i] * getXSec( E,my_abs[i],my_erg[i] );
    }
    return result;
}

double photonxsec::ppe( double E )
{
    double result = 0.0;
    for ( unsigned int i=0;i<my_wo.size();++i )
    {
        result += my_wo[i] * getXSec( E,my_ppe[i],my_erg[i] );
    }
    return result;
}
```

Listing B.63: errh.cpp

```
#include "errh.hpp"

#include <sstream>

error::error( const std::string& a )
{
    my_error = a;
}

std::string error::getError( )
{
    return my_error;
}
```

Listing B.64: errh.hpp

```
#ifndef _errorhandling_hpp_included_
#define _errorhandling_hpp_included_

#include <exception>
#include <string>
#include <vector>
#include <iostream>
#include <sstream>

class error {
    public:
    //! pure virtual function to print error
    virtual void PrintError() = 0;
    virtual std::string getError();

    error( const std::string& );
    virtual ~error() {};

    protected:
    std::string my_error; //!< error string
};

class fatal_error : public error {
public:
    //! print the fatal error
    void PrintError( )
    {
        std::cout << "fatal_error: " << my_error << std::endl;
    }
    //! construct the fatal error with a message
    fatal_error( const std::string& a ) : error(a) { };

protected:

};

class warning : public error {
public:
    //! print the warning
    virtual void PrintError()
    {
        std::cout << "warning: " << my_error << std::endl;
    }
    //! construct the warning with a message
    warning( const std::string& a ) : error(a) { };
    void addWarn ( std::string a ) { warnMsgs.push_back(a); return; };

protected:
    std::vector<std::string> warnMsgs;

};


/*
void HandleException()
{
    try {
        throw;
    }
    catch ( fatal_error& e ) {
        e.PrintError();
        throw;
    }
    catch ( warning& e ) {
        e.PrintError();
    }
}
*/

#endif
```

Listing B.65: fileio.cpp

```
// ugly hack to get it to work with windows...
#define ISWIN false
#if defined(MSDOS) || defined(OS2) || defined(WIN32) || defined(__CYGWIN__)
  #define ISWIN true
#endif


#include "fileio.hpp"
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <fstream>
//#include <stdlib.h>
#include <sys/stat.h>


#include <boost/iostreams/filtering_streambuf.hpp>
#include <boost/iostreams/filtering_stream.hpp>
#include <boost/iostreams/copy.hpp>
#include <boost/iostreams/filter/zlib.hpp>

using namespace std;

// return system-dependent folder separator
string sep()
{
    if ( ISWIN )
    {
        return "\\";
    }
    else
    {
        return "/";
    }
}

void deldir( const string& dir )
{
    if ( ISWIN )
    {
        if ( system( ("rmdir "+dir).c_str() ) != 0 )
        {
            std::cerr << "could not delete directory " << dir << std::endl;
        }
    }
    else
    {
        if ( system( ("rm -r "+dir+" 2> /dev/null").c_str() ) != 0 )
        {
            std::cerr << "could not delete directory " << dir << std::endl;
        }
    }
}

void delfile( const string& file )
{
    if ( ISWIN )
    {
        if ( system( ("del "+file).c_str() ) != 0 )
        {
            std::cerr << "could not delete file " << file << std::endl;
        }
    }
    else
    {
        if ( system( ("rm "+file).c_str()) != 0 )
        {
            std::cerr << "could not delete file " << file << std::endl;
        }
    }
}

void mkdir( const string& dir )
{
```

```
    if ( ISWIN )
    {
        if ( system( ("mkdir "+dir).c_str() ) != 0 )
        {
            //std::cerr << "could not make directory " << dir << std::endl;
        }
    }
    else
    {
        if ( system( ("mkdir -p "+dir+" 2> /dev/null").c_str() ) != 0 )
        {
            //std::cerr << "could not make directory " << dir << std::endl;
        }
    }
}

void decompress( const string& filename,const string& dir )
{
    if ( ISWIN )
    {
        throw fileiowarn("decompression not defined for windows yet");
    }

    if ( system( ("tar --directory="+dir+" -xzf "+filename).c_str() ) != 0 )
    {
        throw fileiowarn("could not decompress file " + filename);
    }
    // boost way
    //const char* outfile = filename.c_str();
    //const char* infile = (filename+".z").c_str();
    //using namespace std;
    //using namespace boost::iostreams; // Added this line.
    //ifstream in(infile, ios_base::in | ios_base::binary);
    //filtering_streambuf<input> filter;
    //filter.push(zlib_decompressor());
    //filter.push(in);
    //ofstream out(outfile);
    //boost::iostreams::copy(filter,out);
    //in.close( );
    //out.close( );
}

void compress( const string& filename )
{
    const char* infile = filename.c_str();
    const char* outfile = (filename+".z").c_str();
    using namespace std;
    using namespace boost::iostreams;
    ifstream in(infile, ios_base::in | ios_base::binary);
    ofstream out(outfile, ios_base::out | ios_base::binary);
    filtering_streambuf<output> filter;

    filter.push(zlib_compressor());
    filter.push(out);
    boost::iostreams::copy(in,filter);
}



void writebin( const vector<double>& v,const string& filename )
{
    // STL's copy algorithm is quite slow
    //
    // typedef ostream_iterator<double> oi_t;
    // output << setw ( 15 )
    // << setiosflags ( ios::scientific ) << setprecision ( 10 );
    // copy (v.begin(), v.end(), oi_t(output, " "));

    // writing binary files takes about
    // 1/2 as long, with 1/2 the file size
    //
    ofstream output (filename.c_str(), ios::binary);
    for (size_t i = 0; i < v.size(); ++i)
    {
        output.write( (char*)&(v[i]),sizeof(double) );
    }
```

```
    output.close( );
}

void writebincomp( const vector<double>& v,const string& filename )
{
    // open filename + .z extension for output
    ofstream output( (filename+".z").c_str(),ios::binary );
    //std::cout << "writing file " << filename+".z" << std::endl;
    using namespace std;
    using namespace boost::iostreams;
    // initialize boost compressing stream as a filter
    boost::iostreams::filtering_ostream filter;
    filter.push(zlib_compressor());
    // link the filter to stream compressed data to the output file
    filter.push(output);
    // write data to the filter (which will go to the file)
    for (size_t i = 0; i < v.size(); ++i)
    {
        filter.write( (char*)&(v[i]),sizeof(double) );
    }
}

vector<double> readbin( const string& filename )
{
    //std::cout << "reading file " << filename << std::endl;
    // check if the file name exists
    if ( ! exists( filename ) )
    {
        //std::cout << "file doesn't exist,
        // reading compressed version" << std::endl;
        // if it doesn't exist, see if it's compressed version exists
        return readbincomp( filename );
    }
    // open file name for reading
    ifstream input( filename.c_str(),ios::binary );
    vector<double> v;
    // while we haven't reached the end of file, read in values
    // one-by-one into vector
    while ( input.peek() != EOF )
    {
        double readval;
        input.read((char*)&readval, sizeof(double));
        v.push_back(readval);
    }
    input.close();
    //std::cout << "done reading file" << std::endl;
    return v;
}

vector<double> readbincomp( const string& filename )
{
    //std::cout << "reading compressed file " << filename+".z" << std::endl;
    // check if the file name exists
    if ( ! exists( filename+".z" ) )
    {
        throw fileiowarn("file " + filename + " or " + filename + ".z does not exist");
    }
    // open compressed file name for reading
    ifstream input( (filename+".z").c_str(),ios::binary );
    using namespace std;
    using namespace boost::iostreams;
    // initialize boost filter
    boost::iostreams::filtering_istream filter;
    filter.push(zlib_decompressor());
    // connect boost filter to the input file
    filter.push(input);
    // read values from filter to the vector
    vector<double> v;
    while ( filter.peek() != EOF )
    {
        double readval;
        filter.read((char*)&readval, sizeof(double));
        v.push_back(readval);
    }
    //std::cout << "done reading compressed file" << std::endl;
    return v;
```

```
}

template < class T >
void writetxt( const vector<T>& v,const string& filename )
{
    // open file
    std::ofstream output( filename.c_str() );
    // set formatting
    output << std::setiosflags ( std::ios::scientific ) << std::setprecision ( 10 );
    // write values in text format one by one
    for ( size_t i=0;i<v.size();++i )
    {
        output << v[i] << "\n";
    }
    output.close();
}

template void writetxt<double>( const vector<double>& v,const string& filename );
template void writetxt<int>( const vector<int>& v,const string& filename );
template void writetxt<string>( const vector<string>& v,const string& filename );

double readfloat( ifstream& inp )
{
    string mystring;
    inp >> mystring;
    if ( mystring.find("E") == string::npos
        && mystring.find("e") == string::npos )
    {
        //std::cout << mystring << std::endl;
        size_t pos = mystring.find("-");
        if ( pos == string::npos )
        {
            pos = mystring.find("+");
        }
        if ( pos != string::npos )
        {
            mystring.insert(pos,"E");
        }
        return strtod(mystring.c_str(),NULL);
    }
    else
    {
        return strtod(mystring.c_str(),NULL);
    }
}

vector<double> readtxtbadformat( const string& filename )
{
    // open file
    ifstream input (filename.c_str());
    // read values one by one
    vector<double> v;
    while ( true )
    {
        if ( input.peek() != EOF )
        {
            v.push_back(readfloat(input));
        }
        else
        {
            break;
        }
    }
    input.close();
    //std::cout << "done reading" << std::endl;
    return v;
}

template< class T >
vector<T> readtxt( const string& filename )
{
    // open file
    ifstream input (filename.c_str());
```

```
    if ( ! input.good() )
    {
        throw fileiowarn("filename "+filename+" does not exist");
    }
    // read values one by one
    T dummy;
    vector<T> v;
    while ( true )
    {
        input >> dummy;
        if ( ! input.eof() )
        {
            v.push_back(dummy);
        }
        else
        {
            break;
        }
    }
    input.close();
    return v;
}

template vector<double> readtxt( const string& );
template vector<int> readtxt( const string& );
template vector<string> readtxt( const string& );

template < class T >
void writeVal( T val,const std::string& filename )
{
    ofstream out(filename.c_str());
    out << val;
    out.close();
}

template < class T >
T readVal( const std::string& filename )
{
    ifstream in(filename.c_str());
    T val;
    in >> val;
    in.close();
    return val;
}


template void writeVal( double val,const std::string& );
template double readVal( const std::string& );

template void writeVal( int val,const std::string& );
template int readVal( const std::string& );

// check if file exists, returns true if file does exist
bool exists( const std::string& filename )
{
    //ifstream input (filename.c_str());
    //bool isOkay = input.good();
    //input.close();
    ////isOkay = input.eof() && isOkay;
    //return isOkay;

    struct stat st;
    return (!stat(filename.c_str(),&st));
}
```

Listing B.66: fileio.hpp

```
#ifndef _fileio_hpp_
#define _fileio_hpp_

#include <cstdlib>
#include <vector>
#include <string>
#include <sstream>
#include <iomanip>
#include <fstream>

#include "errh.hpp"


class fileiowarn : public warning
{
    public:

    virtual void PrintError()
    {
        std::cout << "file io error: " << my_error << std::endl;
    }

    fileiowarn( const std::string& a ) : warning(a) { };

    private:

};

// stringify anything that knows what the << operator does
template< class T >
std::string str( const T& a )
{
    std::stringstream ss;
    ss << a;
    return ss.str();
}

// convert anything into a number, or really anything for that matter
template< class T >
T fromstr( const std::string& a )
{
    T result;
    std::istringstream(a) >> result;
    return result;
}

std::string sep();
void deldir( const std::string& );
void delfile( const std::string& );
void mkdir( const std::string& );
void decompress( const std::string&,const std::string& );
void compress( const std::string& );
void writebin( const std::vector<double>&,const std::string& );
void writebincomp( const std::vector<double>&,const std::string& );
std::vector<double> readbin( const std::string& );
std::vector<double> readbincomp( const std::string& );

template < class T >
void writetxt( const std::vector<T>&,const std::string& );

template < class T >
std::vector<T> readtxt( const std::string& );

std::vector<double> readtxtbadformat( const std::string& );

template < class T >
void writeVal( T val,const std::string& );

template < class T >
T readVal( const std::string& );

bool exists( const std::string& );
```

```
template <class T>
bool Find ( T& str ,std::string keyword,bool rewind )
{
    int initialPos = str.tellg();
    if ( rewind )
    {
        str.clear();
        str.seekg ( 0,std::ios::beg );
    }
    bool found;
    char dummy;
    char begincomment = '#';
    const int size = keyword.size();
    do
    {
        found = false;
        dummy = str.get();
        if ( dummy == begincomment )
        {
            do
            {
                dummy = str.get();
                if ( dummy == '\n' )
                {
                    break;
                }
            } while ( ! str.eof() );
            continue;
        }

        if ( dummy == keyword[0] )
        {
            found = true;
            for ( int i=1; i<size; ++i )
            {
                dummy = str.get();
                if ( dummy != keyword[i] )
                {
                    found = false;
                    str.seekg ( -1,std::ios::cur );
                    break;
                }
            }
        }
    } while ( ! ( str.eof() ) && ! found );
    if ( str.eof() )
    {
        str.clear ( );
    }
    if ( ! found )
    {
        str.seekg ( initialPos,std::ios_base::beg );
    }
    return found;
}

#endif
```

Listing B.67: matrix.cpp

```
#include "matrix.hpp"
#include <cmath>

#define pi 3.14159265358979323846

//! @brief generates a transformation matrix
//!
//! given two vectors of arbitrary length and values
//! this function generates a matrix that maps the values
//! to one another when applied to a vector
//! @param a original (old) vector of floats
//! @param b new vector of floats
//! @return transformation matrix
matrix < double > genTransform ( const std::vector < double >& a,
    const std::vector < double >& b )
{
    const int numOldErg = static_cast<int> ( a.size() );
    const int numNewErg = static_cast<int> ( b.size() );
    if ( ! numOldErg > 0 || ! numNewErg > 0 )
    {
        return matrix<double>(0,0);
    }
    matrix < double > transMat ( numNewErg-1,numOldErg-1 );

    //std::cout << "old energy size = " << a.size() << std::endl;
    //std::cout << "new energy size = " << b.size() << std::endl;


    // First check to see if a == b, then just make normal identity
    //std::cout << "checking if we can use normal identity" << std::endl;
    if ( a.size() == b.size() )
    {
        const double eps = 1e-20;
        bool match = true;
        for ( unsigned int e=0;e<a.size();++e )
        {
            //std::cout << "a,b: " << a[e] << " " << b[e] << std::endl;
            if ( fabs(a[e]-b[e]) > eps )
            {
                //std::cout << "they don't match" << std::endl;
                match = false;
                break;
            }
        }
        if ( match )
        {
            //std::cout << "making standard identity matrix" << std::endl;
            for ( unsigned int e=0;e<a.size()-1;++e )
            {
                //std::cout << "row,col = " << e << std::endl;
                transMat(e,e) = 1.0;
            }
            //std::cout << "done, returning identity" << std::endl;
            return transMat;
        }
    }



    int oldIdx = 0;
    int newIdx = 0;
    do
    {
        // new upper energy bound
        const double newub = b[newIdx+1];
        // old upper energy bound
        const double oldub = a[oldIdx+1];
        // new lower energy bound
        const double newlb = b[newIdx];
        // old lower energy bound
        const double oldlb = a[oldIdx];

        // five cases are possible
        //
        // 1) lower and upper bound of old spectrum lies completely within
```

```
        //    the new one, so we can just add it to the new spectrum bin
        //
        double frac = 0.0;
        if ( oldlb > newlb && oldub <= newub )
        {
            frac = 1.0;
        }
        //
        // 2) the lower boundary of the old spectrum lies below but
        //    the upper boundary is within, reduce the fraction added
        //    by the energy distance between the old upper bound
        //    and new lower bound
        //
        else if ( oldlb <= newlb && oldub > newlb && oldub <= newub )
        {
            frac = ( oldub - newlb ) / ( oldub - oldlb );
        }
        //
        // 3) the upper boundary of the old spectrum lies above but
        //    the lower boundary is within, similar to case (2)
        //
        else if ( oldlb > newlb && oldlb <= newub && oldub > newub )
        {
            frac = ( newub - oldlb ) / ( oldub - oldlb );
        }
        //
        // 4) the upper boundary of the old spectrum lies above and
        //    the lower boundary of the old spectrum lies below, have to
        //    compute fraction that new bin occupies in old bin
        //
        else if ( oldlb <= newlb && oldub > newub )
        {
            frac = ( newub - newlb ) / ( oldub - oldlb );
        }
        // 4) the old bin lies completely below the new bin, the data is lost
        //
        else if ( oldlb < newlb && oldub < newlb )
        {
            frac = 0;
        }

        transMat ( newIdx,oldIdx ) += frac;

        if ( oldub > newub )
        {
            newIdx++;
        }
        else
        {
            oldIdx++;
        }

    }
    while ( oldIdx < (numOldErg-1) && newIdx < (numNewErg-1) );

    return transMat;
}
```

Listing B.68: matrix.hpp

```
#ifndef _matrix_hpp_included_
#define _matrix_hpp_included_

#include <vector>
#include <ostream>
#include <limits>
#include <utility>
#include "errh.hpp"

//! @class matrix
//! @brief templated class for two-dimensional matrices
template < class T >
class matrix
{
    public:

    int getM ( ) const;

    int getN ( ) const;

    void resize ( int,int );

    void identity ( );

    matrix<T> inverse( );

    matrix ( int,int );

    matrix ( ) { };

    T& operator () ( int,int );
    T operator () ( int,int ) const;

    void operator = ( const matrix<T>& a );
    void operator = ( const std::vector<T>& a );
    void operator *= ( double );

    void setDiagonal( T );
    void setDiagonal( const std::vector<T>& );


    void insertColumn( int rowIdx,const std::vector<T>& newcol );

    int getColMaxIdx( int );

    matrix<T> transpose( );

    void optimize( );
    bool isOptimized() const { return ( my_Midx.size() >0 ); };
    int getMIdx(int idx) const { return my_Midx[idx]; };
    int getNIdx(int idx) const { return my_Nidx[idx]; };
    int numIdx() const { return static_cast<int>(my_Midx.size()); };

    std::vector< std::vector<T> > getData() const
    {
        return my_data;
    }

    void clear()
    {
        my_data.clear();
    }

    protected:

    T& val( int i,int j );
    T val( int i,int j ) const;
    void sizeme ( int,int );

    std::vector< std::vector<T> > my_data;

    std::vector< int > my_Midx;
    std::vector< int > my_Nidx;

};
```

```
template<class T>
void matrix<T>::optimize( )
{
    const int M = getM();
    const int N = getN();
    for ( int i=0;i<M;++i )
    {
        for ( int j=0;j<N;++j )
        {
            if ( val(i,j) > 1e-50 )
            {
                my_Midx.push_back(i);
                my_Nidx.push_back(j);
            }
        }
    }
}

template < class T >
void matrix<T>::insertColumn( int rowIdx,const std::vector<T>& newcol )
{
    const int M = getM();
    const int N = getN();
    if ( static_cast<int>(newcol.size()) != M )
    {
        throw fatal_error("cannot insert column of different size");
    }
    matrix<T> temp(M,N+1);
    for ( int m=0;m<M;++m )
    {
        for ( int n=0;n<rowIdx;++n )
        {
            temp(m,n) = val(m,n);
        }

        temp(m,rowIdx) = newcol[m];

        for ( int n=rowIdx;n<N;++n )
        {
            temp(m,n+1) = val(m,n);
        }
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
}

template < class T >
int matrix<T>::getColMaxIdx( int row )
{
    T max = val(row,0);
    int maxIdx;
    const int N = getN();
    for ( int n=0;n<N;++n )
    {
        if ( val(row,n) > max )
        {
            max = val(row,n);
            maxIdx = n;
        }
    }
    return maxIdx;
}

template < class T >
void matrix<T>::setDiagonal( T a )
{
    if ( getM() != getN() )
    {
        throw fatal_error("matrix not suitable for \
setting diagonal, not square");
    }
    const int M = getM();
    for ( int i=0;i<M;++i )
    {
        val(i,i) = a;
```

```
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
}

template < class T >
void matrix<T>::setDiagonal( const std::vector<T>& a )
{
    if ( getM() != getN() )
    {
        throw fatal_error("matrix not suitable for \
setting diagonal, not square");
    }
    if ( static_cast<int>(a.size()) != getN() )
    {
        throw fatal_error("cannot assign vector to \
diagonal, not same size");
    }
    const int M = getM();
    for ( int i=0;i<M;++i )
    {
        val(i,i) = a[i];
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
}

template < class T >
matrix<T> matrix<T>::transpose( )
{
    matrix<T> temp(getN(),getM());
    const int M = getM();
    const int N = getN();
    for ( int i=0;i<M;++i )
    {
        for ( int j=0;j<N;++j )
        {
            temp(j,i) = val(i,j);
        }
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
    return temp;
}

//! @brief returns the number of rows in the matrix
//! @return number of rows M
template < class T >
int matrix<T>::getM ( ) const
{
    return static_cast<int>(my_data.size());
}

//! @brief returns the number of columns in the matrix
//! @return number of columns N
template < class T >
int matrix<T>::getN ( ) const
{
    if ( getM() > 0 )
        return static_cast<int>(my_data[0].size());
    else
        return 0;
}

//! @brief resize the matrix
//! @param m rows
//! @param n columns
template < class T >
void matrix<T>::resize ( int m,int n )
{
    sizeme ( m,n );
    return;
}

//! @brief turn matrix into identity
//! zeros everywhere except for ones along the diagonal
```

```
template < class T >
void matrix<T>::identity ( )
{
    if ( getM() != getN() )
    {
        throw fatal_error("matrix not suitable for \
identity, not square");
    }
    const int M = getM();
    const int N = getN();
    for ( int i=0;i<M;++i )
    {
        for ( int j=0;j<i;++j )
        {
            val(i,j) = 0.0;
        }

        val(i,i) = 1.0;

        for ( int j=i+1;j<N;++j )
        {
            val(i,j) = 0.0;
        }
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
    return;
}

//! @brief private function that performs the resizing
//! @param m rows
//! @param n columns
template < class T >
void matrix<T>::sizeme ( int M,int N )
{
    my_data.resize(M);
    for ( int m=0;m<M;++m )
    {
        my_data[m].resize(N);
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
    return;
}

//! @brief constructor that sizes the matrix
//! @param m rows
//! @param n columns
template < class T >
matrix<T>::matrix ( int m,int n )
{
    sizeme ( m,n );
}

template < class T >
T& matrix<T>::val( int i,int j )
{
    return my_data [i][j];
}

template < class T >
T matrix<T>::val( int i,int j ) const
{
    return my_data [i][j];
}

//! @brief retrieves the reference to a single element in the matrix
//! @param i the ith row element
//! @param j the jth column element
//! @return element i,j in the matrix
template < class T >
T& matrix<T>::operator () ( int i,int j )
{
    return my_data [i][j];
//    return data.at(i+j*getM());
}
```

```
//! @brief retrieves a single element in the matrix
//! @param i the ith row element
//! @param j the jth column element
//! @return element i,j in the matrix
template < class T >
T matrix<T>::operator () ( int i,int j ) const
{
    return my_data [i][j];
//    return data.at(i+j*getM());
}


//! @brief assign matrix to another
//! @param a RHS matrix
template < class T >
void matrix<T>::operator = ( const matrix<T>& a )
{
    my_data = a.getData();
    my_Midx.resize(0);
    my_Nidx.resize(0);
    return;
}

template < class T >
void matrix<T>::operator = ( const std::vector<T>& a )
{
    resize ( a.size(),1 );
    const int M = getM();
    for ( int i=0;i<M;++i )
    {
        val(i,0) = a[i];
    }
    my_Midx.resize(0);
    my_Nidx.resize(0);
    return;
}

template < class T >
void matrix<T>::operator *= ( double a )
{
    const int M = getM();
    const int N = getN();
    for ( int i=0;i<M;++i )
        for ( int j=0;j<N;++j )
            my_data[i][j] *= a;
    return;
}

//! @brief matrix-vector multiplication
//! @param mat matrix
//! @param vec vector
//! @return resultant vector from multiplication
template < class T >
std::vector<T> operator * ( const matrix<T>& mat,
    const std::vector<T>& vec )
{
    if ( mat.getN() != static_cast<int>(vec.size()) )
    {
        throw fatal_error ( "matrix-vector multiplication \
dimension mismatch" );
    }

    std::vector < T > result ( mat.getM() );

    if ( mat.isOptimized() )
    {
        const int nIdx = mat.numIdx();
        for ( int i=0;i<nIdx;++i )
        {
            const int I = mat.getMIdx(i);
            const int J = mat.getNIdx(i);
            result[I] += mat(I,J)*vec[J];
        }
    }
    else
```

```
    {
        const int M = mat.getM();
        const int N = mat.getN();
        for ( int i=0;i<M;++i )
        {
            for ( int j=0;j<N;++j )
            {
                result[i] = result[i] + mat(i,j)* vec[j];
            }
        }
    }

    return result;
}

//! @brief matrix-matrix multiplication
//! @param left left matrix
//! @param right right matrix
//! @return resultant matrix from multiplication
template < class T >
matrix<T> operator * ( const matrix<T>& left,const matrix<T>& right )
{
    if ( left.getN() != right.getM() )
    {
        throw fatal_error ( "matrices not suitable for \
multiplication, n1 neq m2" );
    }

    matrix < T > result ( left.getM(),right.getN() );

    const int lM = left.getM();
    const int rN = right.getN();
    for ( int i=0;i<lM;++i )
    {
        for ( int j=0;j<rN;++j )
        {
            for ( int k=0;k<lM;++k )
            {
                result(i,j) = result(i,j) + left(i,k)* right(k,j);
            }
        }
    }

    return result;
}

//! @brief matrix-matrix addition element-by-element
//! @param left left matrix
//! @param right right matrix
//! @return resultant matrix
template < class T >
matrix<T> operator + ( const matrix<T>& left,const matrix<T>& right )
{
    if ( left.getN() != right.getN() || left.getM() != right.getM() )
    {
        std::cout << "left.getN() = " << left.getN() << std::endl;
        std::cout << "right.getN() = " << right.getN() << std::endl;
        std::cout << "left.getM() = " << left.getM() << std::endl;
        std::cout << "right.getM() = " << right.getM() << std::endl;
        throw fatal_error ( "matrices not suitable for addition" );
    }

    matrix < T > result ( left.getM(),right.getN() );

    const int lM = left.getM();
    const int rN = right.getN();
    for ( int i=0;i<lM;++i )
    {
        for ( int j=0;j<rN;++j )
        {
            result(i,j) = left(i,j) + right(i,j);
        }
    }

    return result;
}
```

```
//! @brief matrix-matrix subtraction element-by-element
//! @param left left matrix
//! @param right right matrix
//! @return resultant matrix
template < class T >
matrix<T> operator - ( const matrix<T>& left ,const matrix<T>& right )
{
    if ( left.getN() != right.getN() || left.getM() != right.getM() )
    {
        std::cout << "left.getN() = " << left.getN() << std::endl;
        std::cout << "right.getN() = " << right.getN() << std::endl;
        std::cout << "left.getM() = " << left.getM() << std::endl;
        std::cout << "right.getM() = " << right.getM() << std::endl;
        throw fatal_error ( "matrices not suitable for subtraction" );
    }

    matrix < T > result ( left.getM(),right.getN() );

    const int lM = left.getM();
    const int rN = right.getN();
    for ( int i=0;i<lM;++i )
    {
        for ( int j=0;j<rN;++j )
        {
            result(i,j) = left(i,j) - right(i,j);
        }
    }

    return result;
}

//! @brief scales a matrix by a constant
//! @param mat matrix
//! @param a constant
//! @return resultant scaled matrix
template < class T >
matrix<T> operator * ( const matrix<T>& mat,double a )
{
    matrix < T > result ( mat.getM(),mat.getN() );

    const int M = mat.getM();
    const int N = mat.getN();
    for ( int i=0;i<M;++i )
    {
        for ( int j=0;j<N;++j )
        {
            result(i,j) = mat(i,j) * a;
        }
    }

    return result;
}


//! @brief scales a matrix by a constant
//! @param mat matrix
//! @param a constant
//! @return resultant scaled matrix
template < class T >
matrix<T> operator * ( double a,const matrix<T>& mat )
{
    return mat*a;
}

//! @brief scales a matrix by dividing by a constant
//! @param mat matrix
//! @param a constant
//! @return resultant scaled matrix
template < class T >
matrix<T> operator / ( const matrix<T>& mat,double a )
{
    return mat*(1.0/a);
}
```

```
//! @brief adds two vectors together element-by-element
//! @param a left vector
//! @param b right vector
//! @return resulting vector
template < class T >
std::vector<T> operator + ( const std::vector<T>& a,const std::vector<T>& b )
{
    if ( static_cast<int>(a.size()) != static_cast<int>(b.size()) )
    {
        throw fatal_error ( "mismatch vector sizes in addition" );
    }

    const int N = static_cast<int>(a.size());
    std::vector<T> result(N);
    for ( int i=0;i<N;++i )
    {
        result[i] = a[i] + b[i];
    }
    return result;
}

template < class T >
std::vector<T> operator - ( const std::vector<T>& a,const std::vector<T>& b )
{
    if ( static_cast<int>(a.size()) != static_cast<int>(b.size()) )
    {
        throw fatal_error ( "mismatch vector sizes in addition" );
    }

    const int N = static_cast<int>(a.size());
    std::vector<T> result(N);
    for ( int i=0;i<N;++i )
    {
        result[i] = a[i] - b[i];
    }
    return result;
}

//! @brief scale a vector by a constant element-by-element
//! @param a vector
//! @param b constant
//! @return resultant scaled vector
template < class T >
std::vector<T> operator * ( const std::vector<T>& a,double b )
{
    const int N = static_cast<int>(a.size());
    std::vector<T> result(N);
    for ( int i=0;i<N;++i )
    {
        result[i] = a[i]*b;
    }
    return result;
}

//! @brief scale a vector by a constant element-by-element
//! @param a vector
//! @param b constant
//! @return resultant scaled vector
template < class T >
std::vector<T> operator * ( double b,const std::vector<T>& a )
{
    return a*b;
}

//! @brief scale a vector by division by a constant element-by-element
//! @param a vector
//! @param b constant
//! @return resultant scaled vector
template < class T >
std::vector<T> operator / ( const std::vector<T>& a,double b )
{
    return a*(1.0/b);
}

template < class T >
std::vector<T> operator + ( const std::vector<T>& a,double b )
```

```
{
    const int N = static_cast<int>(a.size());
    std::vector<T> result(N);
    for ( int i=0;i<N;++i )
    {
        result[i] = a[i] + b;
    }
    return result;
}

template < class T >
std::vector<T> operator + ( double b,const std::vector<T>& a )
{
    return a+b;
}

template < class T >
std::vector<T> operator - ( const std::vector<T>& a,double b )
{
    return a+(-1.0*b);
}

template < class T >
std::vector<T> operator - ( double b,const std::vector<T>& a )
{
    return b+(-1.0*a);
}

//! @brief normalize a vector
//! @param a vector to normalize
//! @return normalized vector
template < class T >
std::vector<T> normalize ( const std::vector<T>& a )
{
    const int I = static_cast<int>( a.size() );
    T sum = 0;
    std::vector<T> newVec = a;
    for ( int i=0;i<I;++i )
    {
        sum += a[i];
    }
    if ( sum > std::numeric_limits<T>::min() )
    {
        for ( int i=0;i<I;++i )
        {
            newVec[i] = a[i]/sum;
        }
    }
    return newVec;
}


//! @brief find dot product of two vectors
//! @param a vector a
//! @param b vector b
//! @return dot product
template < class T >
T dotProduct( const std::vector<T>& a,const std::vector<T>& b )
{
    if ( a.size() != b.size() )
    {
        throw fatal_error("cannot dot product non matching vectors");
    }
    T result = 0;
    for ( unsigned int i=0;i<a.size();++i )
    {
        result = result + a[i]*b[i];
    }
    return result;
}

template < class T >
std::vector<T> projection( const std::vector<T>& a,
    const std::vector<T>& b )
{
    if ( a.size() != b.size() )
```

```
    {
        throw fatal_error("cannot dot product non matching vectors");
    }
    std::vector<T> result(a.size());
    result = b*( dotProduct(a,b)/pow(normalize(b),2.0) );
    return result;
}

template < class T >
std::vector<T> crossProduct( const std::vector<T>& a,
    const std::vector<T>& b )
{
    if ( a.size() != 3 ||  b.size() != 3 )
    {
        throw fatal_error("cannot cross product \
non-three-dimensional vectors");
    }
    std::vector<T> result(3);
    result[0] = a[1]*b[2]-a[2]*b[1];
    result[1] = a[2]*b[0]-a[0]*b[2];
    result[2] = a[0]*b[1]-a[1]*b[0];
    return result;
}

template < class T >
std::vector<T> projVecOnPlane( const std::vector<T>& a,
    const std::vector<T>& n )
{
    return crossProduct(n,crossProduct(a,n)/mag(n))/mag(n);
}

template < class T >
double angle( const std::vector<T>& a,const std::vector<T>& b )
{
    return acos( dotProduct(a,b)/(mag(a)*mag(b)) );
}

template <class T>
T mag( const std::vector<T>& a )
{
    return sqrt(dotProduct(a,a));
}

template <class T>
T sum( const std::vector<T>& a )
{
    T result = 0.0;
    for ( unsigned int i=0;i<a.size();++i )
    {
        result = result + a[i];
    }
    return result;
}

template <class T>
std::ostream& operator << ( std::ostream& output,
    const std::vector<T>& a )
{
    output << "(";
    for ( unsigned int i=0;i<a.size()-1;++i )
    {
        output << a[i] << " ";
    }
    output << a.back() << ")";
    return output;
}

//! build transformation matrix
matrix < double > genTransform ( const std::vector < double >&,
    const std::vector < double >& );

template < class T >
matrix<T> matrix<T>::inverse( )
{
    const int N = getN();
    const int M = getM();
```

```
    if ( M != N )
    {
        throw fatal_error("non-square matrix can't be inverted");
    }
    const int N2 = N+N;
    matrix aug(M,N2);

    for ( int i=0;i<M;++i )
    {
        for ( int j=N;j<N2;++j)
        {
            aug(i,j) = 0.0;
            aug(i,N+i) = 1.0;
        }
    }

    for (int i=0; i<M; ++i)
    {
        for (int j=0; j<N; ++j)
        {
            aug(i,j) = val(i,j);
        }
    }

    for (int i=0; i<M; ++i)
    {
        for (int j=0; j<i; ++j)
        {
            T factor = aug(j,i)/aug(i,i);
            for(int k=0; k<N2; ++k)
                aug(j,k) = aug(j,k) - aug(i,k)*factor;
        }
        for (int j=(N-1); j>i; --j)
        {
            T factor = aug(j,i)/aug(i,i);
            for (int k=0; k<N2; ++k)
                aug(j,k) = aug(j,k) - aug(i,k)*factor;
        }
    }

    for (int i=0; i<M; ++i)
    {
        T factor = aug(i,i);
        for (int j=0; j<N2; ++j)
        {
            aug(i,j) = aug(i,j)/factor;
        }
    }

    matrix temp(M,N);
    for (int i=0; i<M; ++i)
    {
        for (int j=0; j<N; ++j)
        {
            temp(i,j) = aug(i,N+j);
        }
    }
    return temp;
}

#endif
```

Listing B.69: mtally.cpp

```
#include "mtally.hpp"
#include <cmath>

using namespace std;

double& mtally::erg( int e )
{
    return my_erg[e];
}

vector<double>& mtally::erg( )
{
    return my_erg;
}

double& mtally::cos( int e )
{
    return my_cos[e];
}

vector<double>& mtally::cos( )
{
    return my_cos;
}

double& mtally::tim( int e )
{
    return my_tim[e];
}

vector<double>& mtally::tim( )
{
    return my_tim;
}

double& mtally::val( int cs,int f,int u,int s,int m,int c,int e,int t )
{
    return my_val[idx(cs,f,u,s,m,c,e,t)];
}

vector<double>& mtally::val( )
{
    return my_val;
}

double& mtally::err( int cs,int f,int u,int s,int m,int c,int e,int t )
{
    return my_err[idx(cs,f,u,s,m,c,e,t)];
}

std::vector<double>& mtally::err( )
{
    return my_err;
}

double mtally::erg( int e ) const
{
    return my_erg[e];
}

vector<double> mtally::erg( ) const
{
    return my_erg;
}

double mtally::cos( int e ) const
{
    return my_cos[e];
}

vector<double> mtally::cos( ) const
{
    return my_cos;
}
```

```
double mtally::tim( int e ) const
{
    return my_tim[e];
}

vector<double> mtally::tim( ) const
{
    return my_tim;
}

double mtally::val( int cs,int f,int u,int s,int m,int c,int e,int t ) const
{
    return my_val[idx(cs,f,u,s,m,c,e,t)];
}

vector<double> mtally::val( ) const
{
    return my_val;
}

double mtally::err( int cs,int f,int u,int s,int m,int c,int e,int t ) const
{
    return my_err[idx(cs,f,u,s,m,c,e,t)];
}

vector<double> mtally::err( ) const
{
    return my_err;
}

int mtally::idx( int cs,int f,int u,int s,int m,int c,int e,int t )
{
    return t+ntim*(e+nerg*(c+ncos*(m+nmult
        *(s+nseg*(u+nuser*(f+nflag*cs))))));
}

int mtally::idx( int cs,int f,int u,int s,
    int m,int c,int e,int t ) const
{
    return t+ntim*(e+nerg*(c+ncos*(m+nmult
        *(s+nseg*(u+nuser*(f+nflag*cs))))));
}

void mtally::resize( int ncs_,int nflag_,int nuser_,int nseg_,
    int nmult_,int ncos_,int nerg_,int ntim_ )
{
    ncs = ncs_;
    nflag = nflag_;
    nuser = nuser_;
    nseg = nseg_;
    nmult = nmult_;
    ncos = ncos_;
    nerg = nerg_;
    ntim = ntim_;

    my_erg.resize( nerg );
    my_cos.resize( ncos );
    my_tim.resize( ntim );
    my_val.resize( ncs*nflag*nuser*nseg*nmult*ncos*nerg*ntim );
    my_err.resize( ncs*nflag*nuser*nseg*nmult*ncos*nerg*ntim );
}

// This only exists because MCNP will output some numbers in m files like:
//   1.2345-101
// forgetting the "E" and messing everything up
double mtally::readfloat( ifstream& inp )
{
    string mystring;
    inp >> mystring;
    if ( mystring.find("E") == string::npos
        && mystring.find("e") == string::npos )
    {
        size_t pos = mystring.find("-");
        if ( pos != string::npos )
        {
```

```
            mystring.insert(pos,"E");
        }
        else
        {
            pos = mystring.find("-");
            mystring.insert(pos,"E");
        }
        return strtod(mystring.c_str(),NULL);
    }
    else
    {
        return strtod(mystring.c_str(),NULL);
    }
}

// return the first character of a stl string
char fch( const std::string& a )
{
    return a.at(0);
}

// return the last character of a stl string
char lch( const std::string& a )
{
    return a.at(a.size()-1);
}

void mtally::parse( int tnum,ifstream& inp )
{
    string token;
    int tempint;
    // cout << "parsing tally " << tnum << endl;

    // read tally number to confirm
    inp >> token;
    inp >> my_tnum;
    // cout << "tally number: " << my_tnum << endl;
    if ( tnum != my_tnum )
    {
        throw fatal_error("somethings wrong -- tally \
number specified does not match file");
    }
    inp >> tempint;
    inp >> tempint;
    getline( inp,token );// get to next line

    // read line of 0's and 1's representing
    // which particles are used in the problem
    getline( inp,token );

    // read number of cells/surface bins
    inp >> token;
    if ( *(token.begin()) != 'f' )
    {
        throw fatal_error("error: expected token \"f\" \
instead found \""+token+"\"");
    }
    inp >> ncs;
    getline( inp,token );// get to next line
    // read cells/surfaces used in the problem
    for ( int i=0;i<ncs;++i )
    {
        inp >> token;
    }
    // cout << "\t" << "found " << ncs  << " cell/surface bins" << endl;
    getline( inp,token );// get to next line

    // read number of flagged bins
    inp >> token;
    if ( *(token.begin()) != 'd' )
    {
        throw fatal_error("error: expected token \"d\" \
instead found \""+token+"\"");
    }
    inp >> nflag;
    if ( nflag == 0 )
```

```
        nflag++;
    // cout << "\t" << "found " << nflag << " flagged bins" << endl;
    getline( inp,token );// get to next line

    // read number of user bins
    inp >> token;
    if ( *(token.begin()) != 'u' )
    {
        throw fatal_error("error: expected token \"u\" \
instead found \""+token+"\"");
    }
    inp >> nuser;
    if ( nuser == 0 ) // mcnp prints zero if there's only one bin...
        nuser++;
    // check for total/cumulative bins, which we don't want to read
    bool usertotal = *(token.rbegin()) == 't' && token.length()>1;
    if ( usertotal )
        nuser--;
    my_userCum = *(token.rbegin()) == 'c' && token.length()>1;
    //if ( usercum )
    //  nuser--;
    // cout << "\t" << "found " << nuser << " user bins" << endl;
    getline( inp,token );// get to next line
    // read list of user bins
    if ( nuser > 1 )
    {
        // mcnp won't always print the user bins,
        // see if next line has 's' token
        int pos = inp.tellg();
        inp >> token;
        inp.seekg(pos);
        if ( *(token.begin()) != 's' )
        {
            for ( int i=0;i<nuser;++i )
            {
                inp >> token;
            }
            getline( inp,token );
        }
    }

    // read segment bins
    inp >> token;
    if ( *(token.begin()) != 's' )
    {
        throw fatal_error("error: expected token \"s\" \
instead found \""+token+"\"");
    }
    inp >> nseg;
    if ( nseg == 0 ) // mcnp prints zero if there's only one bin...
        nseg++;
    // check for total/cumulative bins, which we don't want to read
    bool segtotal = *(token.rbegin()) == 't' && token.length()>1;
    if ( segtotal )
        nseg--;
    my_segCum = *(token.rbegin()) == 'c' && token.length()>1;
    //if ( segcum )
    //  nseg--;
    // cout << "\t" << "found " << nseg << " segment bins" << endl;
    getline( inp,token ); // get to next line

    // read multiplier bins
    inp >> token;
    if ( *(token.begin()) != 'm' )
    {
        throw fatal_error("error: expected token \"m\" \
instead found \""+token+"\"");
    }
    inp >> nmult;
    if ( nmult == 0 ) // mcnp prints zero if there's only one bin...
        nmult++;
    // check for total/cumulative bins, which we don't want to read
    bool multtotal = *(token.rbegin()) == 't' && token.length()>1;
    if ( multtotal )
        nmult--;
    my_multCum = *(token.rbegin()) == 'c' && token.length()>1;
```

```
    getline( inp,token );

    // read number of cosine bins
    inp >> token;
    if ( *(token.begin()) != 'c' )
    {
        throw fatal_error("error: expected token \"c\" \
instead found \""+token+"\"");
    }
    inp >> ncos;
    if ( ncos == 0 ) // mcnp prints zero if there's only one bin...
        ncos++;
    // check for total/cumulative bins, which we don't want to read
    bool costotal = *(token.rbegin()) == 't' && token.length()>1;
    if ( costotal )
        ncos--;
    my_cosCum = *(token.rbegin()) == 'c' && token.length()>1;
    //if ( coscum )
    //  ncos--;
    // cout << "\t" << "found " << ncos << " cosine bins" << endl;
    getline( inp,token ); // get to next line
    // read cosine bins (not implemented)
    if ( ncos > 1 )
    {
        my_cos.resize( ncos );
        for ( int i=0;i<ncos;++i )
        {
            inp >> my_cos[i];
        }
        getline( inp,token );// get to next line
    }


    // energy bins
    inp >> token;
    if ( fch(token) != 'e' )
    {
        throw fatal_error("error: expected token \"e\" \
instead found \""+token+"\"");
    }
    inp >> nerg;
    if ( nerg == 0 ) // mcnp prints zero if there's only one bin...
        nerg++;
    // check for total/cumulative bins, which we don't want to read
    bool ergtotal = *(token.rbegin()) == 't' && token.length()>1;
    if ( ergtotal )
        nerg--;
    my_ergCum = *(token.rbegin()) == 'c' && token.length()>1;
    //if ( ergcum )
    //  nerg--;
    // cout << "\t" << "found " << nerg << " energy bins" << endl;
    getline( inp,token );// get to next line
    // read in energy bins
    if ( nerg > 1 )
    {
        my_erg.resize( nerg );
        for ( int i=0;i<nerg;++i )
        {
            inp >> my_erg[i];
        }
        getline( inp,token );// get to next line
    }


    // time bins
    inp >> token;
    if ( *(token.begin()) != 't' )
    {
        throw fatal_error("error: expected token \"t\" \
instead found \""+token+"\"");
    }
    inp >> ntim;
    if ( ntim == 0 ) // mcnp prints zero if there's only one bin...
        ntim++;
    // check for total/cumulative bins, which we don't want to read
    bool timtotal = *(token.rbegin()) == 't' && token.length()>1;
```

```
    if ( timtotal )
        ntim--;
    my_timeCum = *(token.rbegin()) == 'c' && token.length()>1;
    //if ( timcum )
    //  ntim--;
    // cout << "\t" << "found " << ntim << " time bins" << endl;
    getline( inp,token );// get to next line
    // read in energy bins
    if ( ntim > 1 )
    {
        my_tim.resize( ntim );
        for ( int i=0;i<ntim;++i )
        {
            inp >> my_tim[i];
        }
        getline( inp,token );// get to next line
    }



    // read tally values and error
    //cout << "\t" << "reading in "
    //  << ncs*nflag*nuser*nmult*nerg*nseg*ncos*ntim
    //  << " values and errors" << endl;
    inp >> token;
    if ( token.compare("vals") != 0 )
    {
        // cout << "read: " << token << endl;
        throw fatal_error("somethings wrong -- expected \
to read in vals but did not");
    }
    my_val.resize( ncs*nflag*nuser*nseg*nmult*ncos*nerg*ntim );
    my_err.resize( ncs*nflag*nuser*nseg*nmult*ncos*nerg*ntim );
    // cs = cell/surface bin index
    // f = flag bin index
    // u = user bin index
    // s = segment bin index
    // m = multiplier bin index
    // c = cosine bin index
    // e = energy bin index
    // t = time bin index

    int nuxtra = 0;
    if ( usertotal )
        nuxtra++;

    int nsxtra = 0;
    if ( segtotal )
        nsxtra++;

    int nmxtra = 0;
    if ( multtotal )
        nmxtra++;

    int ncxtra = 0;
    if ( costotal )
        ncxtra++;

    int nextra = 0;
    if ( ergtotal )
        nextra++;

    int ntxtra = 0;
    if ( timtotal )
        ntxtra++;

    for ( int cs=0;cs<ncs;++cs )
    {
        for ( int f=0;f<nflag;++f )
        {
            for ( int u=0;u<nuser+nuxtra;++u )
            {
                for ( int s=0;s<nseg+nsxtra;++s )
                {
                    for ( int m=0;m<nmult+nmxtra;++m )
                    {
```

```
                        for ( int c=0;c<ncos+ncxtra;++c )
                        {
                            for ( int e=0;e<nerg+nextra;++e )
                            {
                                for ( int t=0;t<ntim+ntxtra;++t )
                                {
                                    readDataPoint( inp,cs,
                                        f,u,s,m,c,e,t );
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    // read in tfc (not implemented)
    int curpos = inp.tellg();
    inp >> token;
    //cout << token << endl;
    if ( token.compare("tfc") == 0 )
    {
        int nlines;
        inp >> nlines;
        getline( inp,token );
//      cout << token << endl;
        for ( int i=0;i<nlines;++i )
        {
            getline( inp,token );
            //cout << token << endl;
        }
    }
    else
    {
        inp.seekg(curpos);
    }
}

void mtally::readDataPoint( ifstream& inp,int cs,int f,
    int u,int s,int m,int c,int e,int t )
{
    if ( cs >= ncs ||
        u >= nuser ||
        s >= nseg ||
        m >= nmult ||
        c >= ncos ||
        e >= nerg ||
        t >= ntim )
    {
        double tempdouble;
        // then we exceeded the bounds of the array
        // (usually due to totals over all bins)
        tempdouble = readfloat(inp); // val
        inp >> tempdouble; // err
    }
    else
    {
        // only need to do this for shielding sims, some values
        // with exponents > 100 are printed like 1.04+44
        // without the "E"
        val(cs,f,u,s,m,c,e,t) = readfloat(inp);
        //inp >> val(cs,f,u,s,m,c,e,t);
        inp >> err(cs,f,u,s,m,c,e,t);
    }
}

void mtally::operator= ( const mtally& tal )
{
    resize( tal.ncs,
        tal.nflag,
        tal.nuser,
        tal.nseg,
        tal.nmult,
        tal.ncos,
        tal.nerg,
```

```
        tal.ntim );
    my_erg = tal.erg();
    my_cos = tal.cos();
    my_tim = tal.tim();
    my_val = tal.val();
    my_err = tal.err();
}

void mtally::sumUserBins( )
{
    const int CS = ncs;
    const int F = nflag;
    const int U = nuser;
    const int S = nseg;
    const int M = nmult;
    const int C = ncos;
    const int E = nerg;
    const int T = ntim;
    for ( int cs=0;cs<CS;++cs )
    {
        for ( int f=0;f<F;++f )
        {
            for ( int s=0;s<S;++s )
            {
                for ( int m=0;m<M;++m )
                {
                    for ( int c=0;c<C;++c )
                    {
                        for ( int e=0;e<E;++e )
                        {
                            for ( int t=0;t<T;++t )
                            {
                                for ( int u=1;u<U;++u )
                                {
                                    const double tal1val
                                        = val( cs,f,0,s,m,c,e,t );
                                    const double tal1err
                                        = err( cs,f,0,s,m,c,e,t );
                                    const double tal2val
                                        = val( cs,f,u,s,m,c,e,t );
                                    const double tal2err
                                        = err( cs,f,u,s,m,c,e,t );
                                    val( cs,f,0,s,m,c,e,t )
                                        = tal1val+tal2val;
                                    err( cs,f,0,s,m,c,e,t )
                                        = sqrt(pow(tal1val*tal1err,2.0)
                                        +pow(tal2val*tal2err,2.0));
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}


mtally combine( const mtally& tal1,const mtally& tal2 )
{
    // make sure the tallies can be combined
    if ( tal1.ncs != tal2.ncs
        || tal1.nflag != tal2.nflag
        || tal1.nuser != tal2.nuser
        || tal1.nseg != tal2.nseg
        || tal1.nmult != tal2.nmult
        || tal1.ncos != tal2.ncos
        || tal1.nerg != tal2.nerg
        || tal1.ntim != tal2.ntim )
    {
        throw fatal_error("cannot combine tallies that do not match");
    }


    mtally result;
```

```
const int CS = tal1.ncs;
const int F = tal1.nflag;
const int U = tal1.nuser;
const int S = tal1.nseg;
const int M = tal1.nmult;
const int C = tal1.ncos;
const int E = tal1.nerg;
const int T = tal1.ntim;

result.resize( CS,F,U,S,M,C,E,T );
result.erg() = tal1.erg();
result.cos() = tal1.cos();
result.tim() = tal1.tim();

const double eps = 1e-30;
for ( int cs=0;cs<CS;++cs )
{
    for ( int f=0;f<F;++f )
    {
        for ( int u=0;u<U;++u )
        {
            for ( int s=0;s<S;++s )
            {
                for ( int m=0;m<M;++m )
                {
                    for ( int c=0;c<C;++c )
                    {
                        for ( int e=0;e<E;++e )
                        {
                            for ( int t=0;t<T;++t )
                            {
                                const double tal1val
                                    = tal1.val( cs,f,u,s,m,c,e,t );
                                const double tal1err
                                    = tal1.err( cs,f,u,s,m,c,e,t );
                                const double tal2val
                                    = tal2.val( cs,f,u,s,m,c,e,t );
                                const double tal2err
                                    = tal2.err( cs,f,u,s,m,c,e,t );
                                // check to make sure tallies are non zero
                                if ( tal1val < eps )
                                {
                                    result.val( cs,f,u,
                                        s,m,c,e,t ) = tal2val;
                                    result.err( cs,f,u,
                                        s,m,c,e,t ) = tal2err;
                                    continue;
                                }
                                else if ( tal2val < eps )
                                {
                                    result.val( cs,f,u,
                                        s,m,c,e,t ) = tal1val;
                                    result.err( cs,f,u,
                                        s,m,c,e,t ) = tal1err;
                                    continue;
                                }

                                // If both are non-zero,
                                // then we can compute appropriate
                                // weights to combine the files

                                // compute sum of 1/variance^2
                                double abserr1 = tal1err * tal1val;
                                double abserr2 = tal2err * tal2val;
                                const double var1 = abserr1 * abserr1;
                                const double var2 = abserr2 * abserr2;
                                const double sum = 1/var1 + 1/var2;
                                // now take inverse of the sum
                                // to get the "total variance"
                                const double totvar = 1/sum;

                                // weights are 1/variance * totvar

                                const double w1 = 1/var1 * totvar;
                                const double w2 = 1/var2 * totvar;
```

```
                                    result.val( cs,f,u,s,m,c,e,t )
                                        = w1*tal1val + w2*tal2val;
                                    result.err( cs,f,u,s,m,c,e,t )
                                        = sqrt( totvar )
                                        /result.val( cs,f,u,
                                        s,m,c,e,t );
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return result;
}

// subtracts the second flagged bin from the first flagged bin
// used for shielding tallies
mtally subtract( const mtally& tal )
{
    mtally result;

    const int CS = tal.ncs;
    const int F = tal.nflag;
    const int U = tal.nuser;
    const int S = tal.nseg;
    const int M = tal.nmult;
    const int C = tal.ncos;
    const int E = tal.nerg;
    const int T = tal.ntim;

    if ( F < 2 )
    {
        return tal;
    }

    result.resize( CS,F-1,U,S,M,C,E,T );
    result.erg() = tal.erg();
    result.cos() = tal.cos();
    result.tim() = tal.tim();

    for ( int cs=0;cs<CS;++cs )
    {
        for ( int u=0;u<U;++u )
        {
            for ( int s=0;s<S;++s )
            {
                for ( int m=0;m<M;++m )
                {
                    for ( int c=0;c<C;++c )
                    {
                        for ( int e=0;e<E;++e )
                        {
                            for ( int t=0;t<T;++t )
                            {
                                result.val( cs,0,u,s,m,c,e,t )
                                    = tal.val( cs,0,u,s,m,c,e,t )
                                    - tal.val( cs,1,u,s,m,c,e,t );
                                double abserr1 =
                                    tal.err( cs,0,u,s,m,c,e,t )
                                    *tal.val( cs,0,u,s,m,c,e,t );
                                double abserr2 =
                                    tal.err( cs,1,u,s,m,c,e,t )
                                    *tal.val( cs,1,u,s,m,c,e,t );
                                result.err( cs,0,u,s,m,c,e,t ) =
                                    sqrt(abserr1*abserr1
                                    + abserr2*abserr2);
                            }
                        }
                    }
                }
            }
        }
    }
    return result;
```

```
}


mtally operator+ ( const mtally& a,const mtally& b )
{
    mtally result;

    const int CS = b.ncs;
    const int F = b.nflag;
    const int U = b.nuser;
    const int S = b.nseg;
    const int M = b.nmult;
    const int C = b.ncos;
    const int E = b.nerg;
    const int T = b.ntim;

    result.resize( CS,F,U,S,M,C,E,T );
    result.erg() = b.erg();
    result.cos() = b.cos();
    result.tim() = b.tim();

    for ( int cs=0;cs<CS;++cs )
    {
        for ( int f=0;f<F;++f )
        {
            for ( int u=0;u<U;++u )
            {
                for ( int s=0;s<S;++s )
                {
                    for ( int m=0;m<M;++m )
                    {
                        for ( int c=0;c<C;++c )
                        {
                            for ( int e=0;e<E;++e )
                            {
                                for ( int t=0;t<T;++t )
                                {
                                    result.val(cs,f,u,s,m,c,e,t)
                                        = a.val(cs,f,u,s,m,c,e,t)
                                        + b.val(cs,f,u,s,m,c,e,t);
                                    double abserr1 =
                                        a.err( cs,f,u,s,m,c,e,t )
                                        *a.val( cs,f,u,s,m,c,e,t );
                                    double abserr2 =
                                        b.err( cs,f,u,s,m,c,e,t )
                                        *b.val( cs,f,u,s,m,c,e,t );
                                    result.err( cs,f,u,s,m,c,e,t ) =
                                        sqrt(abserr1*abserr1
                                        + abserr2*abserr2);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return result;
}



std::vector< mtally > parsemfile( const std::string& filename )
{
    using namespace std;

    //cout << "parsing .m file \"" << filename << "\"" << endl;
    ifstream inp( filename.c_str() );
    if ( ! inp.good() )
    {
        throw fatal_error("could not open .m file");
    }

    string temp;
    // get version,nps,etc
    getline( inp,temp );
```

```
    if ( temp.empty() )
    {
        throw fatal_error("mctal_file_is_empty");
    }
    // get title
    getline( inp,temp );
    // get number of tallies
    int ntal;
    inp >> temp;
    inp >> ntal;
    // read tally numbers
    vector< int > tnum( ntal );
    //cout << "found " << ntal << " tallies: ";
    for ( int i=0;i<ntal;++i )
    {
        inp >> tnum[i];
        //cout << tnum[i] << " ";
    }
    //cout << endl;
    getline( inp,temp ); // get to next line


    // begin reading in tallies
    vector< mtally > tal(ntal);
    for ( int i=0;i<ntal;++i )
    {
//      cout << "parsing tally " << tnum[i] << endl;
        tal[i].parse( tnum[i],inp );
    }

    return tal;

}
```

Listing B.70: mtally.hpp

```
#ifndef _mtally_hpp_
#define _mtally_hpp_

#include <cstdlib>
#include <vector>
#include <string>
#include <fstream>

#include "errh.hpp"


 class mtally
{
    public:

    void parse( int,std::ifstream& );

    void readDataPoint( std::ifstream&,int,int,int,
        int,int,int,int,int );

    double& val( int,int,int,int,int,int,int,int );
    std::vector<double>& val( );
    double& err( int,int,int,int,int,int,int,int );
    std::vector<double>& err( );
    double& erg( int );
    std::vector<double>& erg( );
    double& cos( int );
    std::vector<double>& cos( );
    double& tim( int );
    std::vector<double>& tim( );
    double val( int,int,int,int,int,int,int,int ) const;
    std::vector<double> val( ) const;
    double err( int,int,int,int,int,int,int,int ) const;
    std::vector<double> err( ) const;
    double erg( int ) const;
    std::vector<double> erg( ) const;
    double cos( int ) const;
    std::vector<double> cos( ) const;
    double tim( int ) const;
    std::vector<double> tim( ) const;

    int ncs; // number of cell/surface bins
    int nflag; // number of flagged bins
    int nuser; // number of user bins
    int nseg; // number of segment bins
    int nmult; // number of multiplier bins
    int ncos; // number of cosine bins
    int nerg; // number of energy bins
    int ntim; // number of time bins
    std::vector< double > my_erg;
    std::vector< double > my_cos;
    std::vector< double > my_val;
    std::vector< double > my_err;
    std::vector< double > my_tim;

    void resize( int,int,int,int,int,int,int,int );

    void operator= ( const mtally& );

    void sumUserBins();



    mtally( ) {};
    ~mtally( ) {};

    private:

    bool my_userCum;
    bool my_segCum;
    bool my_multCum;
    bool my_cosCum;
    bool my_ergCum;
    bool my_timeCum;
```

```
    int idx( int,int,int,int,int,int,int,int );
    int idx( int,int,int,int,int,int,int,int ) const;

    double readfloat( std::ifstream& );

    int surfnum;
    int my_tnum;
};

std::vector< mtally > parsemfile( const std::string& );
mtally combine( const mtally& tal1,const mtally& tal2 );
mtally subtract( const mtally& tal );
mtally operator+ ( const mtally&,const mtally& );

#endif
```

Listing B.71: phys.cpp

```
#include "phys.hpp"

#include <cmath>

namespace phys
{

    // energy in MeV
    // theta in radians
    // returns scattering energy in MeV
    double scaterg( double erg,double theta )
    {
        return erg/(1+erg/me*(1-cos(theta)));
    }

    // energy in MeV
    // mu in cosine angle
    // returns cross section per electron
    double sigkn( double erg,double mu )
    {
        //const double re = 2.8179e-13;
        //const double re = 1.0;
        const double lambda = erg/me;
        const double f = 1.0/(1.0+lambda*(1.0-mu));
        return (1.0/2.0)*re*re*f*f*( f+1.0/f-pow(sin(acos(mu)),2) );
    }

    // non-relativistic particles only!!!
    // energy in MeV
    // mass in MeV/c^2
    // returns velocity in cm/s
    double velocity( double erg,double mass )
    {
        return sqrt(2.0*erg/mass)*c*100.0;
    }

}
```

Listing B.72: phys.hpp

```
#ifndef _phys_hpp_included_
#define _phys_hpp_included_


namespace phys
{
    const double pi = 3.14159265358979323846;  // pi
    const double me = 0.510998910; // electron rest mass (MeV/c^2)
    const double avo = 6.0221415e23; // avogadro's number
    const double re = 2.8179e-13; // classical electron radius
    const double c = 2.99792458e8; // speed of light in a vacuum (m/s)
    const double mn = 939.565560; // neutron rest mass (MeV/c^2)


    double scaterg( double erg,double theta );
    double sigkn( double erg,double mu );
    double velocity( double erg,double mass );

}


#endif
```

Bibliography

[1] National Nuclear Security Administration (NNSA). NNSA Second Line of Defense Factsheet, February 2010. http://www.nnsa.energy.gov/mediaroom/factsheets/nnsassecondlineofdefenseprogram.

[2] Department of Homeland Security (DHS). DHS and DOE Launch Secure Freight Initiative, December 2006. http://www.dhs.gov/xnews/releases/pr_1165520867989.shtm.

[3] Customs and Border Protection (CBP). An Overall Picture of Port Security, July 2006. http://www.cbp.gov/xp/cgov/newsroom/fact_sheets/port_security/securing_us_ports.xml.

[4] G.G. Thoreson. A Framework for Efficient Detection Probability Computation in Smuggled Nuclear Material Interdiction. Master's thesis, The University of Texas at Austin, May 2009.

[5] Thermo Fisher Scientific. TPM-903 Transportable Radiation Portal Monitor. http://www.fishersci.com.

[6] SAIC. AT-900 Series Radiation Portal Monitor (RPM) Product Brochure. http://www.saic.com/products/security/pdf/AT-900s.pdf.

[7] R.C. Runkle et. al. Analysis of Spectroscopic Radiation Portal Monitor Data Using Principal Components Analysis. *IEEE Transactions on Nuclear Science*, 53(3), June 2006.

[8] B. Ayaz-Maierhafer and T.A. DeVol. Determination of absolute detection efficiencies for detectors of interest in homeland security. *Nuclear Instruments and Methods in Physics Research A*, 579:410–413, 2007.

[9] The Royal Society. Detecting nuclear and radiological materials, July 2008.

[10] E.R. Siciliano et. al. Comparison of PVT and NaI(Tl) scintillators for vehicle portal monitor applications. *Nuclear Instruments and Methods in Physics Research A*, 550:647–674, 2005.

[11] E.R. Siciliano et. al. Energy calibration of gamma spectra in plastic scintillators using Compton kinematics. *Nuclear Instruments and Methods in Physics Research A*, 594:232–243, 2008.

[12] Department of Homeland Security. *Advanced Spectroscopic Portal Monitors*, 2004. BAA 05-04.

[13] G.F. Knoll. *Radiation Detection and Measurement*. John Wiley & Sons, Inc., 3rd ed. edition, 2000.

[14] G.W. Phillips et. al. A Primer on the Detection of Nuclear and Radiological Weapons. *Center for Technology and National Security Policy*, May 2005.

[15] Saint Gobain. NaI(Tl) and Polyscin NaI(Tl) Sodium Iodide Scintillation Material. http://www.detectors.saint-gobain.com/uploadedFiles/SGdetectors/Documents/Product_Data_Sheets/NaI(Tl)-Data-Sheet.pdf.

[16] S.M. Robinson et. al. Optimal Background Attenuation for Fielded Radiation Detection Systems. *IEEE Transactions on Nuclear Science*, 54(4), August 2007.

[17] R.J. Nikolic et. al. Roadmap for High Efficiency Solid-State Neutron Detectors. volume 6013, October 2005.

[18] R.T. Kouzes et. al. Passive neutron detection for interdiction of nuclear material at borders. *Nuclear Instruments and Methods in Physics Research A*, 584:383–400, 2008.

[19] J.L. Jones et. al. Detection of shielded nuclear material in a cargo container. *Nuclear Instruments and Methods in Physics Research A*, 562:1085–1088, 2006.

[20] C.E. Moss et. al. Comparison of Active Interrogation Techniques. *IEEE Transactions on Nuclear Science*, 53(4), August 2006.

[21] J.L. Jones et. al. Proof-of-Concept Assessment of a Photofission-Based Interrogation System for the Detection of Shielded Nuclear Material, November 2000.

[22] K.A. Jordan and T. Gozani. Detection of U-235 in hydrogenous cargo with Differential Die-Away Analysis and optimized neutron detectors. *Nuclear Instruments and Methods in Physics Research A*, 579:388–390, 2007.

[23] D.R. Slaughter. The nuclear car wash: A system to detect nuclear weapons in commercial cargo shipments. *Nuclear Instruments and Methods in Physics Research A*, 579:349–352, 2007.

[24] Los Alamos National Laboratory. *MCNP A General Monte Carlo N-Particle Transport Code, Version 5*, April 2003. LA-CP-03-0245.

[25] Los Alamos National Laboratory. *MCNPX User's Manual v2.6.0*, April 2008. LA-CP-07-1473.

[26] M.A. Descalle et. al. *COG Validation: SINBAD Benchmark Problems*. Lawrence Livermore National Laboratory, March 2004. UCRL-TR-202654.

[27] S. Agnostinelli et. al. GEANT4 - a simulation toolkit. *Nuclear Instruments and Methods in Physics Research A*, 506:250–303, 2003.

[28] T.A. Wareing et. al. ATTILA, A Three-Dimensional, Unstructured Tetrahedral Mesh Discrete-Ordinates Code. *Transactions of the American Nuclear Society*, 75:146–147, 1996.

[29] R.D. Busch. *A Primer for Criticality Calculations with DANTSYS*. Los Alamos National Laboratory, August 1997. LA-13265.

[30] R. Alcouffe. PARTISN Calculations of 3D Radiation Transport Benchmarks for Simple Geometries with Void Regions. *Progress in Nuclear Energy*, 39(2), 2001.

[31] S.M. Bowman et. al. SCALE 5: Powerful New Criticality Safety Analysis Tools. *The 7th International Conference on Nuclear Criticality Safety*, 2003.

[32] E.I. Novikova et. al. Designing SWORD - SoftWare for Optimization of Radiation Detectors. *IEEE Nuclear Science Symposium Conference Record*, 1:607–612, 2006.

[33] L.E. Smith et. al. Deterministic Transport Methods for the Simulation of Gamma-Ray Spectroscopy Scenarios. *IEEE Nuclear Science Symposium Conference Record*, 14, 2006.

[34] G.A. Warren et. al. Evaluation Framework for Search Instruments. *IEEE Nuclear Science Symposium Conference Record*, 14(17), 2005.

[35] S.M. Robinson et. al. A Simulation Framework for Evaluating Detector Performance in Cargo Screening Applications. *IEEE Nuclear Science Symposium Conference Record*, 2006.

[36] D.J. Mitchell. Preparation of Inject Data Using GADRAS. Presented at the LANL Threat Reduction Workshop, 2008.

[37] C.A. LoPresti et. al. Baseline suppression of vehicle portal monitor gamma count profiles: A characterization study. *Nuclear Instruments and Methods in Physics Research A*, 562:281–297, 2006.

[38] Evaluated Nuclear Structure Data File. `http://www.nndc.bnl.gov/ensdf/`.

[39] H. Yang and D.K. Wehe. Detection of Concealed Special Nuclear Material Using Nuclear Resonance Fluorescence Technique. *IEEE Nuclear Science Symposium Conference Record*, 2009.

[40] J.K. Shultis and R.E. Faw. *Radiation Shielding*. American Nuclear Society, Inc., 2000.

[41] Lawrence Livermore National Laboratory. *RadSrc Library and Application Manual*, Nov. 2007. UCRL-TM-229497.

[42] P. Goldhagen. personal communication, 2011.

[43] M.S. Gordon et. al. Measurement of the Flux and Energy Spectrum of Cosmic-Ray Induced Neutrons on the Ground. *IEEE Transactions on Nuclear Science*, 51(6), 2004.

[44] JEDEC. *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, October 2006.

[45] D.F. Smart and M.A. Shea. World Grid of Calculated Cosmic Ray Vertical Cutoff Rigidities for Epoch 1995.0. *30th International Cosmic Ray Conference*, 2007.

[46] Chuck Plaxico et. al. U08: Finite Element Analysis Crash Model of Tractor-Trailers, August 2009.

[47] P. Sokkappa et. al. *Benchmark Sources for Radiation Detection Architecture Evaluation [revision 2]*. Lawrence Livermore National Laboratory, January 2007. UCRL-TR-218277-REV-2.

[48] C. Petrillo et. al. Solid state neutron detectors. *Nuclear Instruments and Methods in Physics Research A*, 378:541–551, 1996.

[49] G. D'Antona. Measurement Data Processing Using Random Matrices: A Generalized Formula for the Propagation of Uncertainty. *IEEE Transactions on Instrumentation and Measurement*, 53(2), April 2004.

[50] Richard A. Johnson. *Miller & Freund's Probability and Statistics for Engineers*. Prentice-Hall Inc., 6th ed. edition, 2000.

[51] R. Kouzes et. al. Naturally occurring radioactive materials in cargo at US borders. *Packing, Transport, Storage & Security of Radioactive Materials*, 17(1), 2006.

Vita

Gregory George Thoreson, the son of Barbara Evans Thoreson and George Thoreson was born in Pennsylvania on January 17th, 1985. He grew up in Texas where he attended The Woodlands High School. Shortly after graduating Summa Cum Laude in 2003, he enrolled in the Nuclear Engineering program at Texas A&M University. After four years, he graduated Summa Cum Laude, and spent the summer at Los Alamos National Laboratory researching time discretization techniques. He entered the Mechanical Engineering graduate program at The University of Texas at Austin in 2007 with a focus in the area of Nuclear and Radiation Engineering. He spent his summers between semesters at Los Alamos National Laboratory researching homeland security topics. After receiving his Master's degree in 2009, he continued on to finish his Doctorate of Philosophy in the same field in 2011.

Permanent address: 23 Markham Grove Pl.
The Woodlands, TX 77381

This dissertation was typeset with LaTeX† by the author.

†LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.