LA-13744-T
Thesis

Development and Implementation of Photonuclear Cross-Section Data for Mutually Coupled Neutron-Photon Transport Calculations in the Monte Carlo N-Particle (MCNP) Radiation Transport Code

Los Alamos
NATIONAL LABORATORY

This thesis was accepted by the Graduate Faculty of the College of Engineering, and the Graduate School of the University of Florida, Gainesville, Florida, in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The text and illustrations are the independent work of the author and only the front matter has been edited by the CIC-1 Writing and Editing Staff to conform with Department of Energy and Los Alamos National Laboratory publication policies.

LA-13744-T
Thesis

Issued: July 2000

Development and Implementation of Photonuclear Cross-Section Data for Mutually Coupled Neutron-Photon Transport Calculations in the Monte Carlo N-Particle (MCNP) Radiation Transport Code

Morgan C. White

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545

ACKNOWLEDGMENTS

As with any large research project, there are a myriad of people who deserve thanks for their contributions to this work. I would like to try to thank as many of you by name as possible and sincerely apologize if I miss anyone who should have been included. This has been a long hard stretch of my life, but it was time well spent thanks to your support and care.

The first words of thanks must go to my mentor and friend, Samim Anghaie. As chairman of both my masters and doctoral research committees, he has guided me through my opening steps into a new world. Personally, he has been an inspiration. I know few people of as fine character and spirit.

To my committee members, my sincerest appreciation for your time and patience as we have progressed down this path. Without your encouragement and support, this work would never have been possible. I would like to thank Jatindar Palta, Frank Bova and Wolfgang Tome for their patience with an engineer trying to understand the medical physics world; Paul Fishwick for his guidance in the world of computer simulations; and Robert Hanrahan for his unique perspective. A special thanks is due Bob Little, Stephanie Frankle and Mark Chadwick. Their contributions in helping me understand the use of nuclear data for Monte Carlo simulations turned this work into a fulfilling project of which I am truly proud.

The list of people with whom I have had useful discussions is quite long and I would like to thank all of them. I would like to acknowledge by name Grady Hughes, Larry Cox, Ken Adams, Chris Werner, Tom Booth, Dick Prael, Jack Comly, Joann Campbell, Judy Briesmeister, Jeff Favorite, Art Forster, Henry Lichtenstein, John Hendricks, and Gregg McKinney for the many discussions and insights on the Monte Carlo method. Thanks to Indrin Chetty, John Demarco, Tim Solberg, Jim Smathers, Paul Deluca and Dave Rogers for the discussions on medical physics and simulating medical electron accelerators. Thanks to Sam Iverstin and Bill Vernetson for their assistance and the use of the NAA facility at UF. Thanks to Joon Park, Texin Lee and Seyong Kim for their assistance in performing the irradiation experiments around the Phillips MEA.

I would also like to acknowledge the support of those people who have been there for the day to day grind at both the University of Florida and the Los Alamos National Laboratory. Thanks to Beth Bruce, Joan Morehouse, Christine Jolly and Ann Nagy for their assistance in making life easier. Thanks to Ed Dugan, Don Shirk, Alexandra Heath and Tom Seed for their encouragement and support.

I would also like to acknowledge several sources of monetary support. First, thanks to the U.S. Department of Energy. In one way or another, the DOE has paid for my time at graduate school. First, through a four-year Nuclear Engineering fellowship as administered by the Oak Ridge Institute for Science and Education and more recently as a graduate research assistant at LANL. I also need to thank two other resources. The project for the Accelerator Production of Tritium sponsored the creation of photonuclear data here at LANL. This was in part due to the diligent and farsighted efforts of Laurie Waters. Many thanks. Also, thanks to the Advance Computing Laboratory at LANL. Many of the calculations presented here were performed using spare CPU cycles from the Blue Mountain supercomputer.

Finally, I want to thank my family. To my mother and father, you have been the most wonderful parents a son could have. You taught me the power of imagination and persistence. To my brother, many thanks for reminding me to always look at things from a different perspective. And at last, thanks to my wife Sarah for her patience, encouragement and love. I love you all.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

DEVELOPMENT AND IMPLEMENTATION OF PHOTONUCLEAR CROSS-SECTION DATA FOR MUTUALLY COUPLED NEUTRON-PHOTON TRANSPORT CALCULATIONS IN THE MONTE CARLO N-PARTICLE (MCNP) RADIATION TRANSPORT CODE

by

Morgan C. White

ABSTRACT

The fundamental motivation for the research presented in this dissertation was the need to development a more accurate prediction method for characterization of mixed radiation fields around medical electron accelerators (MEAs). Specifically, a model is developed for simulation of neutron and other particle production from photonuclear reactions and incorporated in the Monte Carlo N-Particle (MCNP) radiation transport code. This extension of the capability within the MCNP code provides for the more accurate assessment of the mixed radiation fields.

The Nuclear Theory and Applications group of the Los Alamos National Laboratory has recently provided first-of-a-kind evaluated photonuclear data for a select group of isotopes. These data provide the reaction probabilities as functions of incident photon energy with angular and energy distribution information for all reaction products. The availability of these data is the cornerstone of the new methodology for state-of-the-art mutually coupled photon-neutron transport simulations.

The dissertation includes details of the model development and implementation necessary to use the new photonuclear data within MCNP simulations. A new data format has been developed to include tabular photonuclear data. Data are processed from the Evaluated Nuclear Data Format (ENDF) to the new class ‘u’ A Compact ENDF (ACE) format using a standalone processing code. MCNP modifications have been completed to enable Monte Carlo sampling of photonuclear reactions. Note that both neutron and gamma production are included in the present model.

The new capability has been subjected to extensive verification and validation (V&V) testing. Verification testing has established the expected basic functionality. Two validation projects were undertaken. First, comparisons were made to benchmark data from literature. These calculations demonstrate the accuracy of the new data and transport routines to better than 25 percent. Second, the ability to calculate radiation dose due to the neutron environment around a MEA is shown. An uncertainty of a factor of three in the MEA calculations is shown to be due to uncertainties in the geometry modeling. It is believed that the methodology is sound and that good agreement between simulation and experiment has been demonstrated.

CHAPTER 1
INTRODUCTION

At the beginning of this work, several of the graduate students and faculty from both the Nuclear and Radiological Engineering Department and the Department of Radiation Oncology at the University of Florida had begun an intensive exploration of the radiation environment around typical radiotherapy equipment. The goals in mind were more accurate measurement and simulation of these radiation environments. During the course of these early studies, it was observed that the simulation of photonuclear interactions had not received the systematic treatment that electron, photoatomic and neutron interactions receive in the generally available radiation transport codes. This presented a ripe opportunity for a doctoral project and the work you read today is the final result.

The initial course of action was to determine what tools were available for the task of simulating photonuclear interactions. It was quickly determined that the single greatest obstacle to performing photonuclear simulations was the lack of complete double-differential cross sections describing the reaction cross sections as well as the secondary particle emission spectra from the reactions. Complete, evaluated tabular cross-section data are the fundamental keys to performing high-accuracy Monte Carlo radiation transport simulations. As a result of the lack of such data, the concept proposed at the beginning of this work was to make the best of what was available by developing a method for using experimental photoneutron data to assess interaction probabilities and nuclear modeling to estimate emission spectra. Then, using this data, a working simulation would be implemented for use in making general predictions of the neutron flux in the vicinity of high-energy medical electron accelerators (MEAs).

The importance of the assessment of the neutron field around MEAs is a long-standing subject of debate. It has been known since early in the development of high-energy electron accelerators that they create neutrons as a by-product. The term by-product is used because neutrons are typically viewed as a contaminant, not an asset. Since MEAs have become the workhorse of the radiotherapy community, the production of neutrons from these machines is a significant concern. In the use of MEAs for radiotherapy, the desire is to minimize the harm caused by the neutron dose to the patient and workers during the necessary delivery of electron-photon dose to treat cancerous growths.

The most intense investigation of the neutron production and transport around MEAs was during the 1970's and early 1980's when the electron energies used in routine treatments began to climb above the threshold for significant photoneutron production. The two most noteworthy publications on this subject date from that time. In 1979, the National Bureau of Standards held a conference devoted to examining the production of neutrons from MEAs [1]. Several years later, in 1984, the National Council on Radiation Protection and Measurements (NCRP) released a report [2] documenting their recommendations for assessing risks associated with the production and transport of neutrons within a MEA treatment room. However, despite a wealth of studies produced before, during and since those seminal works, the systematic treatment of the simulation of photoneutron production has not been addressed, hence the defining motivation for the current work.

It should be noted that this work has been approached from the perspective of nuclear and radiological engineering, not necessarily that of medical physics. To reinforce the difference, the goal of this work is clearly stated here. This work seeks to provide a systematic treatment of photoneutron production as a part of the simulation of electron accelerators. It is an applied objective in the sense that the end product is a tool capable of simulating the production and transport of neutrons around a MEA and quantifying the uncertainty. The final product may then be used by future researchers to continue efforts in this field aimed at understanding, evaluating and reducing neutron contamination around electron accelerators.

Early in this work, it was decided that the Monte Carlo N-Particle (MCNP) radiation transport code [3] would be used as the base for the development of a simulation code including photoneutron production. This choice was made for a number of reasons. First and foremost, it was desired to build upon an well-established code in order to take advantage of existing validated algorithms. In addition, the author was already familiar with the use of MCNP for transport simulations.

MCNP provided the desired base for starting an effort to integrate photonuclear physics into a radiation transport code. It already included the algorithms and data necessary for modeling electron, photoatomic and neutron transport. More than that, it has been the product of hundreds of man-years of development dating back to the very originators of the Monte Carlo radiation transport method [4]. This development work has been verified and validated through the efforts of thousands of users for a wide range of problems. Specifically, MCNP includes both an electron-photon and a neutron transport package, each of which have a well-established history of use. In fact, as will be discussed in the following chapter, MCNP had already been used for simple uncoupled simulation of photoneutron production and subsequent neutron transport. Therefore, all that remained was to formalize the coupling between the two transport packages and provide verification and validation of the new functionality.

However, the modification of a large code is a daunting task. MCNP is 40,000 lines of highly interdependent, extremely terse Fortran77 code. When it was first developed, the formal idea of software management had not yet been conceived. Therefore, after floundering among the bits, the author approached the X-5 group at the Los Alamos National Laboratory (LANL), who maintain the MCNP code and its tabular data, about the possibility of a collaboration to pursue this work. The reception this idea received was much more than expected.

For the author, this work embodies the saying about being in the right place at the right time. As discussed above, the primary obstacle to accurate simulation of photonuclear interactions has been the lack of complete, evaluated data necessary for tabular Monte Carlo sampling. Not only were the staff at LANL interested in pursuing a collaboration, they were willing to install the author onsite with access to the MCNP development team and, more importantly, access to first-of-a-kind evaluated photonuclear data. The full significance of this will be discussed in the following chapter.

The remainder of this dissertation is composed of four main chapters and the conclusions. The next chapter provides a basic understanding of the mechanics of photonuclear physics and the data and models available to describe them. While there is not an abundance of experimental data or validated nuclear models, it is important to discuss what is available. In particular, the creation of the newly available evaluated photonuclear data is discussed. As part of this chapter, previous photonuclear studies will be mentioned to indicate how the available data have determined the fidelity of simulation ability possible.

The third chapter begins the original work presented here for consideration. It opens with a brief discussion on the data needed to perform a Monte Carlo simulation. From there it progresses through the development of the necessary formats for storing the data, how evaluated data are manipulated into those formats and, finally, how the data are actually used within the transport code. This chapter is the core of the work presented. The developments discussed meet and exceed the original goal of providing a systematic treatment of photoneutron production.

With this new ability to perform fully coupled photonuclear simulations, the next step was to estimate the uncertainty in the use of the simulation with the current evaluated data. The fourth chapter presents the concepts of verification and validation as methods to assess how well the newly developed simulation capability is able to calculate neutron production from high-energy electrons incident on materials for which evaluated data exist. It presents two sets of yield measurements found in the literature and shows comparisons to the current calculated values. Conclusions about the uncertainty in the new capability are drawn from these comparisons.

As the original motivation for this work was the more accurate simulation of medical electron accelerators, the fifth chapter presents an initial assessment of one of the accelerators currently in use at Shands Cancer Center at the University of Florida. The difficulties in modeling such facilities are discussed and an estimate is made about the uncertainties involved based on experimental measurements made around the MEA during this work. The final chapter summarizes the developments and conclusions of this work.

CHAPTER 2
BACKGROUND

Introduction

The single greatest obstacle to accurate simulation of photonuclear interactions within the confines of Monte Carlo radiation transport has been the lack of evaluated, complete data. The use of probabilities to sample interaction rates and resultant products is the key defining feature of Monte Carlo transport. This can be done either through nuclear models or by tabular data. Evaluated tabular data contain the most accurate description of the data available as determined by an evaluator based on judgement of the experimental measurements and nuclear modeling available. For tabular data, complete indicates that in addition to reaction cross sections, all resultant products are given with energy and angular emission spectra as a function of incident particle energy.

Evaluated data are based on the best judgement of a data evaluator. This small field of researchers has in-depth knowledge of both the experimental data and the nuclear models available to describe nuclear reactions for an incident particle on a given target nucleus. Both experimental data and nuclear models are required for this process. Experimental measurements are the best descriptions of physical reality as they demonstrate measured fact. However, experimental measurements are difficult to obtain and never cover the full regime of interest. Nuclear models are complete descriptions of interactions based on theory. Both are subject to error. Therefore, the evaluator must use experience and judgement to meld theory and experiment into the best available description of the data.

For transport simulations, it is necessary to have complete descriptions of the interactions. Cross sections describe the interaction probability for a particle traversing a material as a function of the incident particle's energy. Emission spectra describe the energy and angle of the secondary particles resulting from an interaction once it has occurred. It is in the estimation of this second set of information that nuclear models are essential. They can provide self-consistent complete descriptions of the emission spectra.

Evaluated, complete tabular data are generally considered the most accurate description of the interactions available. Until very recently such data have not existed for photonuclear interactions. As will be discussed in the section of this chapter entitled Current Developments, several groups of evaluators, both in the United States and internationally, have recently provided such data. Before discussing the newly available data, it is worth stepping back and taking a brief look at the physics of photonuclear interactions and the history of the experimental data and simulation models describing them.

First it is necessary to clarify the use of the term data. "Data" is a much abused word and it must be placed in context for it to be truly meaningful. Within the body of this work, the phrase "experimental data" refers to measurements made under laboratory conditions. "Theoretical data" refers to values computed using some form of an analytical model based strictly on theory or guided by, but not directly taken from, experimental data. The term "evaluated data" has been described above. "Tabular data" indicate values listed at discrete points in the phase space. "Benchmark data" are theoretical, experimental or calculated data considered to be correct. To confuse the issue, data often fall into more than one category.

Physics of Photonuclear Interactions

It is important to spend a few moments discussing the physics behind photonuclear reactions in order to provide a context for describing the information contained within the evaluated data. The description presented here is not intended to be a comprehensive explanation of the nuclear physics underlying this phenomena. Rather it is intended to be an illustrative description presenting the basic concepts and providing useful references for further details.

A photonuclear interaction begins with the absorption of a photon by a nucleus. There are many mechanisms by which this can occur. The data currently available focus on the energy range up to 150 MeV incident photon energy. The value of 150 MeV was chosen as this energy is just below the threshold for the production of pions and the subsequent need for much more complicated nuclear modeling. Below 150 MeV, the primary mechanisms for photoabsorption are the excitation of either the giant dipole resonance or a quasi-deuteron nucleon pair. A conceptual illustration of these processes is given in Figure 2-1.

The giant dipole resonance (GDR) absorption mechanism can be conceptualized as the electro-magnetic wave, the photon, interacting with the dipole moment of the nucleus as a whole. This results in a collective excitation of the nucleus. It is the most intense process by which photons interact with the nucleus. It occurs with highest probability when the wavelength of the photon is comparable to the size of the nucleus. However, this resonance is peaked with a width of only a few MeV. For deformed

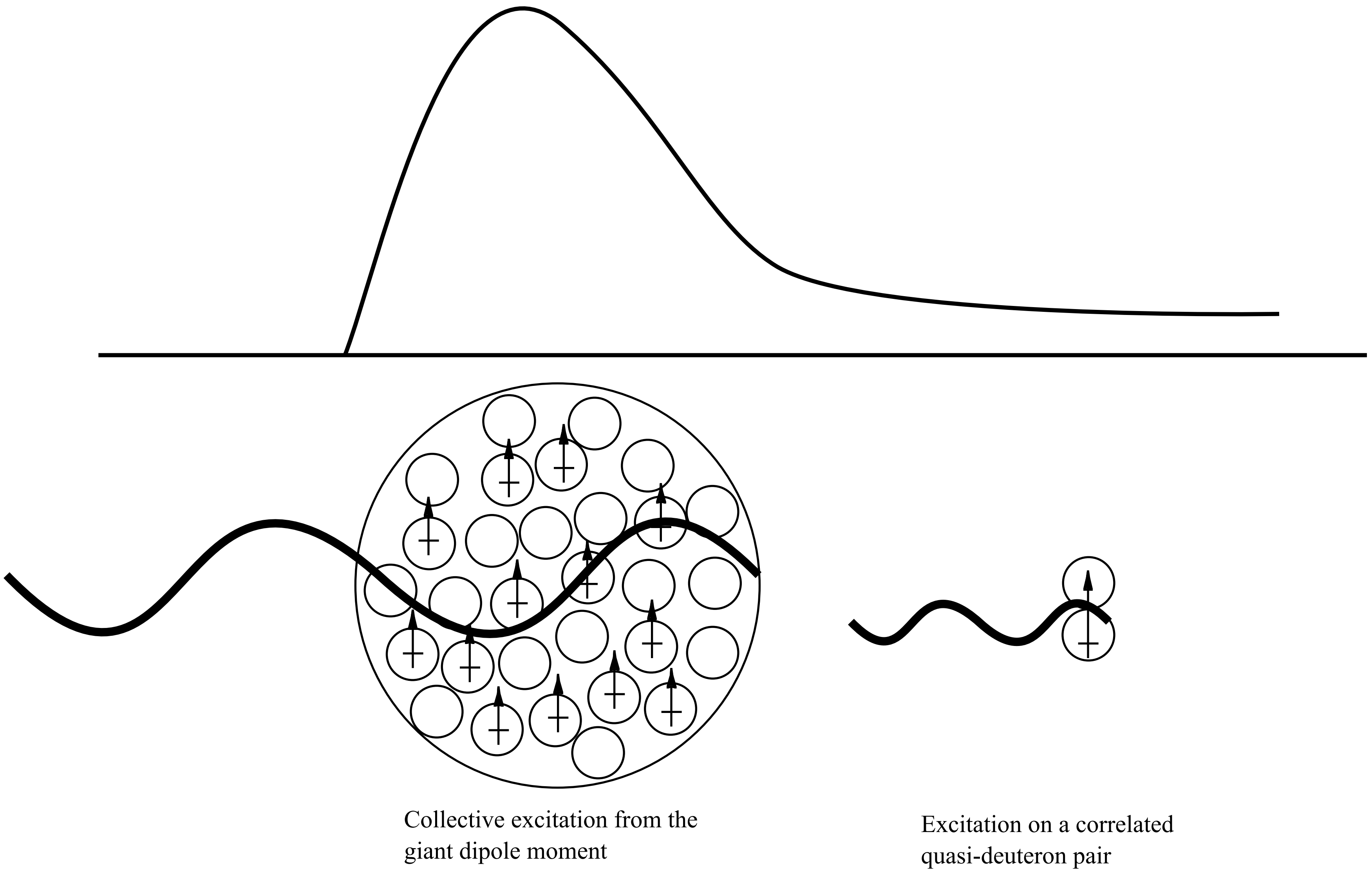
Collective excitation from the giant dipole moment
Excitation on a correlated quasi-deuteron pair

Figure 2-1. Illustrated representation of the giant dipole resonance and quasi-deuteron absorption mechanisms.

nuclei, a double peak is seen due to the variation of the nuclear radius. Outside of the peak region, the GDR reactions are negligible. A more complete description of this process, and of nuclear physics in general, can be found in the text by Bohr and Mottelson [5].

The quasi-deuteron (QD) absorption mechanism can be conceptualized as the electro-magnetic wave interacting with the dipole moment of a correlated neutron-proton pair. In this case, the neutron-proton pair can be thought of as a quasi-deuteron having a dipole moment with which the photon can interact. This mechanism is not as intense as the GDR but it provides a significant background cross section over all incident photon energies. The seminal work describing this process was published by Levinger [6,7]. Recent efforts to model this process includes the work of Chadwick et al. [8].

Once the photon has been absorbed by the nucleus, one or more secondary particle emissions can occur. The secondary particles typically emitted for the energy range below 150 MeV are neutrons, protons, deuterons, tritons, helium-3 or alphas, or a combination thereof. Any emission process that does not leave the residual nucleus in the ground state will also produce secondary gamma-ray emission. The photonuclear threshold for the production of a given secondary particle is governed by the separation energy of that particle. Pre-equilibrium and equilibrium emission are responsible for most of the secondary particles emitted by photon interactions over the energy range under discussion though direct particle emission is possible.

Pre-equilibrium emission can be conceptualized as a particle within the nucleus that receives a large amount of energy from the absorption mechanism and escapes the binding force of the nucleus after at least one but very few interactions with other nuclei.

Typically this occurs from QD absorption of the photon where the incident energy is initially split between the neutron-proton pair. Particles emitted by this process tend to be characterized by higher emission energies and forward-peaked angular distributions. Several references are available on the general emission process after photoabsorption [9-11].

Equilibrium emission can be conceptualized as particle evaporation. Typically this process occurs after the available energy has been generally distributed among the nucleons. In the classical sense, particles boil out of the nucleus as they penetrate the nuclear potential barrier. The barrier may contain contributions from coulomb potential for charged particles and effects of angular momentum conservation. It should be noted that for heavy elements, evaporation neutrons are emitted preferentially as they are not subject to the coulomb barrier. Particles emitted by this process tend to be characterized by isotropic angular emission and evaporation energy spectra. The same references [9-11] apply as for pre-equilibrium emission.

For all of the emission reactions discussed thus far, the nucleus will most probably be left in an excited state. It will subsequently relax to the ground state by the emission of one or more gamma-rays. The gamma-ray energies follow the well known patterns for relaxation. The only reactions which do not produce gamma-rays are direct reactions where the photon is absorbed and all available energy is transferred to a single emission particle leaving the nucleus in the ground state.

Reactions at higher energies, greater than 150 MeV, require more complete descriptions of the underlying nuclear physics. The delta resonance and other absorption mechanisms become significant and the amount of energy involved in the reaction presents the opportunity for the production of more fundamental particles, e.g. pions. While beyond the scope of this current work, descriptions of the physics involved can be found in the paper by Fasso et al. [12].

The study of photonuclear physics has been important for two communities, nuclear physics and health physics. Obviously, the current work approaches this subject from the viewpoint of the second community. As such, the descriptions above have tried to convey a general picture of the mechanisms governing photonuclear interactions. To bring this into the overall picture of photon transport, the probability that a photon will undergo a photonuclear interaction is not more than, and typically much less than, seven percent of the total photon interaction probability. However, this mechanism can provide a source of nuclear particles, specifically neutrons, that constitute a significant health physics risk.

Experimental Photonuclear Data

Experimental measurements provide the fundamental values describing interaction probabilities, i.e. reaction cross sections, and the subsequent yield and spectra of secondary particles. Unfortunately, there are relatively few accurate measurements of the photonuclear reaction cross sections in comparison to the measurements that have been made for neutron reactions. The vast majority of the available experimental data are the result of unfolding measurements made by bremsstrahlung irradiation. These data suffer from extremely large uncertainties due to the unfolding process and are not generally accurate enough to be used on their own as a basis for evaluated data. The cross section measurements using tagged bremsstrahlung or photon emission from in-flight positron annihilation are generally considered to be highly accurate. However, relatively few measurements of this type have been made. Further, similar to neutron data, these measurements typically do not include photonuclear secondary emission spectra. For the experimental data that are available, several compilations similar to the Neutron Barn Book [13] exist.

The first comprehensive listing of experimental data was produced by Fuller working at the National Bureau of Standards (NBS) and is generally known as the Photonuclear Data Sheets [14,15]. This compilation includes references to all known publications of experimental data at that time. Dietrich and Berman published two editions of the Photonuclear Atlas [16,17]. These compilations contain only measurements made with photons produced from the in-flight annihilation of positrons or tagged bremsstrahlung. The most recent compilation of photonuclear data has been carried out by Varlamov et al. [18]. These are the primary references for locating the available experimental photonuclear data.

As most of the references in these publications contain the measured data in the form of plots, it is worth mentioning two further resources. Dietrich and Berman created an electronic tabulation of the cross-section measurements. The tabulation is available from the authors. The EXFOR database [19] maintained by the National Nuclear Data Center contains tabular listings of many, but not all, of the reported measurements. Varlamov et al. have tried to update the EXFOR database with the data from their compilation but it still contains only a small percentage of the reported data.

Previous Photonuclear Studies

This section provides a brief description of some of the known studies that have assessed photonuclear interactions for various reasons. It is not a comprehensive review of the available literature but rather provides examples of the type of work performed in past. It is described here in order to provide a context by which to show how the current work has advanced the previously available capabilities.

One of the first studies in this field was the work of Alsmiller et al. [20-22] carried out at the Oak Ridge National Laboratory in the late 1960's. These calculations were performed using an intra-nuclear-cascade model to estimate the neutron yield and spectra from 150 MeV electrons incident on selected materials. This work was carried out in part to help select the target material for use as a photoneutron source in the Oak Ridge Electron Linear Accelerator Facility (ORELA). Subsequent work by this group included extending their code, known as PICA, to handle higher-energy incident photons [23,24]. Similar studies have been performed by Hansen et al. [25] and Kase et al. [26].

The FLUKA [12], MARS [27] and CEM [28] codes provide Monte Carlo sampling of photonuclear interactions in the medium- to high-energy regime. These codes compute the interaction probabilities and secondary particle emission spectra on-the-fly during the transport process from nuclear models. Recent work on the FLUKA radiation transport code is particularly noteworthy in that it uses empirical fits to experimental data to extend its applicability to the GDR (low-energy) region.

These codes are all alike in the sense that they suffer from inaccuracies inherent in estimating the photoabsorption process. While there are systematic trends in the photoabsorption data, there can be significant dissimilarities for some isotopes that neither theoretical nuclear models nor empirical fits reproduce. These dissimilarities are most noticeable at lower energies, especially around the GDR peak, and for lighter isotopes. However, these codes are primary used for intermediate- to high-energy (100's of MeV to TeV) particle-accelerator modeling and as such the inaccuracies inherent in the models at low energies are not of great concern. It should be noted that transport based primarily on nuclear models has the advantage that reactions can be computed on any target material and emission descriptions are available for all secondary particles.

Many studies needing an accurate assessment of neutron production from photons in the low-energy regime have used a different methodology. The photon flux in a given geometry can be calculated either by analytical theory or by an electron-photon transport code. This flux can then be folded with an experimental photoneutron cross section to estimate the neutron production. Emission energy and angle spectra are assumed and the neutron source "manufactured" in this manner may undergo further transport. Examples of studies using this method include works by Swanson [29-33], McCall et al. [2], Manfredotti et al. [34], Agosteo et al. [35,36], Gallmeier [37], Liu et al. [38] and Chadwick et al. [39]. These studies have focused on estimating the health physics effects of neutrons produced from low-energy electron accelerators. The use of this method for that purpose is difficult due to the meticulous care necessary in coupling the procedures and significant error is possible if done inappropriately.

The are many difficulties in using experimental data in an uncoupled simulation. Experimental data must be found covering the relevant reactions, isotopes and energy range of interest to the simulation. Where such data do not exist, interpolation should be used but introduces an unknown source of error. Experimental data describing emission spectra do not generally exist. Therefore, emission spectra must be estimated using nuclear modeling or other techniques. The simple models that have been used in past to estimate these distributions often do not fully represent the true emission spectra.

Simulations using this methodology are typically run in a series of incremental steps. First, the photon flux is used to estimate the neutron production. Error can be introduced at this step by the inadequacies in the description of the photon flux, the experimental photoneutron cross-section data and the folding method used to estimate the neutron source. The neutron source is then given energy and angular distributions and subsequent neutron transport is performed. Error can be introduced at this step by the inadequacies in the spatial description of the sampled neutron source and in the energy and angular distributions and their appropriate assignment to the source neutrons. Lastly, the statistical correlation to estimate the uncertainty in the simulation are lost because the transport is not fully coupled. The works referenced in the previous paragraph have attempted to solve one or some of these problems. The current work seeks to address all of these problems.

Current Developments

The greatest obstacle to Monte Carlo simulation of fully coupled photon-neutron transport using tabular data has recently been overcome. Several projects at the Los Alamos National Laboratory (LANL) have discovered the need to account for photonuclear interactions in accelerator environments. Therefore, the Nuclear Theory and Applications Group (T-2) of the Theoretical Division was commissioned by the Accelerator Production of Tritium (APT) project to produce a number of photonuclear evaluations as part of the LA150 data library [40]. These evaluations were created using the GNASH nuclear model code [41] as guided by experimental data. The library produced contain complete, evaluated data for incident neutrons, protons and photons (for photonuclear reactions) for the energy range up to 150 MeV. For a limited set of isotopes, evaluated data were created in association with the MCNPX code to significantly advance the state-of-the-art in Monte Carlo radiation transport.

To backtrack for a moment, the Monte Carlo N-Particle (MCNP) radiation transport code [3] has the goal of being the most accurate simulation code for neutron-photon-electron radiation transport available. It seeks to accomplish this goal by the use of tabular evaluated data. Interestingly, these same data define the scope of MCNP's applicability. MCNPX [42] has the goal to extend the region of applicability of MCNP to those particles and energies present around high-energy accelerators. It seeks to accomplish this goal by incorporating tabular evaluated data where available and supplementing the tabular data with nuclear models where necessary. Both of these codes desire the incorporation of photonuclear interactions to enable the coupled simulation of photon-neutron transport problems. This work presents the development and implementation of the newly available evaluated photonuclear data for that purpose.

The research community at LANL is not the only group which has recognized the need to provided evaluated photonuclear data. It is interesting to note that the International Atomic Energy Agency (IAEA) has created a Coordinated Research Project (CRP) entitled "Compilation and Evaluation of Photonuclear Data for Applications" [43]. A library containing photonuclear evaluations of 160 isotopes, including some of those produced by T-2 at LANL, will be released in 2000 together with documentation in an IAEA report [44]. This library should provide sufficient tabular data to perform most simulations where photonuclear reactions are of interest.

CHAPTER 3
IMPLEMENTATION: COUPLING PHOTONUCLEAR PHYSICS INTO MCNP(X)

Introduction to Tabular Monte Carlo Radiation Transport

Monte Carlo radiation transport as defined within the scope of the Monte Carlo N-Particle (MCNP) code is the transport of radiation through a geometry by random sampling of *tabular* interaction probabilities. MCNPX is built upon the same foundation but includes the extension to use nuclear models to generate interaction probabilities if tabular data do not exist. The focus of this chapter is to show the steps necessary to provide and use evaluated tabular photonuclear data within MCNP(X). MCNP(X) is used through-out this chapter in reference to both MCNP and MCNPX. A general familiarity with either of these codes is assumed in the following discussion.

Many steps are necessary to implement photonuclear physics within the MCNP(X) code. First the photonuclear data must be available in a format that can be used in a transport code. Traditionally, that means raw evaluated data stored as Evaluated Nuclear Data Files (ENDF) must be processed into A Compact ENDF (ACE) table suitable for Monte Carlo sampling of interaction probabilities. This processing is typically necessary to transform data structures into more easily sampled forms.

Once the data are available in an appropriate ACE format, they must be loaded into the code at runtime and used by the collision routines. This involves defining a user interface to specify which data tables are to be used, extending the i/o routines to store the new data and integrating new algorithms to sample photonuclear collisions.

Provisions must be made within these steps to ensure that existing capabilities, particularly tallies, work correctly and that summary tables include relevant information about the sampling of photonuclear interactions.

The sections of this chapter provide the detailed step-by-step account of the steps taken by this work. Key concepts and algorithms are explained along with the intricate details, e.g. the name of the internal MCNP variable containing the table index. The level of information included is meant to be exhaustive. These are the details necessary such that upon reading the actual coding, the intent of specific code is obvious. This will facilitate the maintenance and extension of this code. It is necessary to document these details here because the MCNP coding style (that was followed for this work) dictates that terse code with minimal comments should be used. Between this philosophy and the sheer complexity of the existing code base, it is often difficult to follow what ought to be simple routines.

Data Storage

Photoatomic Versus Photonuclear Data

Data storage was the first major issue addressed by this work. The MCNP radiation transport code name derives originally from the fact that it was capable of *n*eutron and *p*hoton transport. It later became *N*-*P*article when electron transport was added. MCNPX extends further the latter meaning behind the name. However, only photoatomic interactions have been treated in past: photoelectric absorption, elastic scattering, inelastic scattering and pair production. (Note that the cross section for triplet production is included in pair production and treated identically.) The MCNP(X) ACE files containing these data descriptions are tabulated as cross sections by element.

Photonuclear interactions, indeed all nuclear interactions, are dependant on the specific target nucleus. This creates the first problem. Similar to neutron data, photonuclear data should be tabulated and used by isotope, not by element.

As the separate storage and use of photoatomic and photonuclear data is considered counterintuitive by some, several additional arguments are made to enforce why this should be. Photoatomic and photonuclear data evaluations are not typically from the same source. Most experimentalists, theoreticians and evaluators concentrate on providing data for one or the other, not both. Compilations of the data are generally separate. Photoatomic data are usually updated as a complete, consistent library for all elements. Photonuclear data are expected to follow the path of neutron data where updates occur for individual isotopes as they become available. If they were stored by element, it would be necessary to create a new ACE data set every time either was updated.

For these reasons and more, photonuclear data is stored and accessed separately from photoatomic data. This philosophy removes the necessity of determining how to mix elemental and isotope data in the same storage table and generally provides for easier maintenance and access of the different ACE data sets. However, it also necessitates the creation of a new class of ACE table to contain the photonuclear data for use in MCNP(X).

Standard ACE Tables

The tabular data tables used by MCNP(X) are known by the acronym ACE. This acronym stands for *A* *C*ompact *E*NDF. The Evaluated Nuclear Data File (ENDF) is a collection of formats for storing data and procedures for creating and sampling that data

[45]. It is the de facto international standard for storing nuclear data and is maintained by the National Nuclear Data Center (http://www.nndc.bnl.gov/nndc/) at the Brookhaven National Laboratory. The ACE table format contains this data in a form more suitable for random sampling by a transport code.

There are currently eight classes of ACE data tables in use by MCNP: continuous-energy neutron 'c', discrete-reaction neutron 'd', neutron dosimetry 'y', S(α,β) thermal 't', continuous-energy photon 'p', continuous-energy electron 'e', multigroup neutron 'm' and multigroup photon 'g' tables. MCNPX extends this to nine classes with continuous-energy proton 'h' tables. Photonuclear tables will now add a new class of data to MCNP(X). As this new data describes characteristic nuclear interactions, the photonuclear table format draws heavily on the continuous-energy neutron and proton table formats. Therefore it is useful to examine how the continuous-energy format has evolved for storing nuclear interaction data.

ACE tables use a system of parameters and locators to access data stored in a one-dimensional array. Table 3-1 shows the standard structure for an ACE table. As originally conceived, the NXS array stores all parameters, the JXS array stores all locators and the XSS array contains the actual data. In a library file, each set of tabular data has its own header with NXS and JXS entries and its own XSS array. In the MCNP(X) executable code, a single XSS array contains all tabular data necessary for the simulation and NXS/JXS are two-dimensional arrays (by array entry and table index).

For describing a single incident and emitted particle type, a fixed NXS/JXS array size is a reasonable solution. The original neutron data tables used only five parameters and twelve locators to do this. Photon production data were added later by duplicating

Table 3-1. Standard ACE table description.

Line Address		Contents	Format (Fortran Standard)
Relative	Absolute		
1	IRN	ZAID, Atomic Weight, Temperature, Date Processed	A10, 2E12.0,1X, A10
2	IRN+1	Comment	A80
3 – 6	IRN+2 – IRN+5	Inherited fields currently unused (Fill with zeros or leave blank)	4(I7, F11.0) per line
7 – 8	IRN+6 – IRN+7	(NXS(I):I=1..16)	8(I9) per line
9 – 12	IRN+8 – IRN+11	(JXS(I):I=1..32)	8(I9) per line
13 - …	IRN+11 – …	(XSS(I):I=1..LXS)	4(E20.0) per line

key elements (parameters and locators for secondary emission data) to reference photon production data. This required three new parameters and eight new locators. Two additional locators for neutron data were also added to form the table which exists for use by MCNP4B [3, Appendix F]. More recently, delayed neutron data added yet another parameter and four new locators [46].

MCNPX expands the capabilities of MCNP to include tracking light-ions as well as other particles of interest to high-energy particle accelerators [42]. Whereas MCNP only expects to transport secondary photons and neutrons, MCNPX transports all light particles. To do this, the data libraries needed to be expanded to include emission information for an arbitrary number of new secondary particles. Currently, these new neutron tables include, as appropriate, secondary emission data for proton, deuteron, triton, helium-3 and alpha particles in addition to neutron and photon data. This placed a burden on the neutron continuous-energy tables that could not be solved by using the traditional NXS/JXS framework. It was solved by escaping the box and introducing the IXS array.

The IXS array was introduced into the neutron continuous-energy format to store locators to secondary charged-particle emission information [47,48]. Unlike NXS and JXS that are fixed in length, the IXS array is stored within the XSS array and can be expanded to contain entries for as many sets of secondary-particle information as needed. In the expanded neutron table, the JXS array is used to locate the neutron and photon production data and the IXS array for all other secondary particles. Proton tables were adapted in this format as well. The photonuclear data table takes the next logical step and references all secondary-emission information through the IXS array.

Photonuclear Class 'u' ACE Table

Photonuclear interactions describe photon-induced nuclear processes. The evaluated files used to store photonuclear data make use of the same ENDF formats and procedures as neutron and proton data. This implies that the same concepts used in ACE tables to store other nuclear data should be used for the photonuclear table such that the existing storage and sampling algorithms can be used for all types of nuclear interaction. However, the established neutron/proton table has become extremely convoluted. Therefore, photonuclear tables have logically reorganized the data to significantly simplify access.

The photonuclear format has been modified from the existing neutron/proton format to treat all secondary-particle production in a self-consistent manner. However, this table still draws heavily on the sub-formats established for neutron and proton data. The remainder of this section focuses on presenting the key concepts for the photonuclear class 'u' ACE table format. This description is supplemented by Appendix A that includes the full details of all appropriate data structures, how they are stored, what error checking can be performed and recommendations on use of specific sub-formats for photonuclear data.

The NXS array now contains only those parameters that apply to the table as a whole. The NXS parameters are presented in Table 3-2. The first entry is the length of the XSS array. This entry is mandatory for all ACE tables such that they may be manipulated in a generic fashion. The second entry contains a target identifier and is standard for those ACE tables where it is applicable. The next three entries contain the only three global parameters needed for transport: the number of energies in the main energy grid, the number of cross sections included in the table and the number of secondary particles for which emission data are included.

Secondary particle information consists of parameters and locators. The parameter NEIXS (in conjunction with NTYPE) can now be used to determine the memory requirement to store the IXS array elements. The IXS array has twelve entries in this format version. The parameter NPIXS is the number of secondary-emission

Table 3-2. Description of the NXS Array elements in a photonuclear class ‘u’ ACE format.

Entry	Parameter	Fixed numeric descriptive
NXS(1)	LXS	Length of the XSS data block
NXS(2)	ZA	Atomic and mass number of the target isotope ZA = Z*1000 + A
NXS(3)	NES	Number of energy entries in the main energy grid
NXS(4)	NTR	Number of reaction cross sections
NXS(5)	NTYPE	Number of secondary particle types with emission information
NXS(6)	NPIXS	Number of parameter entries in the IXS array 2 of 12 IXS entries in the current format are parameters
NXS(7)	NEIXS	Number of entries in IXS array per secondary particle The current table format includes 12 IXS entries
NXS(8-15)		Unused (Fill with value zero)
NXS(16)	TVN	Table Format Version TVN=1 for the current table format

parameters at the start of each IXS array. There are two parameters in this format version. All other IXS array entries are assumed to be locators and are subject to updates as data are moved within memory in the MCNP(X) executable at runtime.

The table format version parameter (TVN) is the first attempt at documenting each table-type format as it is produced. This marks photonuclear format number one. If it is changed later, e.g. expanded to hold a new sub-format, the table-format-version would be updated at that time to indicate what information may be within the table. This also introduces a mechanism whereby backwards compatibility can be maintained without rigidly enforcing the exact table structures.

Similar to the NXS array, the JXS array has also been reduced to contain only those locators that are general to the table. The JXS array entries are presented in Table 3-3. The first five entries for this table are locators for reaction data that traditionally has been accessed through the overloaded ESZ entry. As the overall number of JXS entries

Table 3-3. Description of the JXS Array elements in a photonuclear class ‘u’ ACE format.

Entry	Locator	Offset to array of…
JXS(1)	ESZ	Main energy grid
JXS(2)	TOT	Total cross-section data
JXS(3)	NON	Total non-elastic cross-section data
JXS(4)	ELS	Elastic cross-section data
JXS(5)	THN	Total heating number data
JXS(6)	MTR	MT reaction numbers
JXS(7)	LQR	Q-value reaction energy data
JXS(8)	LSIG	Cross-section locators (relative to SIG)
JXS(9)	SIG	Primary locator for cross-section data
JXS(10)	IXSA	First word of IXS array
JXS(11)	IXS	First word of IXS block
JXS(12-32)		Unused (Fill with zeros)

has been reduced, it is felt that overloading the ESZ locator was unnecessary. Therefore, each set of reaction data is accessed through its own locator.

By tradition the first locator, ESZ, is the index in the XSS array for the main energy grid. The four other reaction data sets that have traditionally been accessed through this entry now have individual locators. The total cross-section is now located by second entry, TOT. The energy and total cross-section data are the fundamental values necessary for computing the distance-to-collision during photon transport.

The elastic cross section is now located by the fourth entry, ELS. The elastic cross section for photons interacting with the nucleus is negligible in comparison to other photonuclear reactions. The evaluated data files are not required to include it. Therefore, since this locator has isolated the elastic cross-section entries, if the elastic cross section is not included in the original evaluation, its locator is set to zero, no entries are made in the XSS block and the elastic scattering of photons on the nucleus is ignored during the transport process.

The absorption cross section has typically been included for the purpose of biasing. For shielding problems, it is sometimes useful to simulate capture implicitly. In this type of simulation, only non-absorption reactions are considered at the collision site and the particle weight is updated accordingly. This biasing technique ensures that a neutron or photon always leaves the collision site. This is the default treatment for both neutron and photoatomic interactions.

Photonuclear absorption almost always produces a secondary gamma ray. Only photonuclear processes involving a transition directly to the ground state of the nucleus do not. Additionally, the secondary particle of interest from the photonuclear interaction is rarely, if ever, a photon. Therefore, the implicit capture biasing technique is not useful and the absorption cross section has been replaced by the non-elastic cross section.

The non-elastic cross section is located by the third entry, NON. This change has the benefit that when the elastic cross section is not present, the non-elastic cross section is identically the total cross section. Therefore, the locators can be set to index the same data, thereby reducing storage needs.

The total heating numbers are located by the fifth entry, THN. Energy deposition is often of interest and the F6 tally in MCNP(X) uses the heating numbers for this purpose. The total heating number is an estimate of the average amount of energy the incident particle deposits locally at the collision site. There are numerous assumptions involved in calculating this value (see Appendix A for a more complete description). As it is difficult to compute and not used by this work, this locator is given a zero value to indicate that no data are currently available. When the ability to produce class ‘u’ ACE tables migrates into the NJOY nuclear data processing code, it will be possible to compute the heating numbers.

The next four locators provide information about the reaction cross sections. The MTR locator is the index to the reaction-type listing. These are the MT reaction-type numbers as defined in the ENDF format manual [45]. The LQR locator is the index to an array of Q-values corresponding to each reaction. The LSIG and SIG locators index the location of the cross-section arrays.

The IXSA locator is an index to the IXS array. As described above, the IXS array contains the parameters and locators for all secondary-emission information. The meanings of the IXS array entries are described below. The IXS locator is a convenience. The data within an ACE table can be listed in any order desired, so long as the locators are appropriately updated. However, for the sake of sanity, the data should be listed in the order corresponding to their appearance in the table description. If this is done, the IXS locator is the first word of the IXS block of secondary-particle information located within the XSS array.

The IXS array should be thought of as a two-dimensional array containing a set of parameters and locators for each secondary particle. There are NTYPE (1 … J … NTYPE) secondary particle emission descriptions in the IXS block. The IXS array entries are listed for the Jth secondary particle in Table 3-4. Two parameters are necessary for each set of emission data. The secondary particle type is identified by the parameter IPT as described in Table 3-5. The number of reactions that produce this secondary particle is given by parameter NTRP.

Table 3-4. Description of the IXS Array elements in a photonuclear class 'u' ACE format.

Entry	Parameter	Fixed number descriptive
IXS(1,J)	IPT(J)	Particle IPT number
IXS(2,J)	NTRP(J)	Number of MT reactions producing this particle
Entry	Locator	Offset to array of…
IXS(3,J)	PXS(J)	Total particle production cross-section data
IXS(4,J)	PHN(J)	Particle average heating number data
IXS(5,J)	MTRP(J)	Particle production MT reaction numbers
IXS(6,J)	TYRP(J)	Reaction coordinate system data
IXS(7,J)	LSIGP(J)	Reaction yield locators (relative to SIGP)
IXS(8,J)	SIGP(J)	Primary locator for reaction yield data
IXS(9,J)	LANDP(J)	Reaction angular distribution locators (relative to ANDP)
IXS(10,J)	ANDP(J)	Primary locator for angular distribution data
IXS(11,J)	LDLWP(J)	Reaction energy distribution locators (relative to DLWP)
IXS(12,J)	DLWP(J)	Primary locator for energy distribution data

Table 3-5. Association of particles with their symbol and IPT index number as defined in MCNP(X).

Particle Name	Symbol	IPT
neutron	n	1
photon	p	2
electron	e	3
proton	h	9
deuteron	d	31
triton	t	32
helium_3	s	33
alpha	a	34

There are ten locators that have been determined to be necessary for locating secondary-emission data. The locators and parameters are kept separate, parameters first, within the IXS array such that the locators can be updated as described within the Setup and Storage section below. As mentioned above, a full description of the heating number concept is found in Appendix A. Heating numbers are not used by this work and thus all PHN locators have been set to zero and no further values are given.

The locators PXS, MTRP, LSIGP, SIGP and the parameter NTRP are used to determine the secondary-particle production. The production cross section for the secondary particle is located by the entry PXS. The reactions which contribute to the production of this particle are located by the entry MTRP. The yield data for each reaction are contained in the SIGP block as located by the LSIGP array entries. This is a change from the neutron table format which overloaded the TYRP entries to contain yield data as well as the reaction coordinate system. The TYRP array entries designate the reaction coordinate system. The emission distributions are stored in the ANDP and DLWP blocks as located by the offsets in the LAND and LDLWP arrays. The full details of the emission formats are complex. In order to avoid cluttering this chapter more than is already the case, they have been included in Appendix A.

Data Processing

As discussed in the previous section, evaluated data are available in the ENDF format. It is therefore necessary to have a data processing code capable of translating the ENDF formatted data into the appropriate ACE format. This has traditionally been done by the NJOY code [49]. However, as this was an iterative process to develop the new table format, it was preferable to write a stand-alone processing code that was easily changed in order to explore different formatting options.

The MKPNT data processing code was developed to process data stored in the ENDF format into the ACE class 'u' photonuclear format described above and in Appendix A. As this was a developmental tool whose capability will be subsumed by the NJOY code, it was not necessary to implement full functionality for all possible ENDF formats. Instead, MKPNT is focused on the data that were currently available. This section will discuss how the data were processed and what formats were used. The full source code for MKPNT is given in Appendix B.

For the purposes of this work only data from the LA150 library produced at the Los Alamos National Laboratory have been used for simulations. Some preliminary data have been made available from the other institutions involved in the International Atomic Energy Agency (IAEA) Coordinated Research Program (CRP). However, the IAEA data were provided on the condition that they were to be used for testing purposes only. Therefore, some functionality has been implemented in the MKPNT processing code for the IAEA data which was not necessary for this work but looked to long-term goals.

MKPNT works by loading the ENDF data into memory and then creating the corresponding sections in the ACE format. The ENDF format contains information needed for the ACE table in six “files”. Each section of the ENDF format is known as a file and given a file type index MF, e.g. file MF 6. MKPNT was implemented with a limited understanding of the these six files as described below.

File MF 1 contains general information about the data set. The target identifier ZA is the atomic number times 1000 plus the mass number. ZA, the atomic weight ratio (AWR) and the temperature are taken directly from the corresponding entries. ZA is also used to create the table identifier ZAID by adding an ID. The library number plus the table identifier ‘u’, to indicate a photonuclear table, are collective the table ID. The library number is a unique two digit number chosen at the processing time. All ACE tables also include the date processed.

With preliminary data read, the first step in building the table is to form a unified energy grid. The photonuclear data processed to date have contained relatively few, on the order of tens to hundreds, energy grid points. The energy points are obtained as a superset of the energy grids from each of the reaction cross sections. The units must be adjusted from eV, in ENDF, to MeV, in ACE. All reaction cross sections are found in ENDF File MF 3 and are given as energy/cross-section pairs. Reactions which involve thresholds are checked to ensure that the cross-section values start at zero, if necessary adding the new point. Because there are so few points in these files, as compared to certain neutron data sets that can contain tens of thousands of energy points, thinning of the energy grid is not required.

There can be additional energy points contained within yield tables. Yields are found in File MF 6 of the ENDF format. These energy points are added to the main energy grid if present. The addition of these points allows the verification process to exactly match emission spectra. With this done, the total number of energy points, global NXS parameter NES, is now fixed as well as the data entries for array ESZ.

The locators ELS and THN are currently set to zero. None of the evaluated data provided to date have included the elastic cross section. As discussed in the previous section, THN is a complicated value which was not needed for this study. Algorithms exist in NJOY to compute the total and partial heating numbers and it was felt unnecessary to duplicate that capability within MKPNT.

The cross-section data are obtained next. Each reaction cross-section is taken from its file MF 3 entries and stored in the SIG block as located by the LSIG offsets. The corresponding MT number and Q-value are stored in the MTR and LQR arrays, respectively. The number of reactions is stored in the parameter NTR. The total cross section is computed from the appropriate partials and checked against the values from the ENDF MT 1 total cross section. The verified computed totals are then stored in the TOT array. As no elastic cross sections are present, the NON locator is set equal to the same value as TOT.

Secondary-particle emission data can be specified by either of two methods in the original ENDF evaluation. The first method described here is not recommended but still allowed. For reactions which produce neutrons, files MF 4 and MF 5 give the angular distributions and energy distributions, respectively, as a function of incident particle energy. MF 5 energy distributions are useful for representing certain photonuclear interactions, e.g. fission reactions. Currently, MKPNT can process MF 5 laws 5, 7, 9 and 11. The corresponding MF 4 angular distribution must be isotropic. The neutron yields for fission are taken from file MF 2. All other yield data for reactions specified by this method are implicitly given by the reaction type, e.g. reaction MT 16 implies two emission neutrons. The data from these three files are merged appropriately into the neutron secondary-particle information. To date only IAEA data have used this format. It is highly recommended that all photonuclear reactions be described by the following method as all secondary particles, not just neutrons, can be included.

Secondary-particle emission spectra may also be described in the ENDF file MF 6 format. ENDF file MF 6 contains one section describing each reaction with appropriate subsections for every product from that reaction. MKPNT loops over each file MF 6 section and extracts the appropriate secondary-particle emission data into the ACE table. Currently MKPNT extracts emission data for secondary neutrons, photons, protons, deuterons, tritons, helium-3 ions, and alphas. This set of particles represents was chosen as an alpha particle is the heaviest ion that can be transported by MCNPX. Obviously, MCNP is limited to the transport of secondary neutrons and photons. This processing method uses the standard assumption that all other reaction products are stopped at the collision site without producing any further secondary particles of interest.

MKPNT currently processes LAW 1 correlated energy-angle distributions from file MF 6. The yield for each particle is taken directly from the energy/yield pairs given and the data are stored in the SIGP block as located by the corresponding LSIGP offset. Likewise, the MT number for the production reaction and the coordinate system for the emission particle are stored in the MTRP and TYRP arrays, respectively.

The LANG 1 angular option presents a tabular-energy distribution with Legendre coefficients describing angular dependence. MKPNT only processes isotropic distributions and they are stored as tabulated-energy distributions, ACE Energy Law 4, in DLWP with the corresponding offset in LDLWP. The isotropic angular distribution is indicated by a zero value in the LANDP array and no entries in the ANDP block.

The LANG 2 angular option specifies a tabular-energy distribution with Kalbach88 angular systematics. This emission distribution is stored with the appropriate LDLWP offset in the DLWP block as ACE Energy Law 44. The corresponding LANDP entry is given the value negative one to indicate the presence of the correlated energy/angle distribution and no entries are made in the ANDP block. The Kalbach slope parameter, designated ‘a’, is computed according to Chadwick’s correction [11] of Kalbach’s original formalism [50,51] to account for the reduced momentum of the photon.

The LA150 photonuclear evaluations all use the ENDF format MF 6 LAW 1 LANG 2. As the LA150 evaluated data are used exclusively in this study, processing of the LAW1 LANG 2 format has been exhaustively tested. This is a tedious procedure done either by hand or by script to ensure that the ACE data match the original ENDF data. The other options discussed were implemented to process data provided by the IAEA CRP and their processing has not been checked with the same rigor. All ENDF emission distribution formats not specifically mentioned are not supported.

At this stage the ACE table is almost complete. The photonuclear cross sections and the associated yields for each secondary particle are used to compute the total particle-production cross section that is stored in the PXS array. The number of reactions producing the particle is stored as the parameter NTRP. The particle heating-number locator, PHN, is given a value of zero to indicate no heating numbers are included. The final processing verifies the values for the locators, stores the total number of XSS entries as the NXS parameter LXS and prints the final table to an ASCII file.

With the appropriate additions to the MCNP(X) cross-section directory file XSDIR, the data are now ready for use in a simulation. At the time the simulations in this study were performed, the LA150 library contained evaluated photonuclear data for the following isotopes: ^{27}Al, ^{40}Ca, ^{56}Fe, ^{63}Cu, ^{181}Ta, ^{184}W, ^{206}Pb, ^{207}Pb and ^{208}Pb. The evaluated data were processed into the ACE photonuclear format using the MKPNT code as described here. The collection was given the ID 03u as earlier testing had been done using library numbers 01 and 02.

Coupling Photonuclear Physics into MCNP(X)

Introduction

The work presented here describes a prototype code based on MCNP4B2 [3]. MCNP was chosen as the base code for the reasons described earlier. This section describes the modifications that were made to MCNP4B2 to produce the photonuclear-capable prototype code designated MCNP4BPN. Once verification was complete, these changes were frozen and the final version described here was used for all subsequent calculations presented in this work. Where standard MCNP capabilities are discussed, see the users guide [3] if further explanations are necessary.

The MCNP code package is maintained by the X-5 group of the Los Alamos National Laboratory. A project homepage is maintained and can be accessed through the X-division homepage (http://www-xdiv.lanl.gov/). The code package is distributed by to persons and companies within the U.S. by the Radiation Safety Information Computational Center (http://www-rsicc.ornl.gov/rsic.html). Foreign distribution is handled by the Nuclear Energy Agency (NEA) in Paris, France. Changes to this code are to be made via patch files as documented in the users guide [3, p. C-4]. Appendix C of this dissertation contains the patch file corresponding to the changes described here.

Before launching into the gory details of the changes, a few moments will be spent providing an overview. There were four major tasks necessary to implement photonuclear interactions into MCNP: (1) the user interface needed to be modified to allow specification of photonuclear tables for a given material; (2) the nuclear data sampling routines needed modification to appropriately handle particles other than neutrons; (3) the photon collision routines needed to be updated to include sampling photonuclear events; and (4) the file i/o routines needed to be updated to include reading photonuclear tables and printing summary information about photonuclear interactions.

The specification of materials is done in a very standard manner within MCNP. Several needs drove the final interface for photonuclear. First, the standard interface must be kept. However, it was designed with the concept of one table type for each particle type. Further, it assumes there is always a table available for each component of the material.

The solution to the specification of photonuclear tables was to keep the standard interface as is and augment it to work similarly for specification of the new tables. That is, the components of the material are defined by the material card. Component ZAID entries can be specified directly in the entry or indirectly through the ZA with a default or user specified library ID. Neutron, electron, photoatomic and photonuclear table ZAIDs or library IDs are all acceptable.

However, the interface has also been augmented to include an isotope override for photonuclear tables. Specifically, because more complete isotopic data will likely be available for neutron transport, the best description of the material should be given on the material card by neutron table. The new override card allows the isotope to be changed for any or all of the material components. This promotes the use of the best neutron and photonuclear tabular data for a material. It should be considered at some point in the future to allow material specification by incident particle in MCNP(X).

The nuclear data sampling routines were originally written for incident neutron, neutron emission interactions. They were later updated to include photon emission. Recent interest, including the current work, needed to expand the tables to handle incident photons, protons or neutrons and the subsequent emission of any particle type. The set of updates this necessitated is described only briefly below. It is now commonly known as the ACE modifications and has been implemented [52,53] in the current versions of both MCNPX and MCNP [54].

The revision of the photon collision routines to include photonuclear interactions is the key objective of this work. These routines have been updated to include use of the photonuclear cross section, in addition to the photoatomic, for the sampling of distance-to-next-collision. At a photon collision site, either natural or biased collisions can occur. Biased photonuclear collisions indicate that a contribution from photonuclear interaction to secondary-particle production is to be obtained at every photon collision. In either case, secondary particles are sampled from the evaluated tabular data and made available for further transport.

There are no unexpected changes in the file input/output. A past MCNP user will be able to fully utilize this new capability based on their previous experiences using the code. As mentioned before, the standard material interface will work unchanged. All tables are loaded from standardized ACE libraries specified through one XSDIR directory file. The few new interface options available are simple to use when necessary and are not required. All tallies and summary information include the effects of photonuclear interactions as presented in standard MCNP output tables. Thus, the average user can begin using this capability immediately and become an expert user familiar with all the intricacies over time.

Setup and Storage

Material specification. The first task necessary to use the photonuclear data within MCNP was to implement user options to specify which data to load and to store that data appropriately. It was determined that the material specifications to load photonuclear data should be as similar to what was currently done as possible. However, some extensions have also been made.

At present each material has one list of isotopes and atomic fractions associated with it. For example, the material description for an electron target might be given by the MCNP input card "`m1   71000 1`" or by "`m1   71180 0.00012   71181 0.99988`", both of which indicate that material one is natural tantalum. The first specifies elemental tantalum directly and the second specifies the constituent isotopes by their atomic fractions. Since photoatomic data is stored by element, either card could be used to specify which tables, or in this example table, should be used. However, neutron data is stored by isotope, with a few exceptions, such that the second description is the more accurately represents the material. Photonuclear tables are also stored by isotope and therefore more accurately described by isotopic tables.

Unfortunately, very few photonuclear evaluations were available for this study. Even after the IAEA library is made available, not all isotopes will have an evaluated data file. Some prevision is necessary to allow the user to specify the best photonuclear data available without compromising the fidelity of the representation by other tables, in particular neutron tables. Therefore, a photonuclear isotope override card has been implemented.

To illustrate this problem, consider a material input card describing natural tungsten. The best description for neutron transport is given by the material card "`m1 74182 0.263  74183 0.143  74184 0.3067  74186  0.286`." Notice that this description is incomplete; isotopic ^{180}W with a natural atomic fraction of 0.0013 is not included in the description because a neutron table is not available. MCNP will compensate for this by re-normalizing the sum of the other atom fractions to one. However, photonuclear data are currently available only for ^{184}W. Following the logic of drop what is unavailable and re-normalize, the significant contributions by other isotopes for neutron transport would be missed simply because of the lack of photonuclear tables for the remaining isotopes. The desire for the best representation for both neutron and photonuclear interactions in the material requires a new capability in material input specification.

Photonuclear isotope override card (MPN). The photonuclear isotope override card, designated MPN, has been implemented to allow substitution of photonuclear data. Specifically, for the example above the photonuclear isotope override card "`mpn1 74184 74184 74184 74184`" used in conjunction with the material specification card from the previous paragraph would provide the best data for all particles transported through the material. MCNP4BPN would use photoatomic data for elemental tungsten, the four available neutron tables for the major tungsten isotopes and the ^{184}W data table for all photonuclear collisions.

There are several restrictions on the use of the photonuclear isotope override card. It must be used in conjunction with a material specification card, i.e. M1 with MPN1 describes material one. The override card must come after the corresponding material specification card in the standard MCNP input deck. There must be one entry on the MPN card corresponding to each ZAID entry on the M card. Entries on the MPN card must correspond to a ZA for which a photonuclear table exists or be zero to indicate no photonuclear interactions should be considered for that portion of the isotope.

The photonuclear isotope card has been implemented as a new input card. The number of cards for use in the code was increased by one by incrementing the *nkcd* parameter in the deck **jc**. The new card, *cnm(89)*, was initialized in the deck **ibldat** with the option to allow only integer entries. (The use of **boldface** type in this chapter indicates names of subroutines in the MCNP source code. Likewise, *italic* type signifies the names of variables within those subroutines.)

This capability requires an array to store a different ZA for photonuclear interactions than the default for the material. The M card stores ZA/atom-fraction pairs in the arrays *iza* and *fme*, respectively. The new *izn* array mirrors the *iza* array as a storage location in the dynamically-allocated common-block *dac*. They are both set to length *mix*, the number of isotope/fraction pairs for the specific problem, at runtime.

No processing was necessary for the MPN card during the first reading of the input deck. Therefore the routines **newcd1**, **nxtit1** and **oldcd1** ignore this card. Several processing options are necessary during the final reading of the input deck. When the MPN card is first encountered, the routine **newcrd** checks to ensure that photonuclear physics is turned on in the simulation. If photonuclear physics is on, it then checks to ensure that a material card has already been seen describing the material. If either of these conditions is not met, the MPN card is ignored and a warning message is printed.

Each entry for the card is checked and stored individually. From the card initialization set in deck **ibldat**, the entries are automatically checked to ensure that they are integer numbers. The routine **chekit** refines this criteria to ensure that a valid ZA, in the range 000001 to 999999, or zero has been entered. A fatal error is issued for invalid ZA entries. The routine **nextit** then stores each entry in array *izn* to correspond to the appropriate M card entry. Finally, the routine **oldcrd** checks to ensure that the number of entries on the MPN card corresponds to the number of entries on the M card. If they do not match, all isotopic values are reset to the material default and a warning message is printed stating the card was ignored. If the isotope override has been successful, a warning is printed for each isotope override. It is the responsibility of the user to ensure that appropriate substitutions have been made.

Table ID specification. MCNP allows the user to specify the data table ID to be used for each nuclide by several methods. The first method is to describe materials by complete ZAIDs. For example, natural copper can be described by the material specification card "`m1  29063.60c 0.6917  29065.60c 0.3083`." The ".60c" is the table identifier, ID, indicating the neutron class 'c' tables should come from the ENDF60 library, ENDF60 tables having the unique library number 60. Photonuclear tables can be specified in an analogous manner. The table identifier ".00u" can be used to specify the library, with 00 appropriately replaced, from which to load the photonuclear tables.

The second method to specify a data table for an isotope is to use the defaults as defined in the XSDIR directory file. The XSDIR file includes the lookup table used to determine what data tables are available in each library. If no table identifier (ID) has been specified, the first match to ZA for each class of table will be used. For example, if the XSDIR file contains entries for 29063.22c and 29063.60c in that order and the M card asks for ZA 29063 without an ID in a problem transporting neutrons, the 29063.22c table will be used for neutron collisions in that material as it was seen first. Thus, the order of the ZAID entries in the XSDIR file can be used to determine which tables are used in a problem. This is the reason the default XSDIR file distributed with the MCNP code is ordered such that the recommended tables appear first.

Default LIB specifier. The default library used for a table class can be specified using a material option entry on the material specification card. For MCNP, three material options are available to do this. They are the NLIB, PLIB and ELIB options corresponding to the neutron, photoatomic and electron default library specifiers, respectively. The names of the material options are stored in the variable *hmopt* in the character common block of deck **jc** as initialized by deck **ibldat**.

To illustrate the use of the default library specifier, consider the M card "`m1 29063 0.6917  29065.60c 0.3083 nlib=22c plib=01p`." Any combination of material options can be used as needed but they only apply to that M card. The order of precedence for selecting a ZAID is the full ZAID in the entry pair, the ZA from the entry with the ID from the default library specifier or the first appropriate match to the ZA in the XSDIR file. Back to the example, the neutron, photoatomic and electron tables are selected using the standard XSDIR file as follows. The neutrons tables 29063.22c and 29065.60c are used, the first from the ZA and NLIB library specifier and the second specified directly by ZAID. The photoatomic table 29000.01p is used, selected by the ZA shortened to elemental Z and PLIB library specifier. The electron table 29000.01e is used, selected as the first appropriate electron table listed in the XSDIR file.

The material option PNLIB has been added so that the user can specify the photonuclear default library for a material in an analogous manner. This was done by incrementing the number of material options in deck **jc** and adding the string constant 'pnlib' to the variable *hmopt* initialization in deck **ibldat**.

To make use of the PNLIB material option, consider the example M card "`m1 29063 0.6917  29065 0.3083 pnlib=03u`" with the corresponding MPN card "`mpn1  29063 29063`." The override card specifies that ^{63}Cu should be used in place of ^{65}Cu for photonuclear reactions in the second isotope of the material. The PNLIB default library specifier indicates all photonuclear tables should come from the LA150 revision 3 photonuclear data library, with library number 03. Thus, the

29063.03u table would be attached to both material entries for handling photonuclear collisions.

Table selection and storage. The portion of the coding that controls data table selection and storage required extensive changes to enable loading a new class of table. The storage allocation process was completely rewritten. The specific changes are documented here.

The original section of code responsible for determining the storage needs for the cross-section data used a restrictive, complicated algorithm. It contained dependencies that assumed one table type per particle type attached to each material constituent, i.e. only neutron, photoatomic and electron tables. (This is not strictly true as thermal tables augment the neutron data and are stored separately but that is handled as a separate optional exception.) As implemented, this algorithm was not extensible to include a separate photonuclear table required in addition to a photoatomic table. The algorithm's complexity derived from a convoluted process whereby it determined the number of duplicate tables requested in order to reduce the memory allocated for use in table header and storage arrays. This was an unnecessarily complex process for relatively minor savings in total memory needs.

The section of code that reads material specification cards was heavily revised. Several new arrays were introduced to mirror the existing data pointers. The array *izn* has been described above. To reiterate, the array *izn* contains the material isotope listing for the purpose of simulating photonuclear collisions mirroring the array *iza* which contains the default isotope listing for all other isotope/atom-fraction constituents of the material.

All dynamically allocated memory in MCNP is placed in a single long array, *das*, and referenced by offsets. This creates a confusing situation because all dynamically allocated arrays are actually the same array, through either the Fortran77 equivalence or pointer statement. To illustrate this consider the array *jxs*, itself located in the *das* array, contains locators that are indexes into the other arrays also located in the *das* array. Therefore, all arrays must be referenced by their own pointers, e.g. *ljxs*, that contain the index of the first word of the corresponding array.

In the description of the table format in a previous section above, NXS, JXS and IXS are shown as one and two dimensional arrays. In use within the code, arrays *nxs*, *jxs* and *ixs* are given an additional dimension corresponding to their table index, variable *iex*. Each table has a unique index assigned by its order within the array *xss*. Thus, in use the fifth element of the NXS array for the fourth table is found at the location *nxs*(*lnxs*+5,4). Similarly, the third element of the IXS array for the second emission particle in the fifth table is found at the location *ixs*(*lixs*+3,2,5).

There are numerous variables and arrays associated with the table selection and storage. The array *lme* has dimensions of the number of constituents specified on the material cards by the number of particles available for transport, i.e. *mix* by *mipt*. It contains the neutron, photoatomic and electron table indices for each constituent of all materials in the current simulation. The array *lmn* has dimension *mix* and contains the photonuclear table indices. Arrays *iza* and *izn* have dimensions mirroring *lme* and *lmn* and contain the ZA identifiers for the neutron/photoatomic/electron and photonuclear material constituents, respectively. The tables are selected using their ZA, from either array *iza* or *izn*, and their ID, from either the directly specified ID in array *kmm* or the default specifier located in either array *lxd* or *lxn*. The arrays *nxs*, *jxs* and *ixs* contain the entries corresponding to each of the *mxe* tables in the simulation. The arrays *izn*, *lmn*, *lxn* and *ixs* have been added to the appropriate common blocks to mirror the arrays *iza*, *lme*, *lxd* and *nxs/jxs*, respectively. The parameters *maxsec* and *mixs* are added to the deck **zc** to indicate the maximum number of secondary particles per table and the maximum number of IXS array entries, 8 and 12, respectively.

A number of other new variables are also necessary to the task at hand. The variable *ispn* is added to the fixed common block **fixcom** to hold the flag indicating whether photonuclear physics is on or off. The array *pnt* is added to the fixed, dynamically-allocated common block **dac** to contain the lowest photonuclear threshold-energy for each material. The variable *totpn* is added to the task common block **tskcom** to hold the total photonuclear cross-section value for the current photon at its corresponding energy for each task transporting particles. The variable *npum* is added in the variable common block **varcom** for use in printing an error message. The variable *htn* has been added to the character common block in deck **vv** to contain the string 'cdytpmgue' listing the classes of table in their default order.

Several existing variables have been enlarged to contain new values. The arrays *pax* and *paxtc* contain the weight values by particle type for the overall summary tables for the problem as a whole and the individual tasks, respectively. Their dimensions have been incremented by one, from 16 to 17, to store photoabsorption and photoproduction information by particle. The array *jrwb* is the mapping from particle termination type, *nter*, to the appropriate storage location in the array *pax*. Its dimensions have also been incremented by one to account for photoabsorption termination of the photon. The array *pwb* contains event information for each particle type for each cell. Its dimensions have been incremented in size by two, from 19 to 21, to store photoabsorption and photophoton production for photons as well as photoproduction for other particles. The array *pan* stores interaction activity information by table, currently only photoatomic and neutron, for each cell. Its dimensions have been expanded to include space for photonuclear tables, first index from 2 to 3, as well as new entries, second index from 6 to 8, for the additional photonuclear interactions.

Now that the key variables are known, the initialization algorithms can be described. MCNP starts a problem by reading the default input file “inp”. It makes two passes through the cards in the file. The term card derives from the days of punch cards and is simply a single line of input. The first pass sets up the storage necessary to process the input and stores a few key user input parameters. The second pass stores the remaining user input values.

Since photonuclear interactions is a new capability to MCNP(X), it was decided that during the time it is a beta test capability the user should have to turn on photonuclear physics. This was done in large part so that the existing regression test suite could be used without change. The default may eventually be changed to have photonuclear physics on such that the best available transport algorithms are used unless the user turns them off. The fourth entry on the PHYS:P card is set to flag the use of photonuclear physics. It is read in the first pass through the input file and stored in the variable *ispn*. Any non-zero integer number will indicate that photonuclear physics is to be used during the current transport simulation. Any positive integer indicates natural photonuclear collisions are to be sampled; any negative integer indicates biased photonuclear collisions are to be sampled.

During the first pass, several key parameters are determined about the materials specified. The number of materials specified in the input deck is stored in variable *nmat1* as incremented by routine **newcd1**. The number of isotope/atom-fraction pairs specified on a material card is stored in variable *nwc* as incremented for each entry in the routine **nxtit1**. The number of pairs for the material is then used to update several other variables in the routine **oldcd1**. The total number of pairs seen on all material cards is stored in variable *mix*. The maximum number of pairs for any material is stored in variable *mnnm*. The variable *npn* sets the storage for the array *pan* and is incremented by the number of pairs times the number of cells containing this material. Thermal neutron tables, which contain low energy scattering data to augment neutron tables, are handled by the MT card. The variable *indt* contains the total number of thermal table entries for all MT cards as updated by routine **oldcd1**.

After this first pass through the input file, the storage requirements are computed in the routines **imcn** and **setdas** for the table headers as well as other dynamically allocated variables. The new coding takes a simple approach to allocating space for table headers. In routine **imcn** just after the call to **pass1** that read the input file the input file for the first time, do the following. Count the number of particles that are to be transported in the problem assuming one table set is needed per particle. If photons are transported and electrons are not, increment the count because an electron table set is needed for the thick-target bremsstrahlung routines. If photons are being transported and photonuclear physics is on, increment the count to indicate that two sets of tables are needed for photons. Remember that the variable *mix* contains the total number of isotope/atom-fraction pairs for all materials. The maximum number of tables needed for the problem can be computed by multiplying *mix* times the table sets needed and then adding *indt* to account for thermal tables. This value is stored in variable *mxe1* which is then used in routine **setdas** to allocate storage for table headers.

The set of routines described above has simplified the original coding. It assumes that every isotope/atom-fraction pair will need a set of tables and every thermal table requested is different. The original logic in these routines attempted to account for tables that were used by more than one material and remove the storage allocated for duplicates. This represents a small memory savings in comparison to the amount of coding and work needed. It therefore has been eliminated from the current coding.

Once the routine **setdas** has allocated the dynamic memory, variables are initialized as necessary and the second pass of the input file is made. The arrays *lxd* and *lxn* are initialized to the default table type for each particle for each material (' ' for neutrons, 'p' for photoatomic, 'e' for electrons and 'u' for photonuclear). The array *pnt* is initialized to parameter *huge*, the largest real value allowed by the system. All other arrays of interest to materials are initialized with zero values.

The second pass though the input file checks and stores the remaining user input. In the routine **newcrd**, material cards are checked to ensure they are used by either a cell or a tally multiplier. The M card is ignored otherwise and a warning is printed. The number of materials after discarding those not used is stored in variable *nmat*. The name, i.e. the number from the M card deck, of each material is stored in the array *nmt* in the order they are seen.

The routine **chekit** examines each item on the material card before it is passed to the routine **nextit** to be stored. User input default library specifiers are examined to ensure that the table type is suitable and that the corresponding particle is being transported. Material constituent fractions are checked to ensure they are non-zero and either all atom or all weight fractions. Warning or fatal error messages are printed as appropriate.

The routine **nextit** actually stores the user input values from the input deck in the correct memory location. The ZAID entries are split apart. The ZA is stored in arrays *iza* and *izn*. The MPN card, which must follow the corresponding M card, can then overwrite the ZA value in the array *izn*. The material constituent fractions are stored in array *fme*. Positive values are atom fractions. Negative values are weight fractions that are changed to atom fractions in the routine **rhoden**. A user input default material library specifiers is stored appropriately in either array *lxd* or *lxn*.

The routine **oldcrd** then makes final error checks and completes the storage of material information. If any ZAID entry does not have a corresponding fraction, a fatal error is issued. Otherwise, the number of pairs for the material is stored in array *npq* and locators for the material entries are stored in array *jmd*. Warnings are printed to remind the user if the photonuclear isotope override has been used.

The routine **stuff** determines the actual cross-section tables to be loaded. The array *ixl* contains a coded list of all cross-section tables to be loaded. The first section of the routine **stuff** adds the neutron, photoatomic and electron tables requested to this list. The array *lme* is updated with the table index into array *ixl* to associate each table for each particle type with the appropriate material constituent. A new section of code adds the photonuclear tables requested and performs a similar update for the array *lmn*. The thermal tables are also added to array *ixl* and array *lmt* is used to associate them with the appropriate material.

The order of precedence for the table ID is determined by the algorithms in routine **stuff**. The exact ZAID is requested if specified in the material entry. The ZA and default ID are requested otherwise. The default ID can be from a material option in which case it can include a library number. Otherwise, the request is for the first table of the appropriate type in the XSDIR file.

The list of requested tables in array *ixl* is then sorted and checked for duplicates. The list is sorted by default table order and then alpha-numerically by ZA to facilitate finding and loading the tables. Duplicate entries are removed from the list. Warnings are issued to the user for near duplicate entries, e.g. ZAIDs 29063.22c and 29063.60c. The table index arrays *lme*, *lmn* and *lmt* are updated to maintain correspondence to the appropriate material entry.

The array *ixl* is passed to routine **ixsdir** that determines the tables available for use. The available cross-section tables are listed either on XS cards in the input deck or in the default XSDIR file. The XS card or XSDIR entry provides basic information about the cross-section table including its location in the computer file system. The ZAID entries are checked first against the XS cards and then against the XSDIR entries. The first near match, i.e. correct ZA and table type, is kept in case an exact match is not found. The information from either the near match or the exact match, if found, is stored in the array *ixc*. Near matches are not used for fully specified ZAIDs. For example, if only 29063.60c was found and but ZAID 29063.22c was requested, the near match would not be used.

All array *ixl* entries should now be matched with array *ixc* table location information. As the process may have introduced duplicate entries for matches to partially unspecified tables, they are removed. If any isotope is missing a needed table, the simulation is stopped and an error message is printed. The transport process cannot be run if any cross-section table is missing. Remember that the photonuclear isotope override may request no table be used in which case no table is needed. This completes the input file processing.

The cross-section tables are loaded by the routine **xact** and its subroutines. All cross-section tables are loaded, processed and stored individually except for electron tables. Electron tables are a special case. The electron data tables are loaded last and processed all at once. The new photonuclear tables are handled individually just as all other normal tables.

The routine **getxst** is called to process each individual table. After finding the location of the next table in the appropriate library file, it calls the routine **sread** to read the data into memory. The routine **sread** first checks to ensure that the table header matches the table requested. It then stores the NXS and JXS entries in the corresponding array and all other entries in the *xss* array. Back in the routine **getxst**, the *jxs* locator values are updated to become indices into the runtime *xss* array. If appropriate, the IXS entries are then extracted from the *xss* array and stored in the appropriate *ixs* array. The locators in *ixs* are updated in a manner similar to *jxs*. If appropriate, data that are not needed in the current transport simulation are expunged from the *xss* array again appropriate updating all locators.

To this point only one data table has been stored in the *xss* array at any time. During the transport process all data tables are stored consecutively in the *xss* array. Therefore, all locators are updated one more time to point to where the data will be during the simulation. The table is then written to the file “runtpe” and the next table is processed. When all tables have been processed, the array *xss*, now with all the tables stored in order, is loaded back into memory from the file runtpe. This completes the setup and storage phase.

It is worth noting here that this coding has been subject to extensive review in addition to what was needed for this work. The MCNPX code has recently been updated with the capability to load evaluated proton data to enable tabular sampling of nuclear events. The coding to do this corresponds to the description above except without an isotope override card. This coding has been implemented in MCNPX and was reviewed again at that time. The process was documented [55] and the coding has been in use in MCNPX with no bugs reported to date.

Physics Implementation

In the tabular data regime, MCNP(X) implements a statistical Monte Carlo method to simulate radiation transport. This means that the details of any one history do not necessarily represent physical reality. It is only when the details of many histories are accumulated and considered as a set that average values corresponding to physically meaningful quantities can be determined. This has a significant impact on the data needed to perform the simulation as well as on how the simulation is conducted.

The ACE tabular data include reaction cross sections and the average emission parameters of the secondary particles. For most evaluated data, the word average implies that if a reaction produces two neutrons from a (x,2n) reaction, the emission energy spectrum is the average considering both neutrons together, not separately. In a real collision, the amount of energy the first neutron takes away determines the energy available for the emission of the second neutron. In a statistical sampling process, one averaged emission spectrum is used to sample both neutrons. It is therefore possible to sample reactions in which energy is not conserved for the collision. However, given that enough collisions are sampled, the average emission parameters for the secondary particles are correct.

Statistically average data requires considerably less memory than does true sampling data. As current statistical tables require as much as two megawords of storage per table, routinely using complete data is prohibitively expensive. A complete table would need to include appropriate distributions for every second, third, etc. emission particle and would exponentially increase the storage requirements.

The algorithm for statistical Monte Carlo sampling is simple and straight-forward. During the transport process, the distance to the next event is used along with the direction of flight to move particles through a simulation geometry. If the particle is in a material, the distance-to-collision is one of the possible next events. The distance-to-collision is sampled using a random number and the probability of the particle colliding with an atom in the material. The probability of collision is known as the total macroscopic cross section and is typically given in units of inverse centimeters. The macroscopic cross section is the atom density times the total microscopic cross section for all reactions involving an incident particle type in a given material. The microscopic cross sections are tabulated as a function of the incident particle energy in the ACE tables.

As the routine **hstory** tracks a photon through a medium, it first calls the routine **photot** to compute the total microscopic cross section for the current energy in the current material. Previous to this work, the routine **photot** returned only the total photoatomic cross section. The value of the total photoatomic cross section is stored in variable *totm*. The logic to compute the photoatomic total cross section is left untouched. However, a new test in routine **photot** checks to see if photonuclear physics is on and if so calls the routine **pnctot**.

The routine **pnctot** has been added to compute the total photonuclear microscopic cross section. It accumulates the total photonuclear cross-section in the variable *totpn*. This variable is initialized to zero upon entering the routine. If the incident energy is below the photonuclear threshold, as stored in array *pnt* by material, the routine is done. Otherwise, the total cross section for each isotope in the material is accumulated in *totpn*. Additionally, as each cross section is located, the current energy, the index and offset of the current energy in the main energy grid and the value of the cross section for the isotope are stored in the array *ktc* and *rtc*. This is done such that these values are available immediately if the next request matches the current energy.

Once back in routine photot, the total photonuclear cross section for the material is added to the total photoatomic cross section and stored in *totm* just before returning to routine **hstory**. The routine **hstory** uses this value to compute the total macroscopic cross section, variable *rho*, the atom density, times variable *totm*. The inverse of this value is known as the mean free path and stored in variable *gs*. The distance-to-collision, stored in variable *pmf*, is then computed from the well known Monte Carlo sampling formula. (For general information on the Monte Carlo method for simulating radiation transport, see the reference by Carter and Cashwell [56].) This is an important step in achieving a more correct photon simulation as the use of only the photoatomic cross section can overestimate the distance-to-collision by up to seven percent for photon energies in the giant dipole resonance region and therefore skew the photon collision distribution.

If the next event is a photon collision, the routine **colidp** is called to handle the interaction. Similar to the treatment in routine **photot**, photonuclear interactions are treated in a separate subroutine. If photonuclear physics is on and the photon is above the photonuclear threshold energy, the routine **coldpn** is called at the beginning of **colidp** to handle a possible photonuclear event.

Photonuclear events are rare in comparison to photoatomic events. For this reason, it is useful to have a method to bias the sampling. The variable *ispn*, set by the user as the fourth entry on the PHYS:P card, controls the biasing method. If the value is a positive integer, photonuclear events occur at their natural frequency. That is, sample a random number from zero to one and if it is less than the ratio of the total photonuclear cross section to the total cross section, the collision is a photonuclear event. If not, account for not sampling a photonuclear event, return to the routine **colidp** and sample a photoatomic event in the normal manner.

Biasing forces a photonuclear collision. If the user has set the variable *ispn* to a negative integer, the photon is split. Two particles each of reduced weight (a measure of their importance) are created. One is forced to undergo photonuclear absorption and the other is passed back to the routine **colidp** for normal photoatomic sampling. The weight is adjusted by the probability of each type of event, photonuclear or photoatomic. Specifically, the weight of the photon that undergoes forced photonuclear sampling is scaled by the ratio of the photonuclear cross section to the total cross section. The weight of the photon that undergoes photoatomic sampling is the original photon weight minus the portion that underwent photonuclear absorption.

For either natural or biased photonuclear collisions, it is necessary to update the summary information. If it is a natural sampled photonuclear collision, the summary arrays *pax* and *pwb* are updated to indicate the photon was terminated by a photonuclear absorption. If it is a forced collision, the summaries are updated to account for the weight and energy loss, but do not indicate absorption as the original photon with the remaining weight continues onward.

Once it has been decided that all or part of the incident photon will undergo a photonuclear collision, the target isotope must be chosen. A random number is sampled and a target isotope is chosen based on the ratio of its partial cross section to the total cross section. This is done by accumulating the cumulative probability that the reaction occurred for each isotope in the material in turn until reaching the randomly sampled probability. After the target isotope is chosen, the array *pan* is updated to indicate a collision using that isotope's table.

Based on the target isotope and the incident photon energy, a production cross section can by calculated for each secondary particle of interest. The ratio of the secondary particle-production cross section to the total cross section for that isotope is stored in variable *fp* and represents the average number of particle emissions expected. An integer number of particles suitable for sampling can be obtained by adding a random number from zero to one to the average value *fp* and taking the integer, floor, value. This number is stored in variable *np*. Again, realize that because this is statistical Monte Carlo, the average number of particles emitted is preserved only over large numbers of histories.

Because biased photonuclear particle production can cause considerable variations in weight, it is desirable to have a method to control the emission particle's weight. The weight windows present a reasonable method for achieving this. Weight windows are user input values that control the value of particle weight in an energy region in a spatial region. If the particle is above the weight-window limit, the weight is reduced by splitting it among several identical particles such that the new particles fit in the window. If the particle is below the weight-window limit, Russian roulette is played and particles which survive have their weight increased to a value within the window. The splitting or roulette of particle is limited to a reasonable value for any given step.

If only one energy group exists for the particle of interest at its current position, the weight window control can be applied before sampling the emission parameters. Specifically, the *np* particles to be sampled are split or rouletted appropriately. This is advantageous for two reasons. First, it prevents sampling particles that do not undergo further transported, thus saving CPU time. Second, it splits particles of high weight before sampling emission parameters. This generally provides better statistics faster as more of the emission phase-space is sampled per collision. Obviously, if more than one energy group exists for the weight window, the scheme can only be applied after the particle is sampled and its energy is known. This is done as described below.

A loop over the integer number of particles to be emitted is used to select appropriate emission parameters. Similar to the selection of target isotope, a production reaction is randomly sampled from those available. Once the reaction is picked, the energy and scattering angle are sampled using the standard ACE sampling routines and returned in the laboratory system. The current center-of-mass to laboratory transform does not account for the photon momentum. The routine **rotas** then updates the emission particle's direction of flight. Note that this is a statistical method that randomly samples the reactions producing the emission particles, e.g. if two neutrons are emitted, one may be from a (γ,2n) and the second from a (γ,fission) reaction.

Updates to the ACE sampling routines were needed for four different efforts. The delayed neutron capability implemented by Chris Werner [57] needed to sample neutron emission spectra from a new location in the class 'c' table. The upcoming release of a new ACE continuous-energy neutron library built from the latest release of the ENDF/B-VI evaluated data library will use correlated tabular energy-angle spectra. The coding to implement Energy Law 61, correlated tabular-energy/tabular-angular distribution, and Angular Law 2, tabular angular distributions, were written by Bob Little of X-5 at LANL. The ACE sampling routines have only been used for incident neutron, emitted neutron-photon reactions. Sampling for incident photons and protons with subsequent emission of all light particles required removing certain dependencies within these routines and updating certain algorithms. The effort to use proton tables in MCNPX and the effort to use photonuclear tables in both MCNP and MCNPX required these changes.

In order to avoid multiple implementations, one set of source code was needed implementing the necessary changes. The integration of these changes was coordinated within the scope of this work. The implementation and testing of the updated ACE routines, done in cooperation with Larry Cox of the MCNP development team and Grady Hughes of the MCNPX development team, is documented in two internal memoranda [52,53]. The details of the modifications are left to those documents.

Once the emission particle has been sampled, several updates must be made. The arrays *pax*, *pwb* and *pan* are updated to reflect the particle creation. Particles below their respective energy cutoff are killed. The routines **dxtran** and **tallyd** are called to calculate contributions to dxtran spheres and point detectors. The particles weight is checked against the value for the weight-window, if in a region with energy depend weight windows, and adjusted as described above. Finally, assuming no errors have been encountered, the particle is banked for further transport by the routine **hstory**.

Tallies, Summaries and Other Capabilities

The major MCNP capabilities are all fully functional with the creation of secondary particles from photonuclear interactions as implemented in the present work. Contributions to dxtran spheres and point detectors are calculated as appropriate. Created particles are transported using the standard routines. The weight window scheme is used to control particle weights. The standard MCNP tallies work without change. Only a few auxiliary functions not necessary to transport remain to be integrated.

Summary table information is the last important topic to cover. Biased Monte Carlo can be dangerous if used as a black box. It is possible to force the random walk to skip large areas of importance within a problem such that the answer produced does not reflect physical reality. In order to avoid this, MCNP provides the user a number of summary tables to enable better understanding of the details of the simulation. The example summary tables discussed below are located at the end of the chapter.

All MCNP simulations print a general problem summary. This is a set of tables by particle type that present creation and loss information for external, physical and variance reduction events. Photonuclear adds three new events to the standard MCNP tables: photonuclear absorption of photons, creation of photophotons and creation of photoneutrons. MCNPX will include similar entries for photoproduction of other light particles. Examples of the expanded summary tables for neutrons and photons are shown in Table 3-6 and Table 3-7, respectively.

Implementing the new entries for the problem summary tables was straight-forward. The array *pax* is used to hold the values for these tables. It has been expanded and updated as described above. The routine **sumary** actually prints the table to the output file. New headers were added to this routine for the new table entries. The tables are printed via a loop over each set of entries for each particle type. This loop was appropriately updated to include printing of the new entries. The auxiliary information such as average-time-to-event has also been updated.

Often more insight into the problem is needed than the general problem summary provides. There are three optional print tables that provide more detailed information. This information is provided by cell in each case. This allows the expert user to understand the flow of events within the simulation and hopefully determine if an area or event is under sampled or inappropriately biased.

Table 3-6. Example problem summary table for neutrons.

```
neutron creation    tracks      weight        energy            neutron loss        tracks      weight        energy
                               (per source particle)                                           (per source particle)

source                   0    0.            0.                  escape               20394    3.8037E-05    6.9346E-05
                                                                energy cutoff            0    0.            0.
                                                                time cutoff              0    0.            0.
weight window            0    0.            0.                  weight window            0    0.            0.
cell importance          0    0.            0.                  cell importance          0    0.            0.
weight cutoff            0    0.            0.                  weight cutoff            0    0.            0.
energy importance        0    0.            0.                  energy importance        0    0.            0.
dxtran                   0    0.            0.                  dxtran                   0    0.            0.
forced collisions        0    0.            0.                  forced collisions        0    0.            0.
exp. transform           0    0.            0.                  exp. transform           0    0.            0.
upscattering             0    0.            0.                  downscattering           0    0.            1.0571E-05
                                                                capture                 11    3.6632E-08    1.6110E-07
(n,xn)                   0    0.            0.                  loss to (n,xn)           0    0.            0.
fission                  0    0.            0.                  loss to fission          0    0.            0.
(gamma,xn)           20405    3.8073E-05    8.0078E-05
    total            20405    3.8073E-05    8.0078E-05              total            20405    3.8073E-05    8.0078E-05

  number of neutrons banked                        20405       average time of (shakes)              cutoffs
  neutron tracks per source particle      8.1620E-03             escape            5.3788E-01          tco   1.0000E+34
  neutron collisions per source particle 1.2217E-02              capture           2.5072E-01          eco   0.0000E+00
  total neutron collisions                          30543        capture or escape 5.3760E-01          wc1   0.0000E+00
  net multiplication              0.0000E+00 0.0000              any termination   5.3760E-01          wc2   0.0000E+00
```

Table 3-7. Example problem summary table for photons.

```
 photon creation     tracks      weight        energy            photon loss         tracks      weight        energy
                                (per source particle)                                           (per source particle)

 source                   0    0.            0.                  escape              245286    5.6053E-02    6.5491E-01
                                                                 energy cutoff       306851    1.9838E-02    5.0886E-02
                                                                 time cutoff              0    0.            0.
 weight window            0    0.            0.                  weight window            0    0.            0.
 cell importance          0    0.            0.                  cell importance          0    0.            0.
 weight cutoff            0    0.            0.                  weight cutoff            0    0.            0.
 energy importance        0    0.            0.                  energy importance        0    0.            0.
 dxtran                   0    0.            0.                  dxtran                   0    0.            0.
 forced collisions   233325    0.            0.                  forced collisions        0    0.            0.
 exp. transform           0    0.            0.                  exp. transform           0    0.            0.
 from neutrons            0    0.            0.                  compton scatter          0    0.            1.7336E-01
 bremsstrahlung      226688    9.0549E-02    1.0609E+00          capture                  2    3.2140E-07    3.4372E-06
 p-annihilation           0    0.            0.                  pair production      94574    1.4754E-02    1.8006E-01
 electron x-rays          0    0.            0.
 1st fluorescence         0    0.            0.
 2nd fluorescence         0    0.            0.
 (gamma,xgamma)      186700    2.2239E-04    3.7871E-04          loss to pn. abs.         0    1.2621E-04    2.0788E-03
     total           646713    9.0771E-02    1.0613E+00              total           646713    9.0771E-02    1.0613E+00

   number of photons banked                      646713       average time of (shakes)              cutoffs
   photon tracks per source particle         2.5869E-01         escape              3.0983E-02          tco   1.0000E+34
   photon collisions per source particle  9.4855E-02         capture             1.7548E-02          eco   8.2721E+00
   total photon collisions                       237138         capture or escape 3.0983E-02          wc1   0.0000E+00
                                                                any termination   2.5211E-02          wc2   0.0000E+00
```

Print Table 126 provides general cell activity by particle. It contains information about the population of particles and their average weight, energy and mean free path. No changes were necessary to the methodology for this print table. The values are updated using information that is consistent with the new photonuclear capability.

Print Table 130 provides a detailed weight balance for each cell by particle type. It is split into three parts: external, variance-reduction and physical events. The original version of this table printed with events listed horizontally across the page and cells listed vertically down the page. This completely filled the available 132-column width and could not be expanded to include photonuclear events without exceeding the column limitation. Therefore, when the new photonuclear events were added, the format was rotated such that events are listed vertically down the page and cells are listed horizontally across the page. A maximum of nine cells are printed across the page before the process is repeated. A sub-total for each event type is included as well as a total over all cells. Table 3-8 and Table 3-9 show examples of the new format for the neutron and photon weight balance tables, respectively.

The change in format for Print Table 130 required that the print sequence be revised to print the information rotated as described above. During the revision, the coding was encapsulated in its own routine, **tbl130**, called as needed by routine **action**. It is implemented as a simple series of print statements. For each event, it first prints the header and then the individual cell values one at a time across the page. The 132-column limit corresponds to a maximum of nine cells across the page. New pages are generated as needed and headers are printed each time. The dimension for storage array *pwb* has been incremented by one to provide storage for the calculation of the total over all cells.

Table 3-8. Example page from the neutron weight balance table (Print Table 130).

```
1neutron  weight balance in each cell                                                                    print table 130

      cell index        1           2           3           4           5           6           7           8           9
     cell number       35          40          41          42         401         402         403         404         405

 external events
        entering  2.2513E-04  6.7504E-05  2.2992E-05  2.7263E-04  1.4666E-03  1.4397E-03  2.1264E-03  1.3038E-03  1.4402E-03
          source  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   energy cutoff  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     time cutoff  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
         exiting -2.2511E-04 -1.0820E-04 -2.2992E-05 -2.7262E-04 -1.4666E-03 -1.5762E-03 -2.1262E-03 -1.3038E-03 -2.0719E-03
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal    1.8482E-08 -4.0691E-05  0.0000E+00  5.6769E-09  0.0000E+00 -1.3655E-04  1.6550E-07  7.8432E-09 -6.3166E-04

 var.red. events
   weight window  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
       cell imp.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   weight cutoff  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     energy imp.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
          dxtran  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    forced coll.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     exp. trans.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00

 physical events
          (n,xn)  0.0000E+00  1.8420E-08  0.0000E+00  0.0000E+00  0.0000E+00  8.1360E-09  0.0000E+00  0.0000E+00  7.2428E-08
         fission  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
         capture -1.8482E-08 -3.9059E-06  0.0000E+00 -5.6769E-09  0.0000E+00 -9.8012E-07 -1.6550E-07 -7.8432E-09 -1.0274E-05
   loss to (n,xn)  0.0000E+00 -9.2100E-09  0.0000E+00  0.0000E+00  0.0000E+00 -4.0680E-09  0.0000E+00  0.0000E+00 -3.6214E-08
 loss to fission  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
      (gamma,xn)  0.0000E+00  4.4588E-05  0.0000E+00  0.0000E+00  0.0000E+00  1.3752E-04  0.0000E+00  0.0000E+00  6.4190E-04
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal   -1.8482E-08  4.0691E-05  0.0000E+00 -5.6769E-09  0.0000E+00  1.3655E-04 -1.6550E-07 -7.8432E-09  6.3166E-04

                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
         total    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
```

Table 3-9. Example page from the photon weight balance table (Print Table 130).

```
1photon   weight balance in each cell                                                                      print table 130

      cell index       1           2           3           4           5           6           7           8           9
     cell number      35          40          41          42         401         402         403         404         405

 external events
        entering  1.6142E-03  4.6749E-03  1.6677E-03  1.2534E-06  6.7342E-03  2.4362E-03  2.2120E-01  1.6007E-01  5.6669E-02
          source  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   energy cutoff -1.3090E-05 -1.3067E-03 -6.2040E-08  0.0000E+00  0.0000E+00 -3.9943E-03 -8.3939E-05 -3.0400E-06 -3.0342E-02
     time cutoff  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
         exiting -1.6016E-03 -6.8340E-04 -1.6678E-03 -1.2534E-06 -6.7342E-03 -2.2175E-01 -2.2140E-01 -1.6007E-01 -1.0477E-02
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal   -4.9792E-07  2.6848E-03 -1.2408E-07  0.0000E+00  0.0000E+00 -2.2331E-01 -2.7929E-04 -5.8938E-06  1.5850E-02

 var.red. events
   weight window  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
       cell imp.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
   weight cutoff  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     energy imp.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
          dxtran  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    forced coll.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     exp. trans.  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00

 physical events
   from neutrons  0.0000E+00  8.0097E-08  0.0000E+00  0.0000E+00  0.0000E+00  3.2331E-08  0.0000E+00  0.0000E+00  1.1550E-06
  bremsstrahlung  4.0946E-06  6.3284E-04  1.2408E-07  0.0000E+00  0.0000E+00  2.3342E-01  3.0498E-04  6.3281E-06  6.0550E-02
  p-annihilation  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
 electron x-rays  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
    flourescence  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
     pe. capture  0.0000E+00 -4.3389E-05  0.0000E+00  0.0000E+00  0.0000E+00 -1.3489E-04  0.0000E+00  0.0000E+00 -1.5582E-03
 pair production -3.5967E-06 -3.3323E-03  0.0000E+00  0.0000E+00  0.0000E+00 -1.0162E-02 -2.5684E-05 -4.3428E-07 -7.5133E-02
  pn. absorbtion  0.0000E+00 -3.8444E-05  0.0000E+00  0.0000E+00  0.0000E+00 -1.1947E-04  0.0000E+00  0.0000E+00 -6.2929E-04
  (gamma,xgamma)  0.0000E+00  9.6437E-05  0.0000E+00  0.0000E+00  0.0000E+00  3.0068E-04  0.0000E+00  0.0000E+00  9.1914E-04
                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
      subtotal    4.9792E-07 -2.6848E-03  1.2408E-07  0.0000E+00  0.0000E+00  2.2331E-01  2.7929E-04  5.8938E-06 -1.5850E-02

                  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
         total    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
```

The total over all cells is accumulated as the table is printed and is printed just after the last cell. This process is done for each particle type being transported. It has been verified that the values of the array *pwb* have not been corrupted and that the new print sequence appropriately places the entries.

Print Table 140 provides details of particle interactions by nuclide for each cell. Two tables previously provided information by nuclide and cell for neutron and photoatomic interactions. A new table provides similar information for photonuclear interactions. Table 3-10, Table 3-11 and Table 3-12 show examples of the format for each type of interaction.

For unknown reasons, the production of photons due to neutron interactions was previously listed in the photoatomic summary table. The photon statistics include the total number of tracks produced as well as the average weight and energy of the photons produced. The new location of these values in the neutron nuclide summary is highlighted in Table 3-10. The removal of this information from the photoatomic summary is shown in Table 3-11.

The information provided in Print Table 140 for collisions sampled using photonuclear tables is similar to that provided for neutron tables. The nuclide and its atom fraction for the cell in question are listed first. The total number of collisions and the average weight per collision are listed next. The number of tracks and the average weight and energy of the secondary particles produced are given for both photophotons and photoneutrons. Totals by nuclide are also included.

The changes necessary to implement the revised Print Table 140 were extensive. The array *pan* was expanded in dimension from two to three to allow for a new table type

Table 3-10. Example page from the neutron activity by nuclide table (Print Table 140).

```
1neutron activity of each nuclide in each cell, per source particle                                                              print table 140

   cell  cell    nuclides      atom        total  collisions   wgt. lost   wgt. gain   wgt. gain       tot p    wgt. of        avg p
  index  name              fraction   collisions    * weight  to capture  by fission  by (n,xn)    produced  p produced       energy

      1    35    7014.60c 7.71E-01         2538  1.7247E-06  1.8482E-08  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 8016.60c 2.20E-01          898  5.9341E-07  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                18000.35c 9.60E-03           21  9.5974E-09  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00

      2    40   74184.60c 1.00E+00       512134  3.7335E-04  3.9059E-06  0.0000E+00  9.2100E-09         202  8.0097E-08  6.0720E+00

      3    41    7014.60c 7.71E-01           11  9.3721E-09  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 8016.60c 2.20E-01            4  2.4485E-09  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                18000.35c 9.60E-03            0  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00

      4    42    7014.60c 7.71E-01          629  4.1720E-07  5.6769E-09  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 8016.60c 2.20E-01          242  1.7373E-07  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                18000.35c 9.60E-03           12  4.7780E-09  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00

      6   402   74184.60c 1.00E+00       175033  1.2015E-04  9.8012E-07  0.0000E+00  4.0680E-09          74  3.2331E-08  6.2259E+00

      7   403    7014.60c 8.20E-01         9473  6.3122E-06  1.6550E-07  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 8016.60c 1.80E-01         2933  1.8493E-06  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00


      …     …       …         …          …         …           …           …           …             …         …           …

        total                          41876433  2.8194E-02  1.7193E-04  0.0000E+00  2.4140E-07        8832  1.0267E-05  6.5069E+00


  total over all cells by nuclide        total  collisions   wgt. lost   wgt. gain   wgt. gain       tot p    wgt. of        avg p
                                    collisions    * weight  to capture  by fission  by (n,xn)    produced  p produced       energy

                 1001.60c               1620572  1.0082E-03  4.1615E-06  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 7014.60c                 14171  9.4608E-06  2.1668E-07  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                 8016.60c               2606175  1.6344E-03  3.0568E-07  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                11023.60c                 84243  5.2891E-05  9.9389E-07  0.0000E+00  0.0000E+00          12  2.4892E-07  6.3981E+00
                13027.60c                 77373  4.9247E-05  9.2334E-07  0.0000E+00  0.0000E+00           8  4.9632E-07  7.3646E+00
                14000.60c                729203  4.6399E-04  6.0115E-06  0.0000E+00  0.0000E+00          21  1.2409E-06  6.9548E+00
                18000.35c                    33  1.4375E-08  0.0000E+00  0.0000E+00  0.0000E+00           0  0.0000E+00  0.0000E+00
                20000.60c                 84712  5.4344E-05  2.0322E-06  0.0000E+00  0.0000E+00          90  1.3265E-06  6.3210E+00
                26056.60c                 44103  2.7849E-05  1.7703E-06  0.0000E+00  0.0000E+00          73  1.5110E-06  7.1853E+00
                28058.60c                 13225  9.1831E-06  5.4031E-08  0.0000E+00  0.0000E+00           1  3.2775E-08  8.6418E+00
                29063.60c                 11865  8.3006E-06  5.3625E-08  0.0000E+00  0.0000E+00           2  3.7590E-08  7.2947E+00
                51000.42c               1069402  6.9361E-04  2.1539E-05  0.0000E+00  0.0000E+00        1084  2.3606E-06  6.2558E+00
                74184.60c              19110650  1.3344E-02  1.3293E-04  0.0000E+00  1.9194E-07        7341  2.9072E-06  6.0824E+00
                82208.60c              16410706  1.0838E-02  9.3757E-07  0.0000E+00  4.9455E-08         200  1.0533E-07  6.4524E+00
```

Table 3-11. Example page from the photoatomic activity by nuclide table (Print Table 140).

```
1photoatomic activity of each nuclide in each cell, per source particle                                     print table 140

   cell  cell   nuclides     atom       total  collisions   wgt. lost
  index  name             fraction  collisions    * weight  to capture

      1    35   7000.02p 7.71E-01         212  1.3152E-05  0.0000E+00
                8000.02p 2.20E-01          85  5.2717E-06  0.0000E+00
               18000.02p 9.60E-03          11  6.8244E-07  0.0000E+00

      2    40  74000.02p 1.00E+00       78346  4.8017E-03  4.3389E-05

      3    41   7000.02p 7.71E-01           1  6.2040E-08  0.0000E+00
                8000.02p 2.20E-01           0  0.0000E+00  0.0000E+00
               18000.02p 9.60E-03           0  0.0000E+00  0.0000E+00

      4    42   7000.02p 7.71E-01           0  0.0000E+00  0.0000E+00
                8000.02p 2.20E-01           0  0.0000E+00  0.0000E+00
               18000.02p 9.60E-03           0  0.0000E+00  0.0000E+00

      6   402  74000.02p 1.00E+00      240672  1.4666E-02  1.3489E-04

      7   403   7000.02p 8.20E-01        1616  9.9386E-05  0.0000E+00
                8000.02p 1.80E-01         410  2.4941E-05  0.0000E+00


      …    …       …        …              …       …           …

        total                         6384769  3.8309E-01  4.3187E-03


  total over all cells by nuclide       total  collisions   wgt. lost
                                   collisions    * weight  to capture

                1000.02p                  653  3.5293E-05  0.0000E+00
                7000.02p                 1981  1.2196E-04  0.0000E+00
                8000.02p                44143  2.3585E-03  0.0000E+00
               11000.02p                 1321  6.9470E-05  0.0000E+00
               13000.02p                 5731  3.2715E-04  0.0000E+00
               14000.02p                34281  1.8295E-03  1.2408E-07
               18000.02p                   11  6.8244E-07  0.0000E+00
               20000.02p                 6146  3.3039E-04  1.2407E-07
               26000.02p                 1500  7.9856E-05  1.2409E-07
               28000.02p                 3742  2.2802E-04  1.2384E-07
               29000.02p                  384  2.1824E-05  0.0000E+00
               51000.02p                36061  2.1981E-03  1.3205E-05
               74000.02p              3873350  2.3013E-01  2.2326E-03
               82000.02p              2375465  1.4536E-01  2.0724E-03
```

Table 3-12. Example page from the photonuclear activity by nuclide table (Print Table 140).

```
1photonuclear activity of each nuclide in each cell, per source particle                                         print table 140

   cell  cell    nuclides      atom        total  collisions        tot p     wgt. of       avg p       tot n     wgt. of       avg n
  index  name               fraction  collisions    * weight     produced  p produced      energy    produced  n produced      energy


      2    40  74184.03u 1.00E+00        53756  3.8444E-05        94815  9.6437E-05  1.1576E+00       57388  4.4588E-05  1.4209E+00


      6   402  74184.03u 1.00E+00       165705  1.1947E-04       291682  3.0068E-04  1.1578E+00      175227  1.3752E-04  1.4152E+00


      9   405  82208.03u 9.34E-01      1069157  6.2929E-04       915777  9.1914E-04  1.6709E+00     1048772  6.4190E-04  2.2029E+00


     10   451  82208.03u 9.34E-01       343129  1.8974E-04       276449  2.6978E-04  1.6433E+00      333256  1.9253E-04  2.1867E+00


     12   406  74184.03u 9.98E-01       624461  3.7915E-04       958159  9.1936E-04  1.1622E+00      617744  4.1598E-04  1.3695E+00
               29063.03u 1.00E-03       435474  5.0690E-08       398280  1.0306E-07  1.4272E+00          20  2.5686E-08  1.5598E+00


     14   407  74184.03u 9.98E-01      1564103  1.0354E-03      2575333  2.5601E-03  1.1564E+00     1603294  1.1685E-03  1.3979E+00
               29063.03u 1.00E-03       934420  1.2750E-07       856207  2.4837E-07  1.1638E+00          77  8.7239E-08  1.6041E+00


        total                          5190205  2.3916E-03      6366702  5.0658E-03  1.2768E+00     3835778  2.6011E-03  1.6517E+00


  total over all cells by nuclide        total  collisions        tot p     wgt. of       avg p       tot n     wgt. of       avg n
                                    collisions    * weight     produced  p produced      energy    produced  n produced      energy

               13027.03u                  3889  1.1872E-07         2848  1.4591E-07  1.6055E+00         321  2.6097E-08  1.5615E+00
               20040.03u                 47339  3.9092E-07        29003  1.4934E-07  3.8396E+00          29  4.1120E-09  7.6706E-01
               26056.03u                  6889  1.4796E-07         5803  1.3887E-07  2.3300E+00        2759  1.0913E-07  1.6223E+00
               29063.03u               1369894  1.7819E-07      1254487  3.5144E-07  1.2410E+00          97  1.1293E-07  1.5940E+00
               74184.03u               2408025  1.5724E-03      3919989  3.8766E-03  1.1579E+00     2453653  1.7666E-03  1.3932E+00
               82208.03u               1412286  8.1903E-04      1192226  1.1889E-03  1.6646E+00     1382028  8.3443E-04  2.1991E+00
```

and from six to eight to allow for additional information. The expansion for additional information allowed the photon production from neutron collision data to be moved back into the neutron table listing. The new listing for photonuclear collisions is updated in the routine **coldpn** to account for the number of collisions and their weight as well as the number, weight and average energy of photons and neutrons produced.

Similar to the routine **tbl130**, a new routine was written for Print Table 140. The routine **action** now calls routine **tbl140** to print the nuclide activity information. Again, a simple, brute-force solution was implemented to print the necessary information. It follows the same format as in the original Print Table 140 though it has been condensed in width to allow for the additional information as described above. The values for the values for these existing tables have been checked to verify they have not been corrupted by the relocation. The new values have been checked by hand.

Future Work

The next step in this effort is to implement the prototype coding into the MCNPX code for release to a beta-user community. It is believed that this is the best audience to begin using the new photonuclear physics capability. Over the last three years, these users have helped guide the evolution of the MCNPX code by testing new capabilities and suggesting future directions. As the last features necessary to implement the photonuclear physics capability are finalized, the assistance of this community should ensure that the final product is bug free and user friendly.

There are a few areas of known concern in the current coding. The LA150 photonuclear data make use of only one reaction cross section with secondary particle-production based on appropriate yields and emission spectra given by Kalbach systematics. The errors and warnings in the prototype have been verified to ensure they would catch problems with the LA150 data but further checks are needed to verify they will catch all generic data errors. Similarly, the sampling routines have been verified to correctly use the LA150 data, but further checks are needed to verify that other processed data will work correctly. Last, MCNP and MCNPX are riddled with hidden dependencies. It is expected that further user testing will produce several minor glitches.

The current sampling of photonuclear data using Energy Law 44 is based on the original formalism by Kalbach [50,51]. Chadwick has proposed [11] that the reduced momentum of the photon particle incident on the heavy target is more realistically sampled as isotropic for multi-step compound decay. A new Energy Law including this modification needs to be proposed to the Cross Section Evaluation Working Group (CSEWG), the group which controls the ENDF format.

MCNP(X) provides many auxiliary functions in addition to its mainstream capabilities. Some of these features, such as event log printing, have updated to include photonuclear information. However, several others, including but are not limited to the tally multiplier card, cross section plotting and ptrac file writing, have not been updated. For this work, these features were not necessary. As time permits, they will be added into future implementations.

CHAPTER 4
VERIFICATION AND VALIDATION

Introduction to Verification and Validation

In today's scientific world, the computer has become an essential tool. However, the use of the computer is still an evolving subject. An entire field of study has devoted itself to software quality assurance. The cornerstones of software quality assurance are verification and validation. As this work is in large part a software coding project, it is desirable to address the question of verification and validation here.

Verification in the context of software quality assurance is the process of ensuring the functionality of the software. It can also be thought of as tackling the subtler issue of the garbage in equals garbage out problem. The issue is how to ensure that the functionality of the coding performs in the manner expected and does so for all valid cases. A brief summary of the verification of the present work is presented in the next section. The larger problem of ensuring that the results are not garbage is the domain of validation and the primary focus of this chapter.

The quest of validation in the context of software quality assurance is to prove that the coding and data in question perform with reliable results in the region of interest. For the purpose of this work, the region of interest is the production of neutrons from bremsstrahlung photons in the energy range up through several tens of MeV. As the ability to use tabular photonuclear data within MCNP has not been generally available before, the validation results discussed here provide the basis for estimating the general uncertainty for the this new capability as a whole.

For the purpose of this work, the ideal validation benchmark should include the production of photons via bremsstrahlung radiation and the subsequent production of neutrons from those photons. Additionally, such a benchmark should contain as few extraneous complications as possible and must document enough of the setup and analysis to conclude that a fair comparison to any simulation is being obtained. The literature has been searched for such benchmarks with minimum results.

There is a dearth of experimental data available in the area of photonuclear physics. Of the published results, few are suitable for use as benchmarks. Many early experimentalists measured the photonuclear cross section of materials with bremsstrahlung photons. The raw data of neutrons observed for a specific experimental configuration would be an ideal benchmark. However, the raw data is typically "unfolded" and represented as a set of cross sections. The experimental data thus presented is unusable as a benchmark as the unfolding method is not documented and typically a poor quality cross-section measurement in comparison to those obtained with mono-energetic photons.

There are numerous other measurements in the literature that would be useful but come from complicated systems that are not well documented. Measurements that are based on poorly documented systems cannot be used as benchmarks because too many assumptions must be made in the simulation model. For a measurement to be useful as a benchmark, it must contain a description of the experimental setup that thoroughly documents every significant parameter.

Two studies have been chosen from the available literature for the purpose of validation of the work presented in this dissertation. Swanson [31,32] folded differential photon fluxes calculated from analytical shower theory with measured cross sections to obtain neutron yields from electrons incident on semi-infinite slab geometry. Barber and George [58] reported absolute measurements of neutrons produced per electron incident on several materials.

The results presented by Swanson are not true experimental measurements. However, they are useful for a number of reasons. The Barber and George experiment was the only work found to date that contains the details necessary to be defined as experimental benchmark data. As a second study was desired for comparison, the work of Swanson was chosen. That this study was chosen was not an arbitrary decision. The original motivation for this work was the assessment of the neutron field in a medical accelerator room. The defining work in this area is NCRP Report 79 [2]. Swanson participated as a consultant to the task force which wrote this report. The method he has used to calculate theoretical neutron yields is recommended in the report to calculate the neutron source produced within an accelerator.

Swanson utilizes analytical shower theory and experimental cross-section measurements to obtain an expected neutron yield released by electron bombardment of a material. The accuracy of the neutron yield calculated is highly sensitive to the accuracy of the cross-section measurements used. However, in the absence of the equivalent direct measurement, this is a useful way to calculate neutron yields.

The Barber and George results are experimental and they are considered an excellent benchmark. As such benchmark data are rare, great pains have been taken here to explain how the data was taken in the hopes that experimentalists who read this might be compelled to perform similar measurements. It is hoped that the availability of this new ability to simulate photonuclear interactions in radiation transport will encourage such benchmark experiments.

Comparison of current calculations to each of the sets of data described are presented as a section of this chapter. In each section, the original study is described in enough detail to understand what must be taken into consideration in the simulation. The setup of the simulation including any assumptions is depicted and the results of the current calculation versus the original results are discuss. For completeness, the actual MCNP input decks are included in Appendix D. Comparisons are included for each material which has a corresponding tabular photonuclear data set. The final section of the chapter will summarize the conclusions that have been drawn from these comparisons and assess the overall uncertainty attributable to the evaluated data and its use by the current coding.

Verification

Anyone who has ever written even the simplest piece of software has had to learn something about verification. Rarely does a piece of software compile and run as it is supposed to on the first attempt. Typically, it is necessary to debug the code to remove errors in implementation. Even if it does compile and appear to run, it is wise to run test cases to ensure that reasonable input conditions give reasonable results. Additionally, there are numerous tasks that can be performed during the development cycle of the software to ensure that it functions as expected.

There were four major changes implemented within the existing MCNP code in order to establish the functionality desired by this work. The verification of each major change is discussed separately. This documentation is meant to provide an overview of the verification that was done without actually showing all the details.

The input routines were revised in order to allow specification of a photonuclear table for use by a material. As discussed in the implementation chapter, this required extensive revisions to the original coding. The final implementation was inspected by several code walkthroughs. (A code walkthrough is a detailed inspection of the code modifications by two or more persons to ensure that revisions in question accurate implement the desired functionality.) In addition, the arrays used to store the tabular data were checked during debugging runs to ensure that the appropriate data was stored correctly.

In addition to these algorithms implementation within MCNP for loading photonuclear tables, they were duplicated within MCNPX for loading proton tables [55]. They were also subjected to a code walkthrough at that time. They have seen active use since their implementation with no bugs reported.

The tabular data sampling routines were revised to correctly sample emission parameters for any combination of incident and emitted particle. These changes been duplicated in MCNPX in order to correctly handle proton tables and in MCNP for miscellaneous other reasons [52,54,59]. There have been subjected to several code walkthroughs at various stages in this work. During their implementation into the distribution version of MCNP, they were subjected to extensive verification and quality assurance testing [53]. They have been in active use by this prototype, MCNPX and MCNP with no bugs reported.

The output tables were updated to include details about the photonuclear interactions in a simulation. These changes have been subjected to a walkthrough. They have also undergone testing via debugging runs to ensure that the numbers reported accurately reflect the experience of the simulation. They have been in use throughout this work with no known problems.

The last major section of code revision is the addition of the photonuclear collision routine. This set of algorithms represents the keystone of this work. Similar to the other revisions, it has been subjected to code walkthroughs and debugging runs. In addition, it has undergone a number of testing runs to ensure that it functions appropriately. These tests generally check a specific feature. For example, a simulation can be contrived to look for the correct attenuation of photons through a material in order to ensure that the appropriate cross section is being used. Another example is the use of a thin target surrounded by tally regions to check appropriate sampling of secondary emission energy and angle. Numerous other tests were run to check various aspects of the functionality [60].

In addition to the testing described above, MCNP has a set of regression tests covering it core functionality. These tests were used throughout the development process to ensure that the new or revised capabilities did not damage existing functionality. Additional regression tests were added to check new functionality.

It is worth noting here that verification of large codes is a complicated process. Inherent interdependencies can be overlooked and left unchecked. Despite the variety of verification performed during the development process, a bug was found in the photonuclear collision routine very late in this process. It only affected the activation simulations discussed in the following chapter but it serves as a reminder that verification is a long-term effort over the life of a software project.

Comparison to Theoretical Yields

Calculating Theoretical Yields

During the late 1970's and early 1980's, there was general interest in developing a methodology for estimating the neutron yields at various research and medical electron accelerators for subsequent use in radiation protection calculations. William Swanson, then at the Stanford Linear Accelerator Facility, documented a methodology and reported neutron yields per electron incident on various materials at selected energies up to the GeV range [31,32]. His work has been used by others to provide guidance on neutron source-term calculations for electron accelerators.

Swanson folded experimental photoneutron cross-section measurements with calculated photon fluxes to calculate "theoretical" neutron yields. In the results shown here, analytical shower theory is used to predict the differential photon flux for an electron of a specified energy incident on a semi-infinite (a half-space geometry) material. Experimental photoneutron production cross sections were obtained either as a piecewise continuous function or as a Lorentz parameterization. These were then integrated together by folding the predicted flux with the macroscopic cross-section to calculate the neutron yield.

It is worth digressing for a moment to state the obvious. The use of analytical shower theory to calculate the differential photon flux can be replaced by the use of

Monte Carlo electron-photon simulation. In fact, Swanson checked some of his analytical flux predictions against the Monte Carlo generated differential photon fluxes of Alsmiller and Moran [20] for a 10 radiation length thick (practically equivalent to semi-infinite) lead target. The use of Monte Carlo to estimate the photon flux for a specific geometry and material provides a better estimate of neutron yield from a real component as spatial dependence can be obtained. However, Swanson was focused on obtaining a general method.

Swanson reported yields for twelve materials in their natural elemental state. There is a corresponding evaluated photonuclear data set available for aluminum, iron, copper, tantalum, tungsten and lead. Each of these elements is listed in Table 4-1 with their radiation length, density and the source of the photoneutron production cross-section data used in calculating the "theoretical" yield as reported by Swanson. (For a definition of the radiation length along with the original source for Swanson's values, see the work by Yung-Su [61].)

It is important to remember that the error in these calculations is a function of the error in the prediction of the photon flux and the error in the cross-section data. Either can have large influences on the results. In Swanson's conclusions within the second study, he states that the values obtained in the present work lie as much as 50 percent below the original calculations in the energy range typical for radiation therapy. He then goes on to say that the uncertainties in his present calculations are less than 20 percent [32, p. 357]. This is meant to show that the change in the approximations used to obtain the photon flux improved the results but it also shows the sensitivity of these calculations to the underlying data and assumptions. However, as his comments point out, given the

Table 4-1. Materials and properties used by Swanson to calculate theoretical neutron yields.

Material	Radiation Length (g/cm^2)	Density (g/cc)	Source of photoneutron cross-section data
Aluminum	24.01	2.699	Veyssiere et al. [62]
Iron	13.84	7.875	Montalbetti et al. [63] Data normalized to agree with the Fe/Cu yield ratio determined by Price et al. [64]
Copper	12.86	8.96	Fultz et al. [65]
Tantalum	6.82	16.6	Bergere et al. [66]
Tungsten	6.76	19.3	Veyssiere et al. [67]
Lead	6.37	11.35	Pb-206,7 from Harvey et al. [68] Data scaled by 1.36 [32, p.348] Pb-208 from Veyssiere et al. [69]

correct geometry and an accurate photon flux the underlying uncertainty in the cross sections still leaves a large uncertainty in the yields.

Simulation Setup

Part of the reason these calculations were used as a benchmark was the simplicity of the geometry involved. The neutron yields are reported on semi-infinite slabs. Swanson refers several times to the fact that the semi-infinite condition is practically achieved at a target thickness of 10 to 20 radiation lengths. With this in mind, the simulations presented here use a mono-energetic, point electron beam incident on a recto-linear slab 20 radiation lengths thick with the beam centered such that it is 20 radiation lengths to each edge. The mono-energetic, point representation of the beam is equivalent to the assumptions in the original work.

The yields as reported are for neutron production only. Neutron transport and absorption within the material is not considered. In order to match these numbers, the value calculated by the simulation is taken from the MCNP neutron creation summary table, i.e. the absolute neutron production per incident electron excluding absorption within the target material. A tally of the neutrons escaping the sample is used to determine when neutron production has converged and also to estimate the uncertainty in the neutron yield.

The yields are reported for naturally occurring, elemental materials. The natural abundance of each material is given in Table 4-2 along with the tabular data used for photonuclear and neutron interactions within the sample. For many materials, only selected isotopes have photonuclear evaluated data files available. If this is the case, the isotope(s) available was (were) used by splitting the missing weight equally among those on hand. The implications of this practice are discussed below.

Neutron tables were chosen to match the available photonuclear tables. All isotopes except tantalum had a corresponding LA150 [40] neutron evaluation covering the energy range of interest up to 150 MeV. The tantalum evaluation used is from the standard ENDF60 library [70] with an upper energy limit of 20 MeV. As the yields are

Table 4-2. Natural isotopic abundance for elemental target materials and their isotopic representation due to lack of available tabular data.

Material	Natural Isotopic Abundance's (atom %)	Photonuclear ZAIDs (Atomic Abundance's)
Aluminum	^{27}Al (100%)	13027.03u (100%)
Iron	^{54}Fe (5.8%), ^{56}Fe (91.72%), ^{57}Fe (2.2%) and ^{58}Fe (0.28%)	26056.03u (100%)
Copper	^{63}Cu (69.17%) and ^{65}Cu (30.83%)	29063.03u (100%)
Tantalum	^{180}Ta (0.012%) and ^{181}Ta (99.988%)	73181.03u (100%)
Tungsten	^{180}W (0.13%), ^{182}W (26.3%), ^{183}W (14.3%), ^{184}W (30.67%) and ^{186}W (28.6%)	74184.03u (100%)
Lead	^{205}Pb (1.4%), ^{206}Pb (24.1%), ^{207}Pb (22.1%) and ^{208}Pb (52.4%)	82206.03u (24.5667%), 82207.03u (22.5667%) and 82208.03u (53.8666%)

quoted neglecting neutron transport and absorption within the target, this does not affect the comparison. Electron tables are from the MCNP standard EL1 library and photoatomic tables from MCPLIB02.

Neutron yields are tabulated for ten incident electron energies in the revised study [32]: 10, 15, 20, 25, 34, 50, 100, 150, 500 and 1000 MeV. Because MCNP4B has an upper energy limit for the current electron tables of 100 MeV, only the first seven energies are used in this comparison. Future plans for the MCNP family of codes include the integration of electron tables covering the energy range up to 100 GeV and the inclusion of the CEM nuclear physics module to handle photonuclear interactions above the range of tabular data. Additionally, the IAEA photonuclear data library will include many of the isotopes currently missing. As these advances are made, this set of calculations should be revisited to complete the verification-validation process.

Comparison to Current Calculations

The comparisons between the theoretically derived data and the MCNP calculations are presented here as a set of graphical figures. The error bars on the theoretically derived values are the 20 percent uncertainty quoted by Swanson. The simulations have been run until the relative error in the calculation is negligible. The overall uncertainty of the simulation method and data is the final goal of this discussion. Each figure presents the experimental data as diamonds connected by a solid line with calculated values represented as squares connected by a dashed line. The graphs are presented on a log-scale to enable viewing of the entire range of data at once. It is desired that the reader achieve only a sense of relative comparison from the figures with explanation of the details given in the text. The MCNP input decks and the reported and calculated values are available in Appendix D.

The comparison for aluminum presented in Figure 4-1 shows very good agreement except for the two lowest energy values. Despite using essentially the same threshold energy, there is a factor of five difference in the value of the yield at 15 MeV. This may be explained by differences in the shape of the photoneutron cross section in its rise to the peak value.

Swanson's values are based on the photoneutron cross section as measured at Saclay [62]. Experimental data from a number of institutes [62,71-73] were used in the evaluation process. However, the evaluated data do not include possible small resonance in the rise region as seen in the data from Saclay. The small resonances are physically realistic and these small changes in this region may have large influences on yield calculations for incident energies near threshold.

Iron was a very difficult evaluation as very little experimental data exist for guidance. Of the four naturally occurring iron isotopes, only ^{56}Fe is currently available as an evaluated data set. Considering only ^{56}Fe is probably reasonable considering it represents more than 90 percent of the elemental composition. However, the current calculations over predict the reported values by 30 to 40 percent as shown in Figure 4-2.

The large difference seen between the reported and calculated values may be explained by differences in the photoneutron cross sections. The reported values are based on the experimental data of Montalbetti et al. [63] as scaled [32, p. 355] to agree with data reported by Price et al. [64]. The evaluation is based on more recent experimental data [73-76]. This once again points out the need for evaluated data to

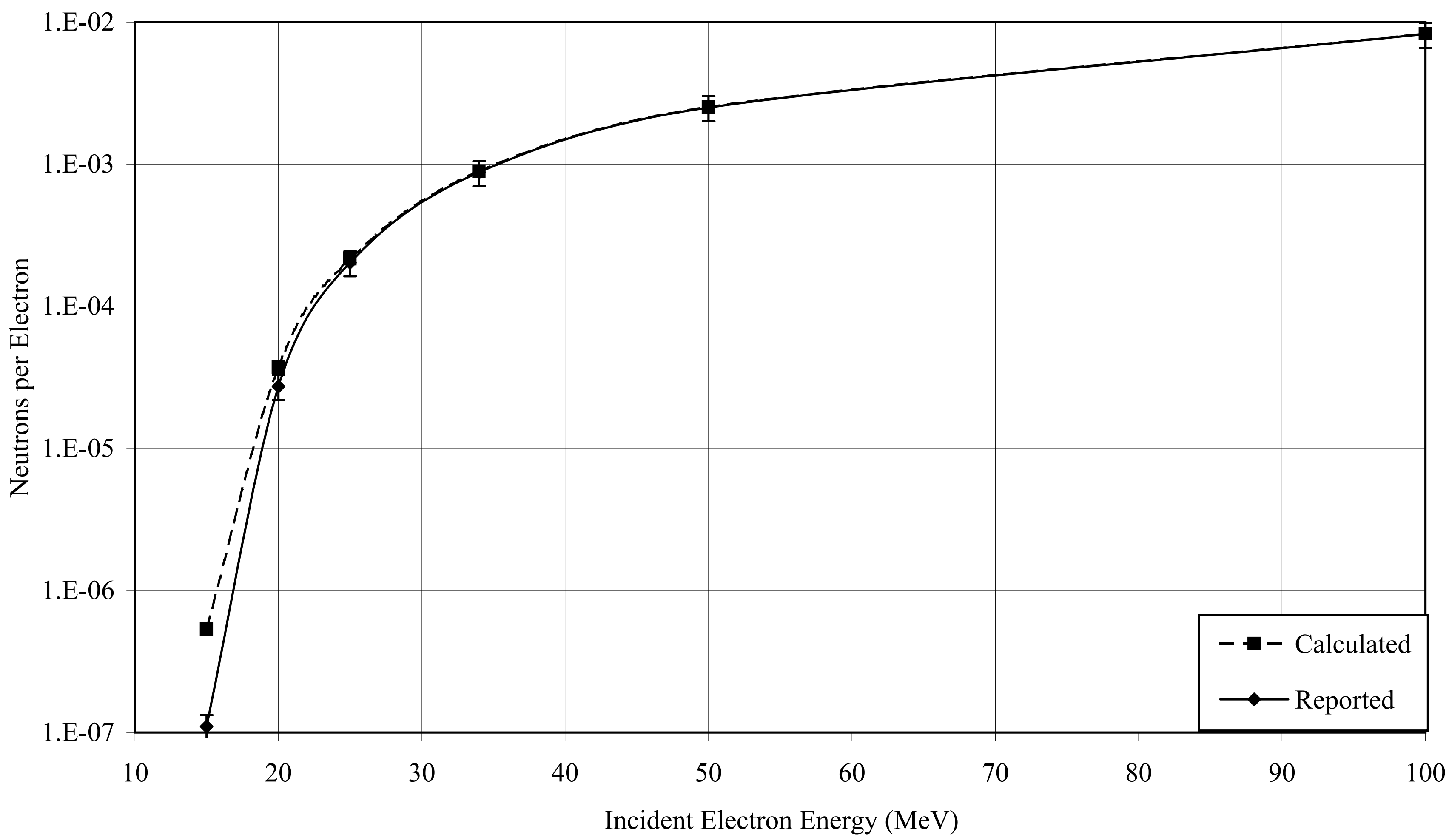

Figure 4-1. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick aluminun target. (Reported values from Swanson, 1979.)

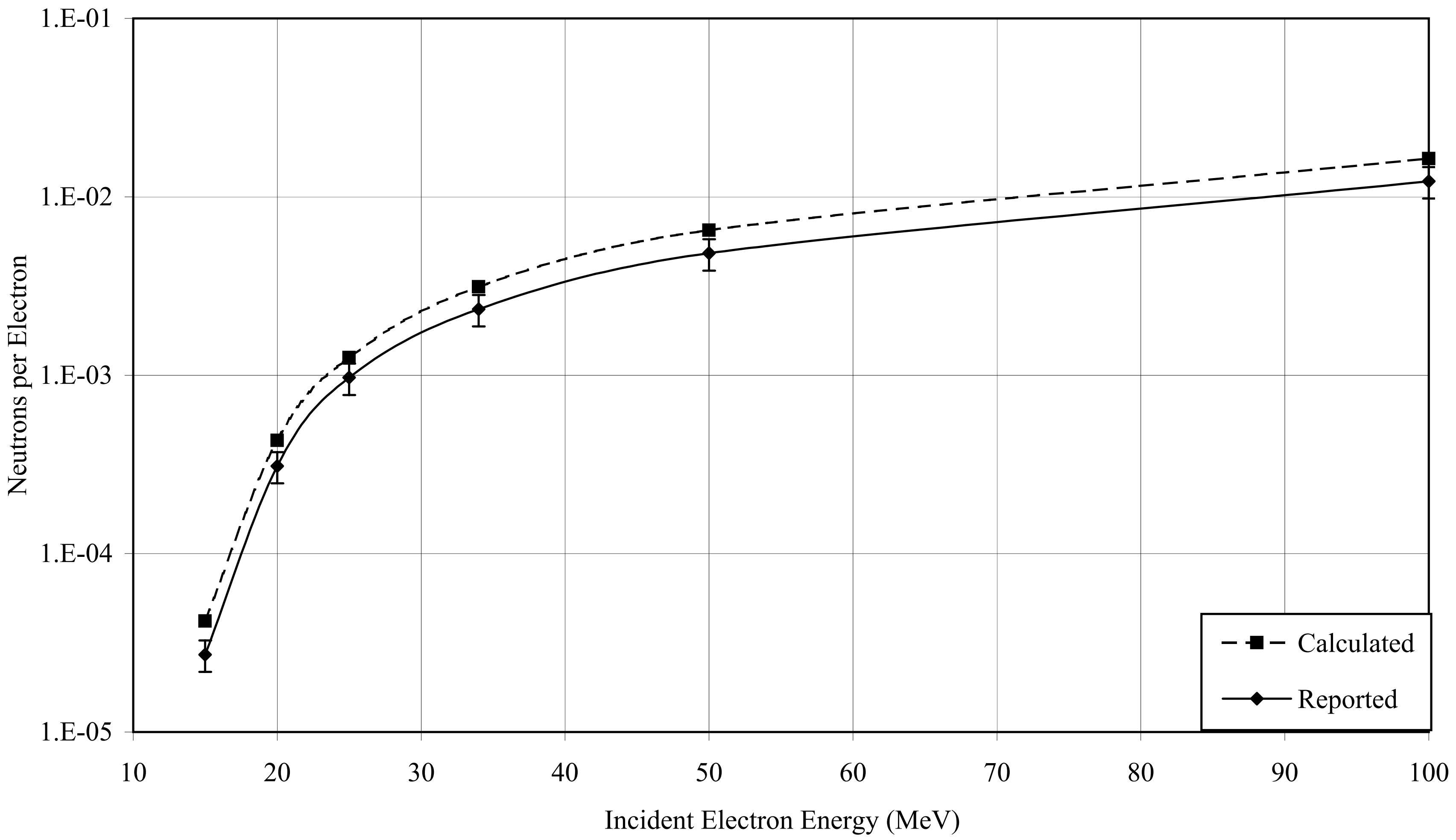

Figure 4-2. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick iron target. (Reported values from Swanson, 1979.)

ensure that accurate representations are provided based on multiple experimental data sets.

Copper shows very good agreement between the current calculations and the reported values of the yields. This is expected since both the reported values and the current work are based on the original measurements by Fultz et al. [65]. However, Figure 4-3 shows an increasing discrepancy between the values as the incident energy increases.

One possible explanation for the discrepancy between the current calculation and the reported value is the lack of an evaluated ^{65}Cu photonuclear data set. One way to estimate the influence of the missing cross-section data is to examine the cross sections for ^{63}Cu versus ^{65}Cu versus ^{Nat}Cu. Table 4-3 shows the values for the integrated photoneutron-yield cross sections. Note that, as expected, the current work is in close agreement to the ^{63}Cu experimental data of Fultz et al. However, it underestimates the photoneutron production from natural copper. As the photon flux increasing, e.g. with increasing incident energy, this underestimation will cause a larger discrepancy.

Table 4-3. Integrated photoneutron yield cross-sections for copper.

Isotope	E_{max} (MeV)	Current Value[a] (mb-MeV)	Reported Value (mb-MeV)	Source of cross-sections for reported value
^{63}Cu	27.8	688	680	Fultz et al. [65]
^{63}Cu	25.1	619	584	Sund et al. [77]
^{65}Cu	27.8	688	817	Fultz et al. [65]
^{Nat}Cu	27.8	688	710	Fultz et al. [65]
^{Nat}Cu	19.6	371	450	Miller et al. [78]

[a] Current values are based only on ^{63}Cu, i.e. they contain no estimation of the influence of ^{65}Cu.

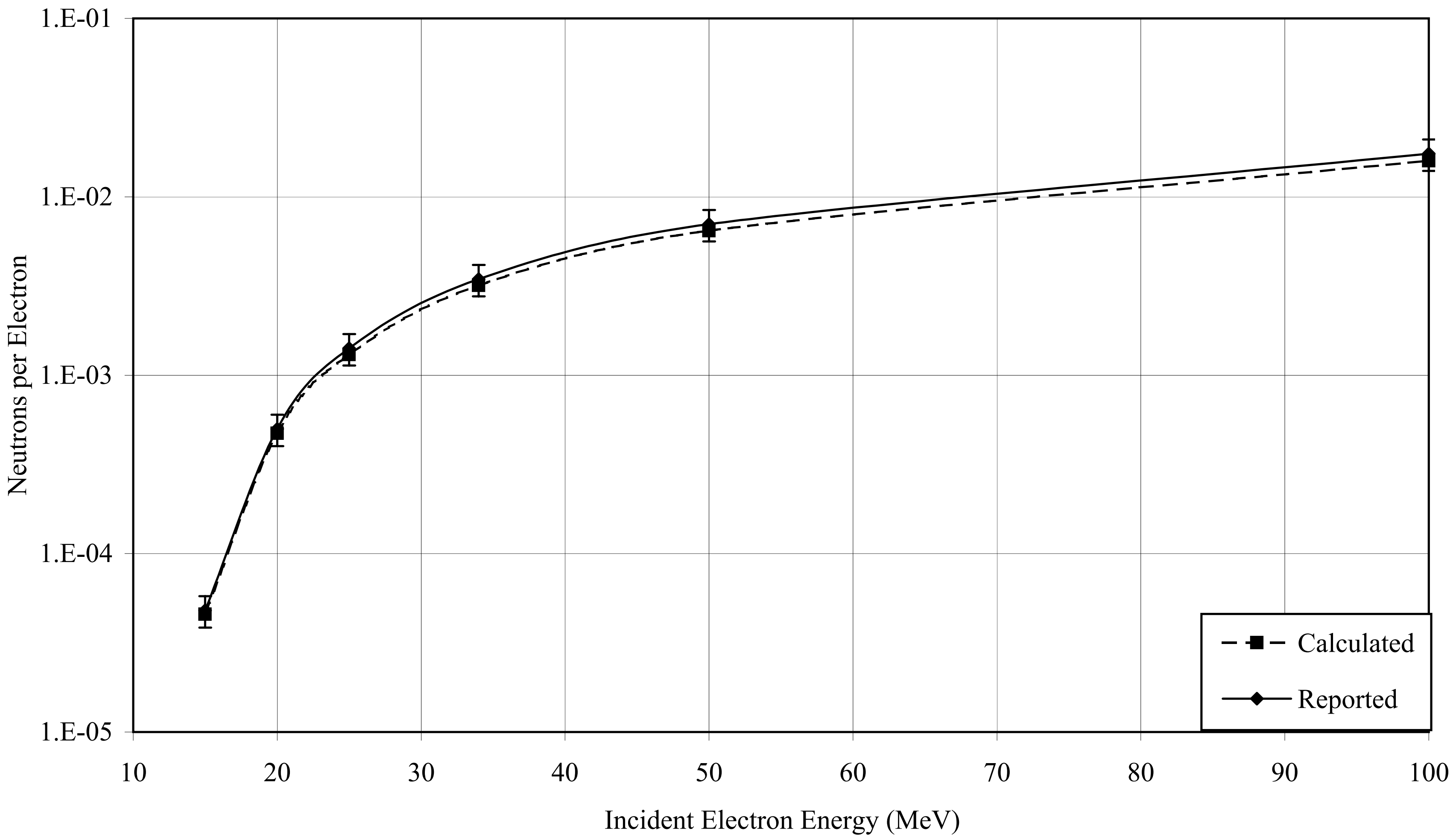

Figure 4-3. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick copper target. (Reported values from Swanson, 1979.)

There is very good agreement, as shown in Figure 4-4, between the calculated and reported yields for tantalum. The slight variations are most likely small differences in (γ,n) and (γ,2n) cross-section shapes. The current work is based on experimental data from Lawrence Livermore [79] scaled by 1.22 as recommended by Lee et al. [80]. The reported yields are based on the experimental data from Saclay [66] which are believed to contain an error in the multiplicity [80] which may account for the slight variations seen.

It is worth examining the three available experimental cross-section measurements, listed in Table 4-4, in order to emphasize a key theme. The small group of individuals who perform nuclear data evaluations have available important information that is not readily apparent to the outsider. It is generally believed that many of the early cross-section measurements at Lawrence Livermore are 15 to 25 percent too low. It is this kind of knowledge that is essential in determining that the higher cross-section values reported in Table 4-4 are more probably correct and the Livermore data should be scaled upwards to match. This need for in-depth knowledge of the experimental data is the reason evaluated data exists and why it is important to the novice to trust the judgement of those who produce this evaluated data.

Table 4-4. Integrated photoneutron yield cross-sections for tantalum.

Isotope	E_{max} (MeV)	Current Value (mb-MeV)	Reported Value (mb-MeV)	Source of cross sections for reported value
^{181}Ta	22.0	3623	2970	Miller et al. [78]
^{181}Ta	24.6	3879	3735	Bramblett et al. [79] Scaled by 1.22 [80]
^{181}Ta	25.2	3929	3799	Bergere et al. [66]

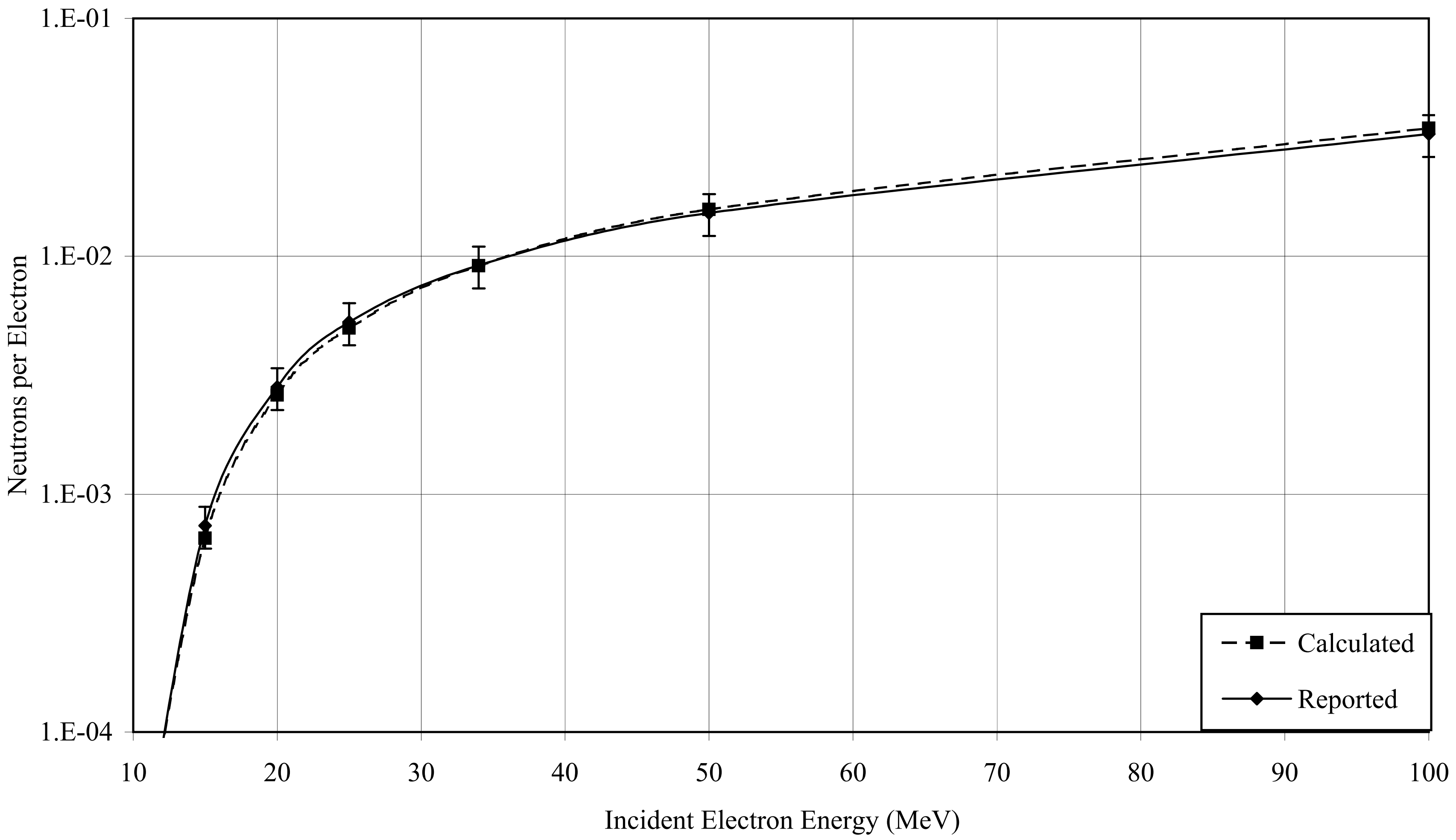

Figure 4-4. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick tantalum target. (Reported values from Swanson, 1979.)

The calculated yields under predict the reported yields for tungsten at all but the highest energies shown in Figure 4-5. The difference is largest near threshold. Swanson's reported values are based on natural tungsten with a threshold of 6.2 MeV. The current evaluated data is only isotopic ^{184}W and has a threshold of 7.5 MeV. Both works are derived from the original measurements of Veyssiere et al. [67]. It is hypothesized that if the remaining isotopes, comprising 70 atomic percent, of tungsten were included the agreement would be closer.

The cross sections for lead have been reanalyzed since the Swanson study was concluded. Based on an analysis performed by Berman et al. [81], the Livermore measurements of Harvey et al. [68] are believed to be too low by a factor of 1.22, not 1.36 as was proposed by Fuller [32, p. 385]. Berman et al. also suggested the Saclay measurement [69] of ^{208}Pb was too high and should be scaled down by a factor of 0.93. Keeping these differences in mind, Figure 4-6 shows acceptable agreement between the calculated and reported values for lead.

Comparison to Measured Yields

Experimental Setup

In an experiment by Barber and George [58], an electron beam was impinged on a variety of targets at various energies and the absolute yield of neutrons per electron was measured. The targets were composed of elemental materials of varying thickness including Al, Cu, Ta and Pb. It is a testament to the diligence in making and reporting these measurements that they have been cited within the literature more than 25 times, including Swanson in his study described above. Their results are worthy of use as benchmarks and ideal for benchmarking the current methodology and data.

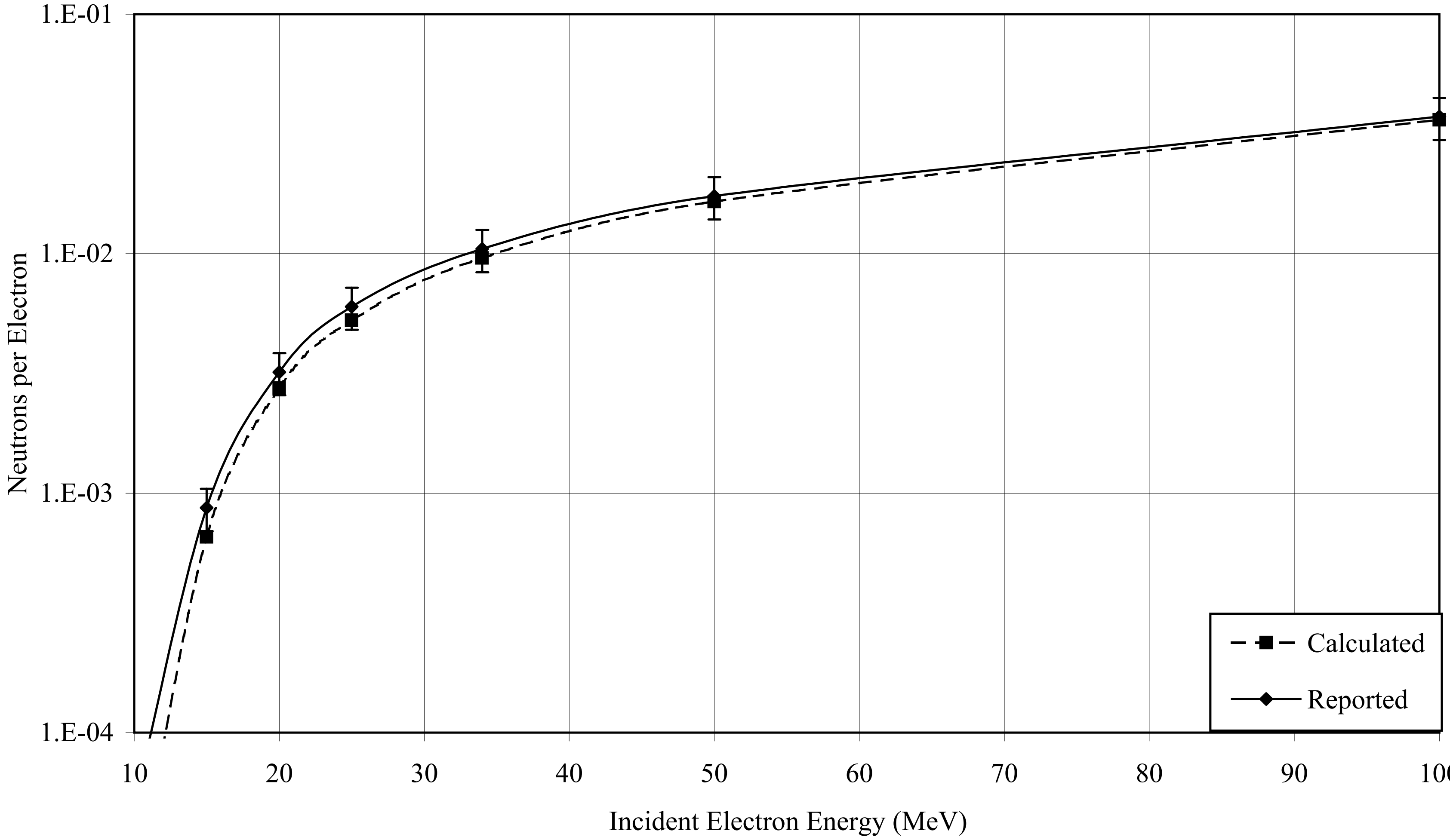

Figure 4-5. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick tungsten target. (Reported values from Swanson, 1979.)

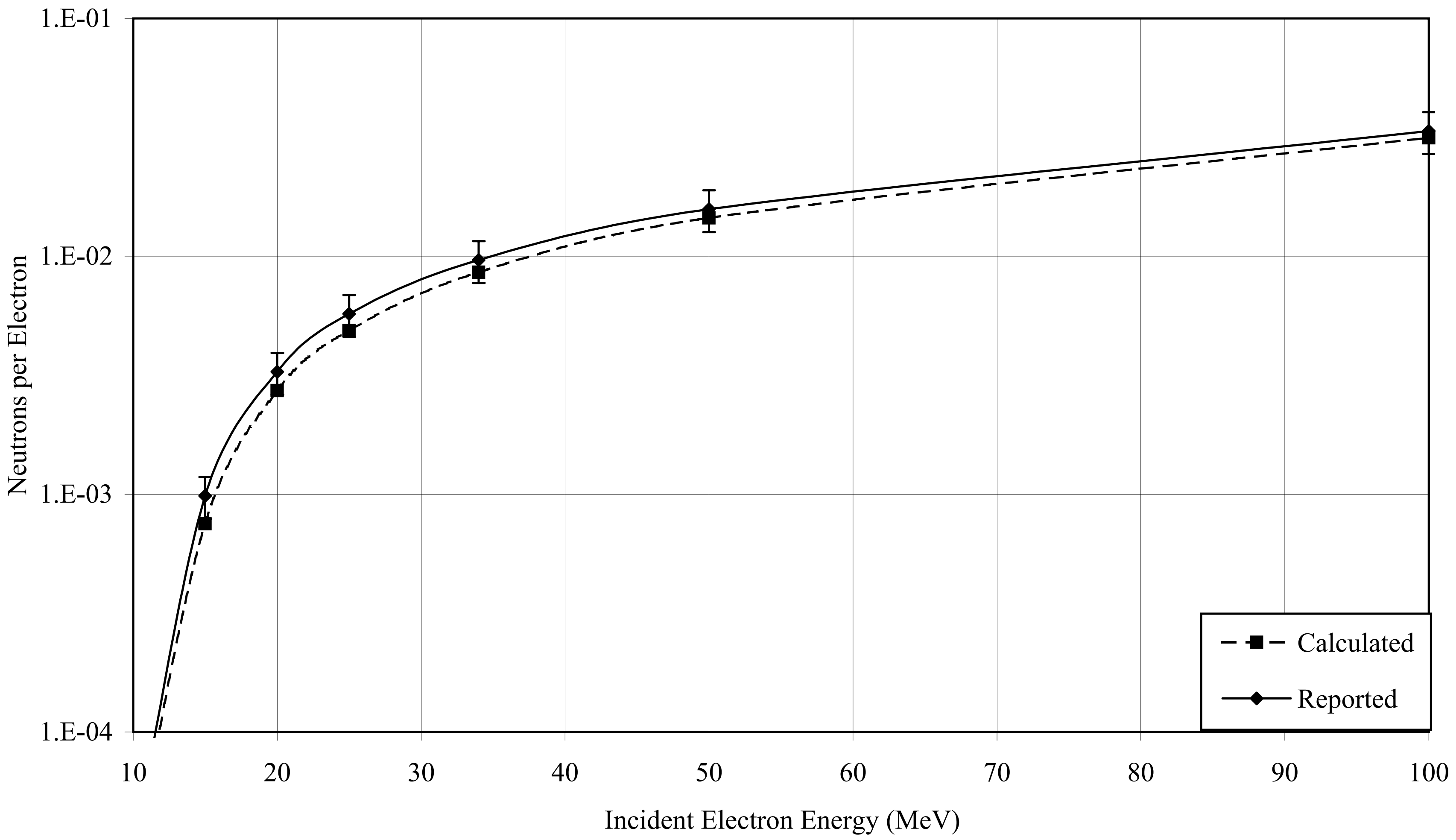

Figure 4-6. Calculated versus theoretical neutron yield for electrons of various incident energy on a thick lead target. (Reported values from Swanson, 1979.)

The Barber and George measurements were made at the Stanford Linear Accelerator Facility (SLAC). The measurements are of use primarily due to the care that was taken both in making the measurements and in documenting how they were made. The Mark II linear accelerator offered a well characterized electron beam. The SLAC facility presented a carefully controlled environment in which to make the measurements. The parameters of interest in representing the experiment as reported in the original paper are discussed below. Experimentalist please take note, these are the issues that must be addressed when defining a benchmark.

A diagram of the experimental layout is shown in Figure 4-7. The electron beam leaving the Mark II linear accelerator is collimated to a diameter of 0.187 inches. This collimated diameter translates to a 0.5 inch beam diameter at the target location for a nominal energy of 30 MeV. No information is provided on the variation in spot size with beam energy and it is shown to be negligible by later simulations. After leaving the collimator, the beam travels through the first of two identical 30-degree deflecting magnets. These magnets translate the main axis of the beam in order to reduce the background radiation escaping through the shielding wall.

The energy of the beam was variable over the range 10 to 36 MeV. The absolute energy was estimated to be calibrated to within 2 percent error by measurements using the photonuclear thresholds of deuterium, oxygen and copper. The energy spread is controlled by the two variable-slit collimators located between the two deflecting magnets. An energy spread of $\Delta E_0/E_0$ equal 2 percent was used throughout the experiment as set by the slit collimators.

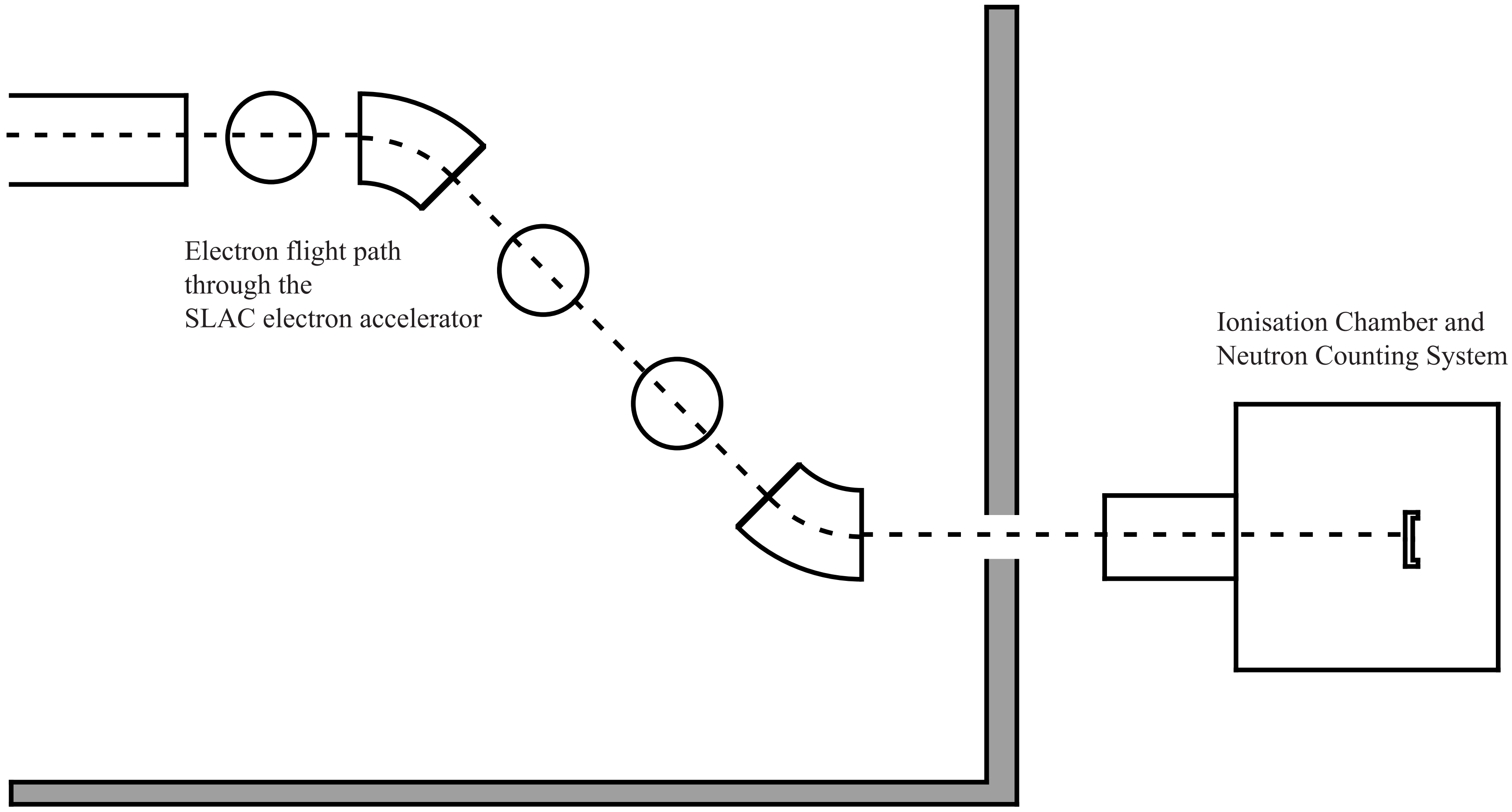

Figure 4-7. Experimental setup for the Barber and George experiments.

The final beam parameter needed for obtaining absolute neutron yields is the number of electrons per beam pulse. Barber had previously validated a method for determining this parameter [82]. An ionization chamber is located just before the experimental apparatus. It consisted of 0.005 inches of Mylar comprising two windows and an 8 inch thick chamber through which hydrogen flowed at one atmosphere of pressure. The response of the ionization chamber to the beam current had been well determined in the previous study. For this set of yield measurements, it was recalibrated at a few energy points by comparison with a Faraday-cup monitor. No estimate was given on the uncertainty in the electron beam intensity.

After transiting the ionization chamber, the electrons passed into a Lucite vacuum chamber and struck the target material. The chamber was 8 inches in diameter and 19 inches long. There were two target locations although the exact position of either is not reported nor the position used for each specific measurement. Lucite was chosen as the vacuum-chamber wall material because its constituents have high photonuclear thresholds and low neutron production cross-sections. A target size of 4.5 inches square was chosen to ensure that even electrons that underwent multiple scattering in the ionization chamber would still strike the target. The only important point not discussed in the paper is if measurements were made to determine the background level, i.e. the neutron count rate with the electron beam and no target present. The comparisons to simulations seem to indicate background levels were considered.

The absolute yield was measured by surrounding the target in a large paraffin-moderator box. The 32 inch square box extended 19 inches along the beam axis. Moderated neutrons were detected by two enriched BF_3 proportional counters extending the length of the box and located symmetrically about the target chamber. Background due to neutrons produced outside of this box was reduced by cladding the box with a thin layer of boron carbide and an outside layer of paraffin 8 inches thick. Again, it is not discussed if a measurement was made to determine the background counting level and if it was accounted for in the reported data.

The absolute counting efficiency of the neutron box assembly was determined with a RaBe neutron source and verified with measurements on photonuclear production in heavy water. For the RaBe source, an efficiency of (0.92±0.05) percent was observed. The D(γ, n)H reaction presents a system in which the neutron yield could be calculated with reasonable accuracy. Using the efficiency determined from the RaBe source, comparison of the calculated versus the measured yields for the heavy-water system agreed within the limits of the relevant uncertainties.

The electron beam delivers an intense photon pulse that temporarily overloads the counting apparatus. To counter this effect, the scaling circuits are gated off for 7.5 microseconds following the beam pulse to allow the associated circuitry to recover from the large pile-up. Since the lifetime of thermal neutrons in the paraffin is much longer than this period, this method is estimated to introduce minimal error (less than three percent). It was further necessary to limit the beam intensity for the high-Z materials in order to maintain this same gating time. It is not reported what effect changing the beam intensity might have, if any, on the other parameters.

Barber and George estimate the uncertainty of their results using the experimental apparatus described above to be 15 percent. Table 4-5 lists the experimental parameters of interest for the subset of experiments that could be modeled using the evaluated data

Table 4-5. Targets and essential experimental parameters are given as used to simulate the experiments of Barber and George (1959).

Target	Density (g/cc)	Thickness (g/cm^2)	Energies (in MeV) for which measurements are reported
Al-I	2.699	24.19	22.2, 28.3 and 34.3
Cu-I	8.96	13.26	16.1, 21.2, 28.3, 34.4 and 35.5* (*only for Cu-I)
Cu-II		26.56	
Cu-III		39.86	
Cu-IV		53.13	
Ta-I	16.6	6.21	10.3, 18.7, 28.3 and 34.3
Pb-I	11.35	5.88	18.7, 28.3 and 34.5
Pb-II		11.42	
Pb-III		17.30	
Pb-IV		22.89	
Pb-VI		34.42	

available at this time. In general, the portion of the target identifier, e.g. Al-I, after the dash indicates the approximate thickness of the target in units of radiation length. As their results were presented as a series of graphical figures, it was necessary to digitize and estimate the value of both the yield and energy for each point of interest.

Simulation Setup

The simulation of the experiment was able to simplify the layout because of the design of the experiment and the way in which the yields were reported. With the information available, it was only necessary to model the electron beam incident on the target material and tally the neutrons exiting the target boundaries. All other complications have been estimated by the original authors. Thus a total neutron yield, i.e. the number of neutrons escaping the target, per incident electron can be computed using the simplified schematic shown in Figure 4-8.

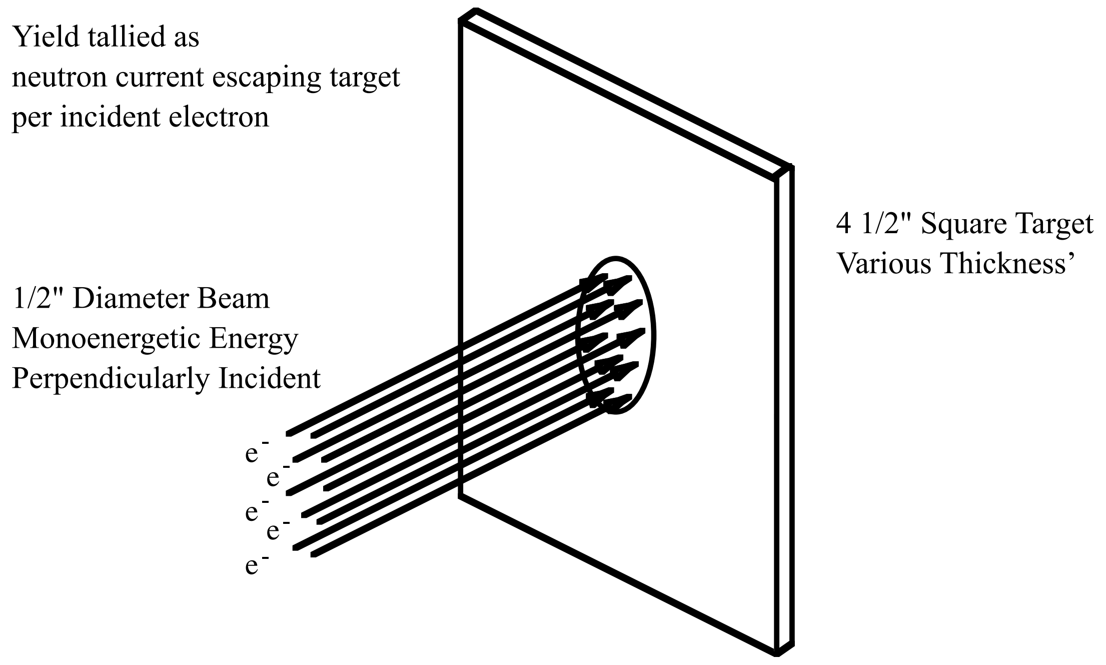

Figure 4-8. Setup used for simulation of the Barber and George experiments.

The electron beam model was further simplified to reduce complications in the input specifications. The relationship of the beam parameters, i.e. how the energy varied as a function of radius in the beam and its subsequent angular distribution, was not documented. The beam was therefore modeled as a mono-energetic, perpendicularly incident source directly on the target.

An estimate of how much a variation in energy could effect the results was made by adjusting the absolute energy of the beam. A one radiation thick tantalum target with a 28.3 MeV electron beam perpendicularly incident was used as a baseline for such changes. The results are presented in Figure 4-9. As the beam spread should be Gaussian about the mean, the net effect of the energy tails should cancel out. This estimate also serves as a predictor for uncertainties due to possible inaccurate reporting of the absolute value of the energy.

Deviations in angle are more difficult to estimate. One method is to look at changes in target thickness. This was done for the baseline case as described above and Figure 4-9 shows that possible deviations in thickness are not negligible. Deviations in incident angle of 1, 2, 3 and 5 degrees result in changes in apparent thickness of 1.7, 3.5, 5.2 and 8.7 percent, respectively. However, since these parameters are undocumented it is believed that a mono-energetic beam is adequate for the level of accuracy desired and that the error induced by variations in the beam should be considered as represented in the experimental uncertainty. As a last note, variations from a beam radius of 0.25 inches were considered and shown to have negligible impact.

Similar to the simulations used for comparison to the Swanson study, the materials with more than one isotope in the natural elemental form had to be represented

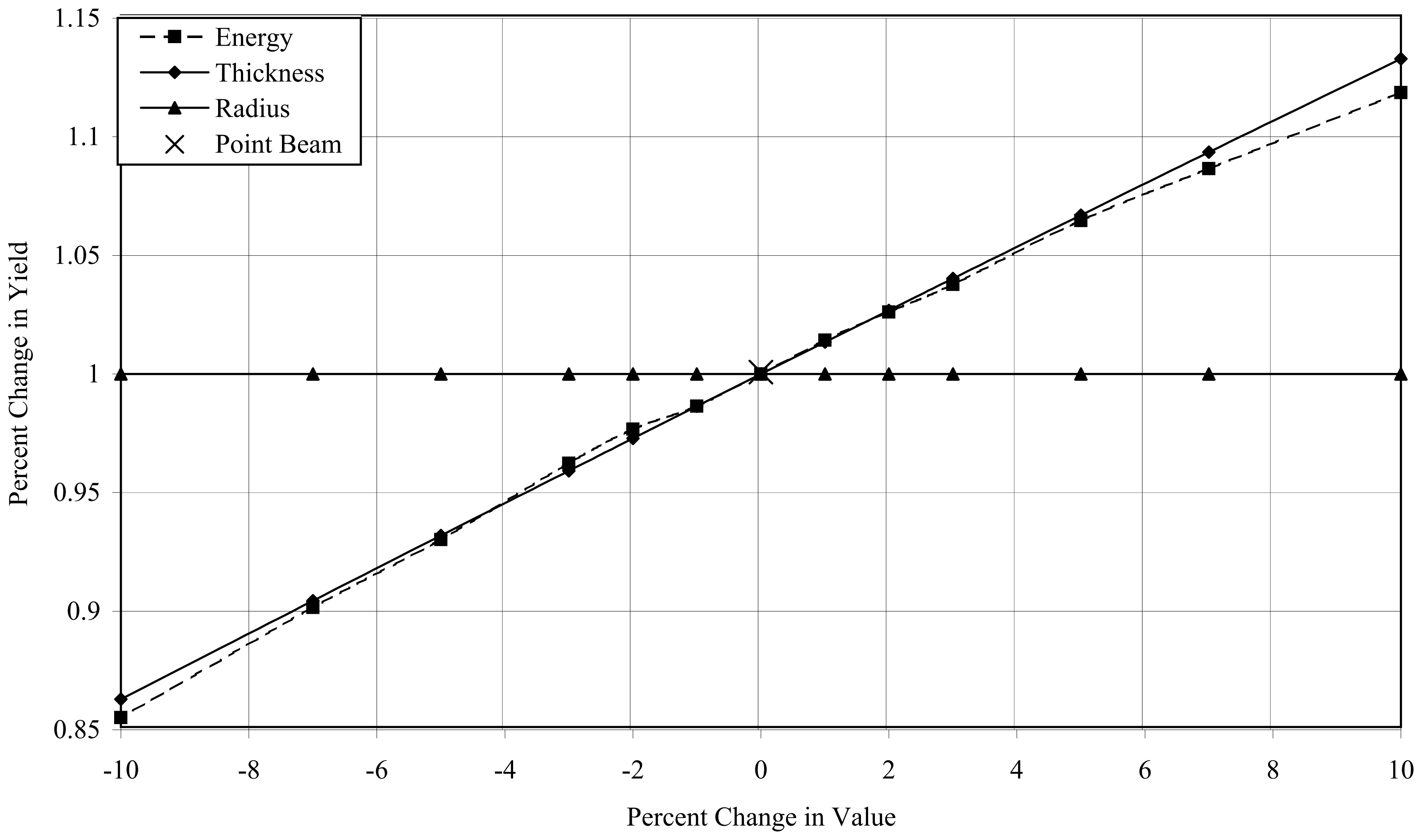

Figure 4-9. Percentage variation in absolute yield as a function of the percent change in various beam parameters.

with the evaluations available. Table 4-2 shows the exact details of which evaluations were used for each target material. Implications of these substitutions will be discussed in the comparison section.

There are several other points worth mentioning about the simulations. The photonuclear threshold marks the lowest energy photon that could produce a neutron. It is therefore unnecessary to track either photons or electrons below this energy. Further, forced collisions can be turned on in the target material such that every photon traversing the target undergoes at least one collision. Photonuclear biasing can be used to force a contribution to photonuclear interaction from each collision. This combination of techniques allows the relative error of the calculated yield to be minimized using run times of approximately 10 to 50 minutes on a single processor of an Origin2000 system. Note that the relative error is a measure of the precision of the calculation, not the accuracy of the simulation. The MCNP input decks as well as the reported and calculated yield values are listed in Appendix D.

Comparison to Current Calculations

The comparisons between the experimental data and the calculations are presented here as a set of graphical figures. The error bars on the experimental values are the 15 percent uncertainty quoted by Barber and George. The calculated yield values are taken directly from the relevant MCNP output deck. The relative error from the calculation is negligible. The uncertainty in the simulations will be discussed in the conclusions. Each figure presents the experimental data as diamonds connected by a solid line with calculated values represented as squares connected by a dashed line.

The comparison of the experimental versus calculated yield for aluminum is shown in Figure 4-10. The agreement shown is reasonably good although it is uniformly low by about 20 percent. As the first data point is at 22 MeV, these data support no further conclusions about possible changes needed in the threshold to peak region of the cross section suggested by the comparison to Swanson's yield values.

The comparison of the experimental versus calculated yield for the four thickness' of copper are shown in Figures 4-11 through 4-14. All four comparisons show similar results. They show good overall agreement with the experimental results, within 15 to 25 percent. As discussed above, the elemental copper has been represented by isotopic ^{63}Cu. The addition of a ^{65}Cu evaluated data set will improve the agreement, probably by five percent overall, and also improve the match of the shape.

The comparison of the experimental versus calculated yield for tantalum shows excellent overall agreement. As seen in Figure 4-15, the results for the region away from threshold are almost identical. However, the calculated value at 10 MeV is an order of magnitude too low. As the threshold energy is believed correct, this raises the possibility that the values of the cross section between threshold and the GDR peak needs to be increased.

The comparison of the experimental versus calculated yield for all five thickness' of lead are shown in Figures 4-16 through 4-20. The results are consistent for all five thickness'. The calculations are generally 20 percent lower than the experimental yields. This is acceptable agreement.

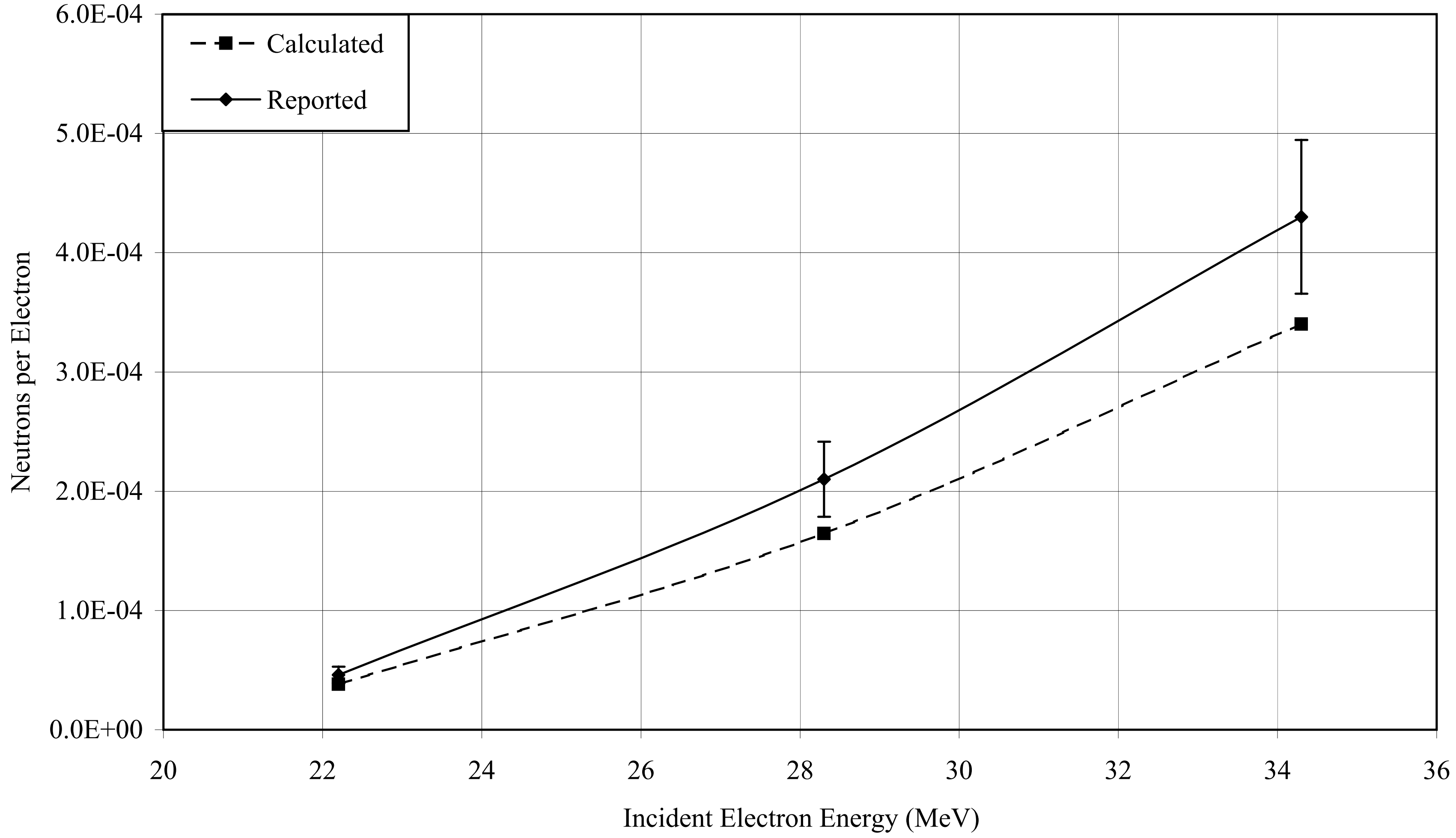

Figure 4-10. Calculated versus experimental neutron yield for electrons of various incident energy on a one radiation length thick aluminum target. (Reported values from Barber and George, 1959.)

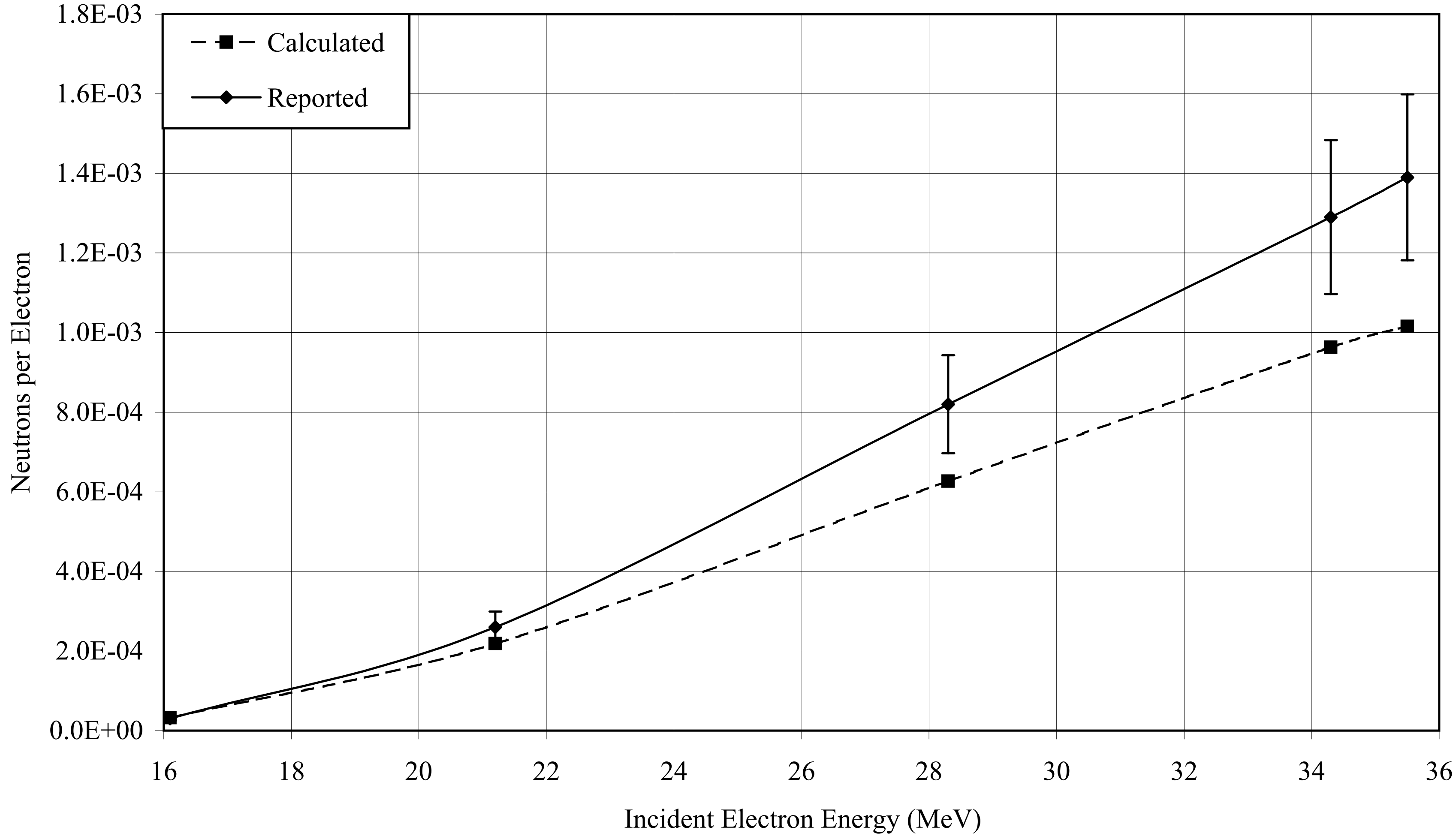

Figure 4-11. Calculated versus experimental neutron yield for electrons of various incident energy on a one radiation length thick copper target. (Reported values from Barber and George, 1959.)

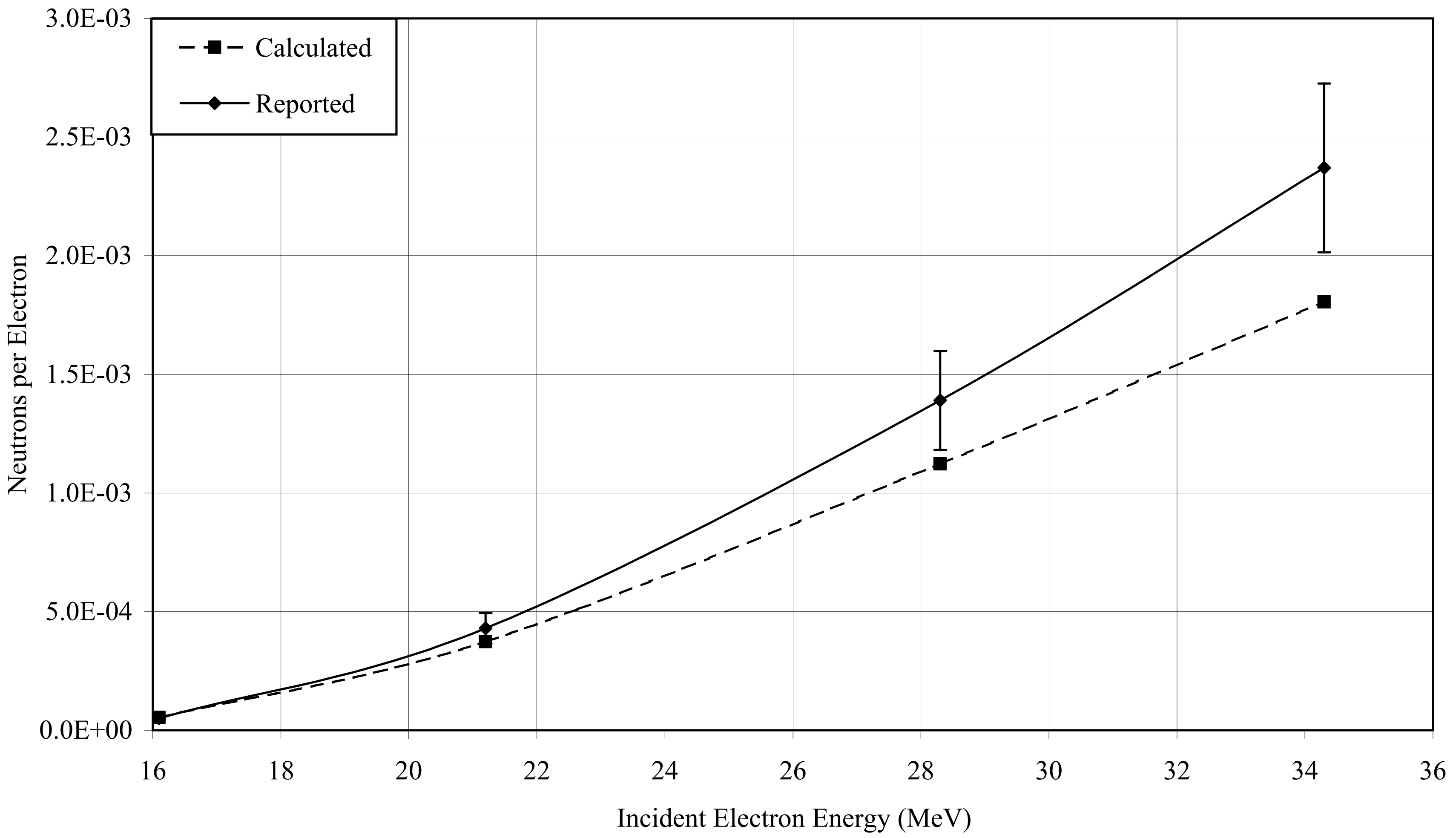

Figure 4-12. Calculated versus experimental neutron yield for electrons of various incident energy on a two radiation length thick copper target. (Reported values from Barber and George, 1959.)

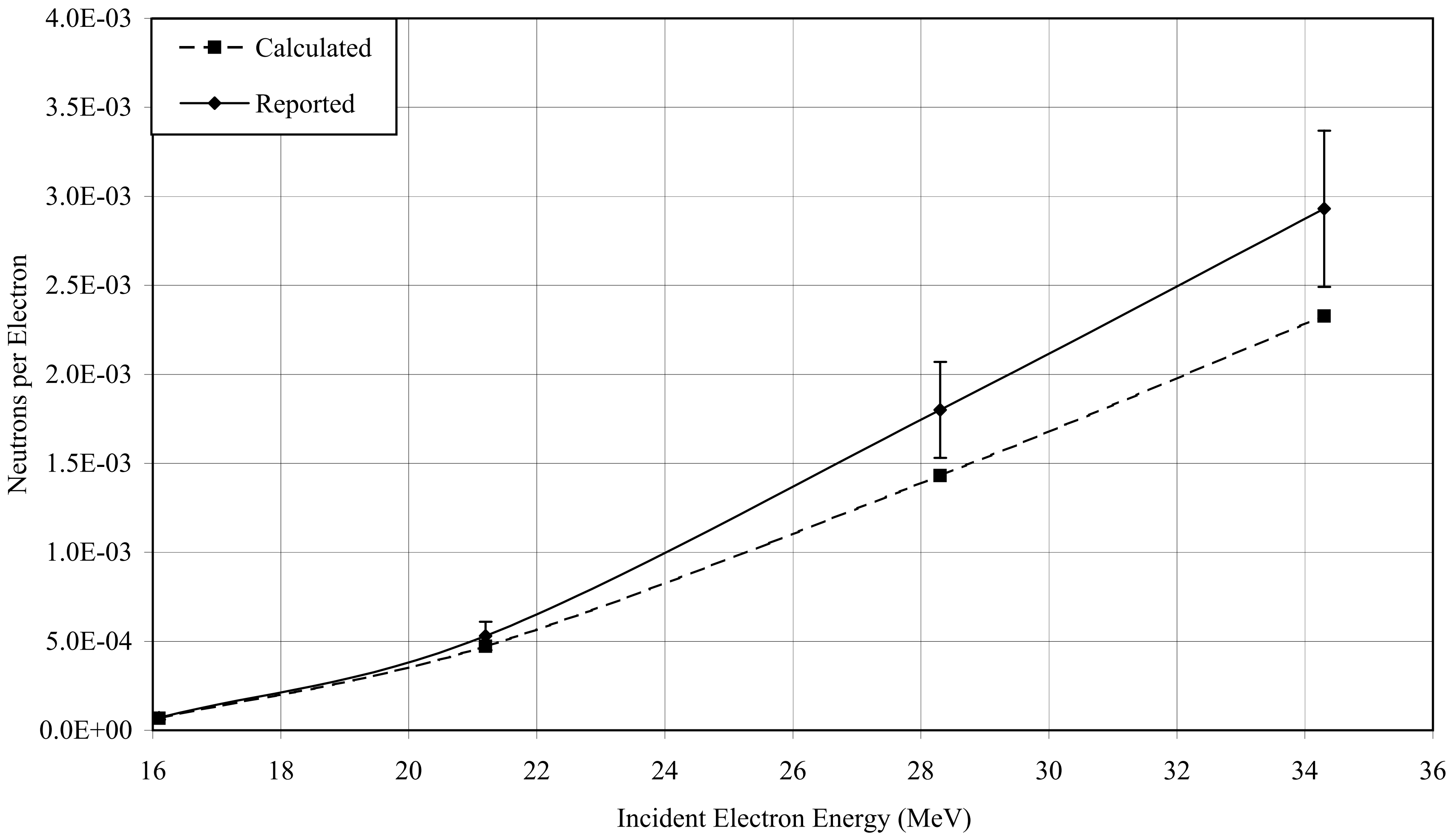

Figure 4-13. Calculated versus experimental neutron yield for electrons of various incident energy on a three radiation length thick copper target. (Reported values from Barber and George, 1959.)

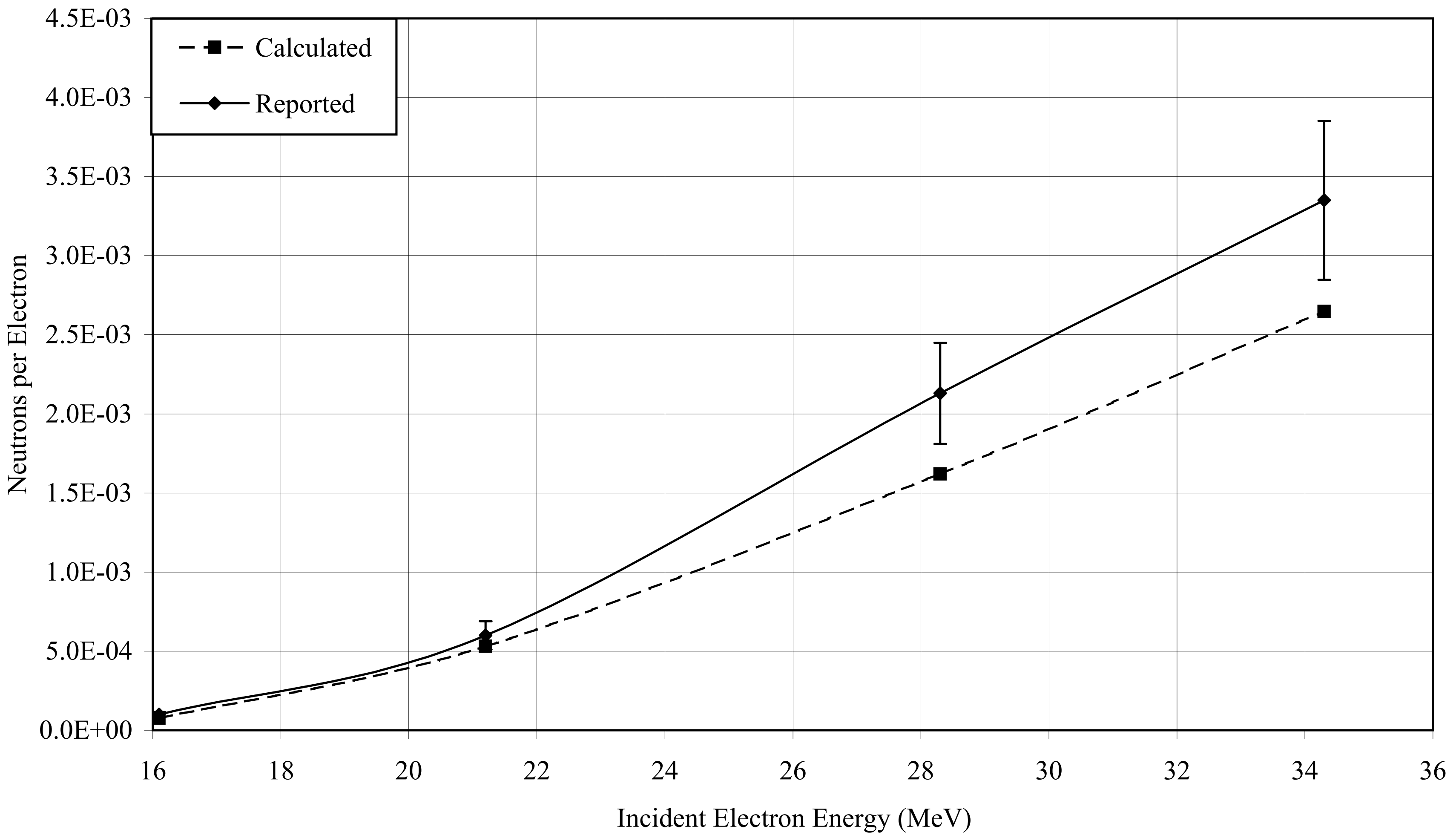

Figure 4-14. Calculated versus experimental neutron yield for electrons of various incident energy on a four radiation length thick copper target. (Reported values from Barber and George, 1959.)

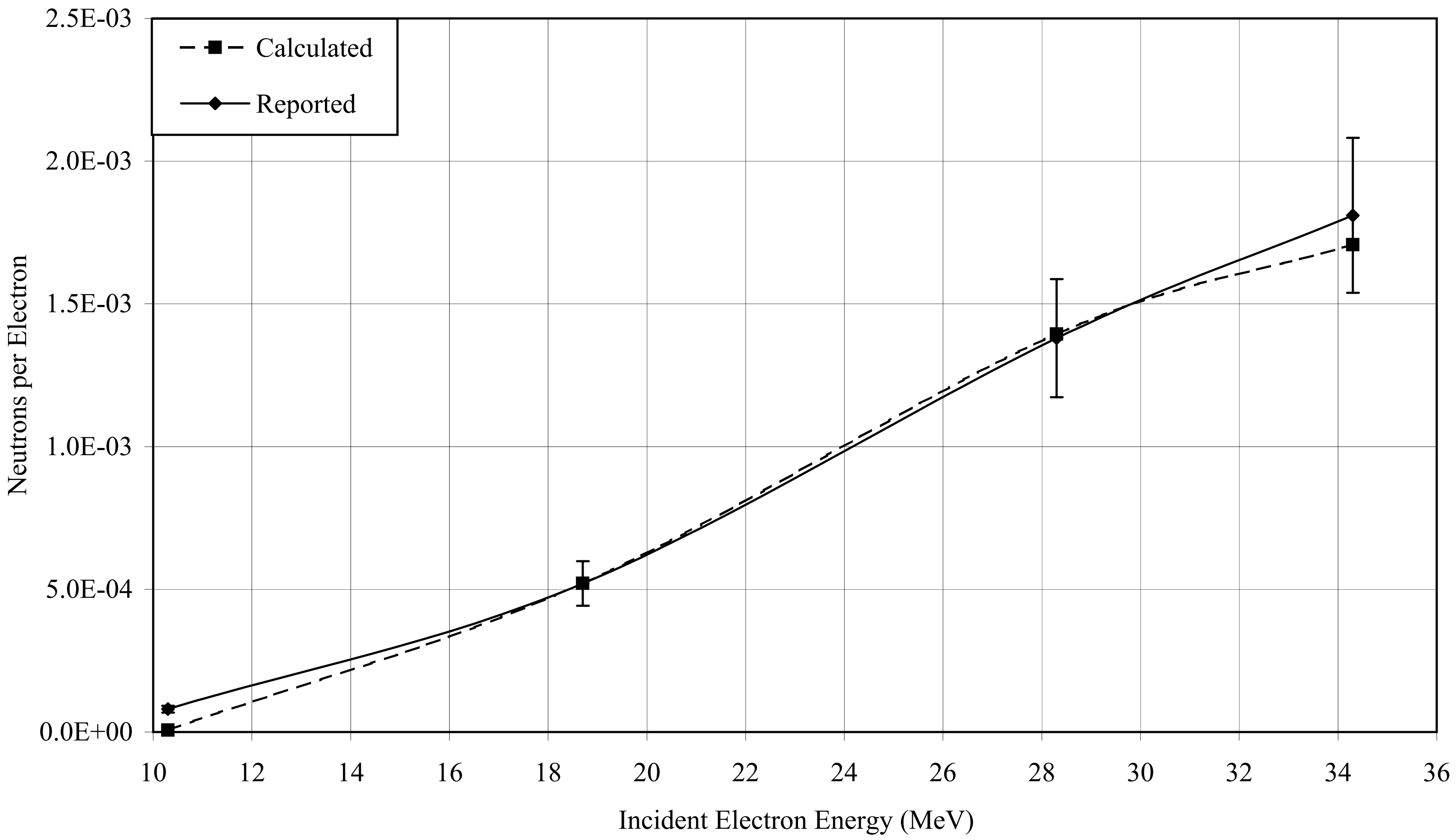

Figure 4-15. Calculated versus experimental neutron yield for electrons of various incident energy on a one radiation length thick tantalum target. (Reported values from Barber and George, 1959.)

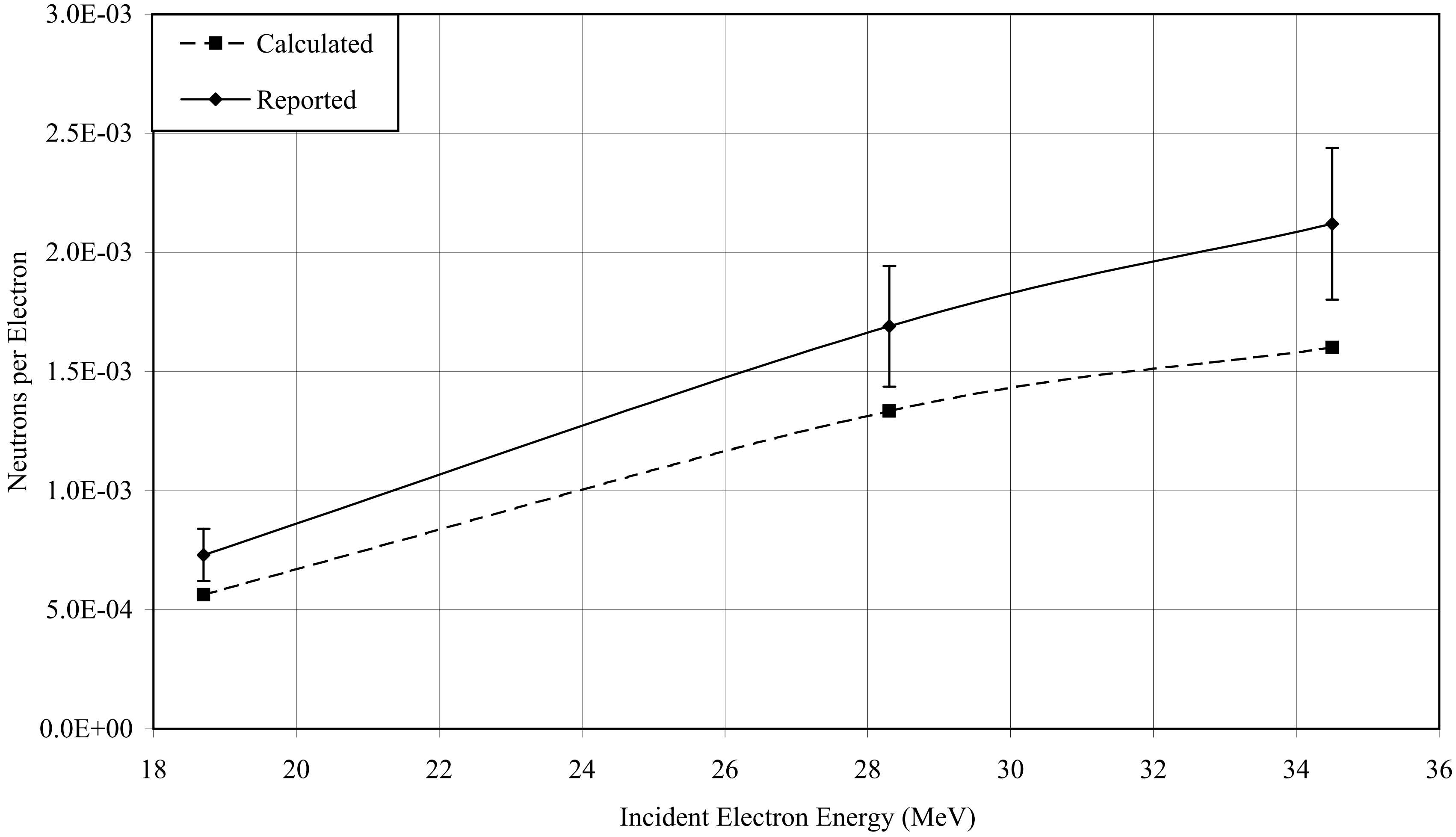

Figure 4-16. Calculated versus experimental neutron yield for electrons of various incident energy on a one radiation length thick lead target. (Reported values from Barber and George, 1959.)

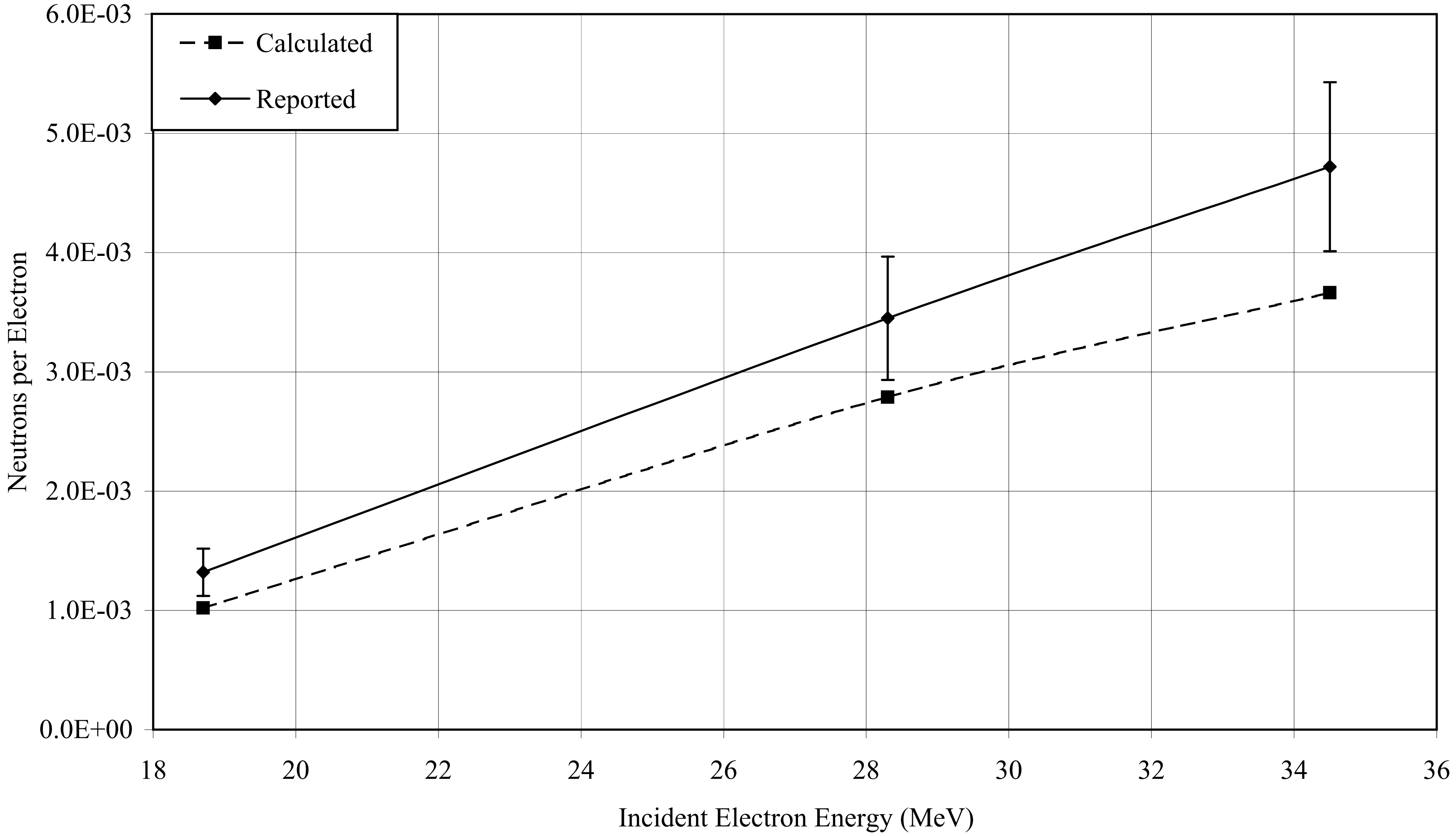

Figure 4-17. Calculated versus experimental neutron yield for electrons of various incident energy on a two radiation length thick lead target. (Reported values from Barber and George, 1959.)

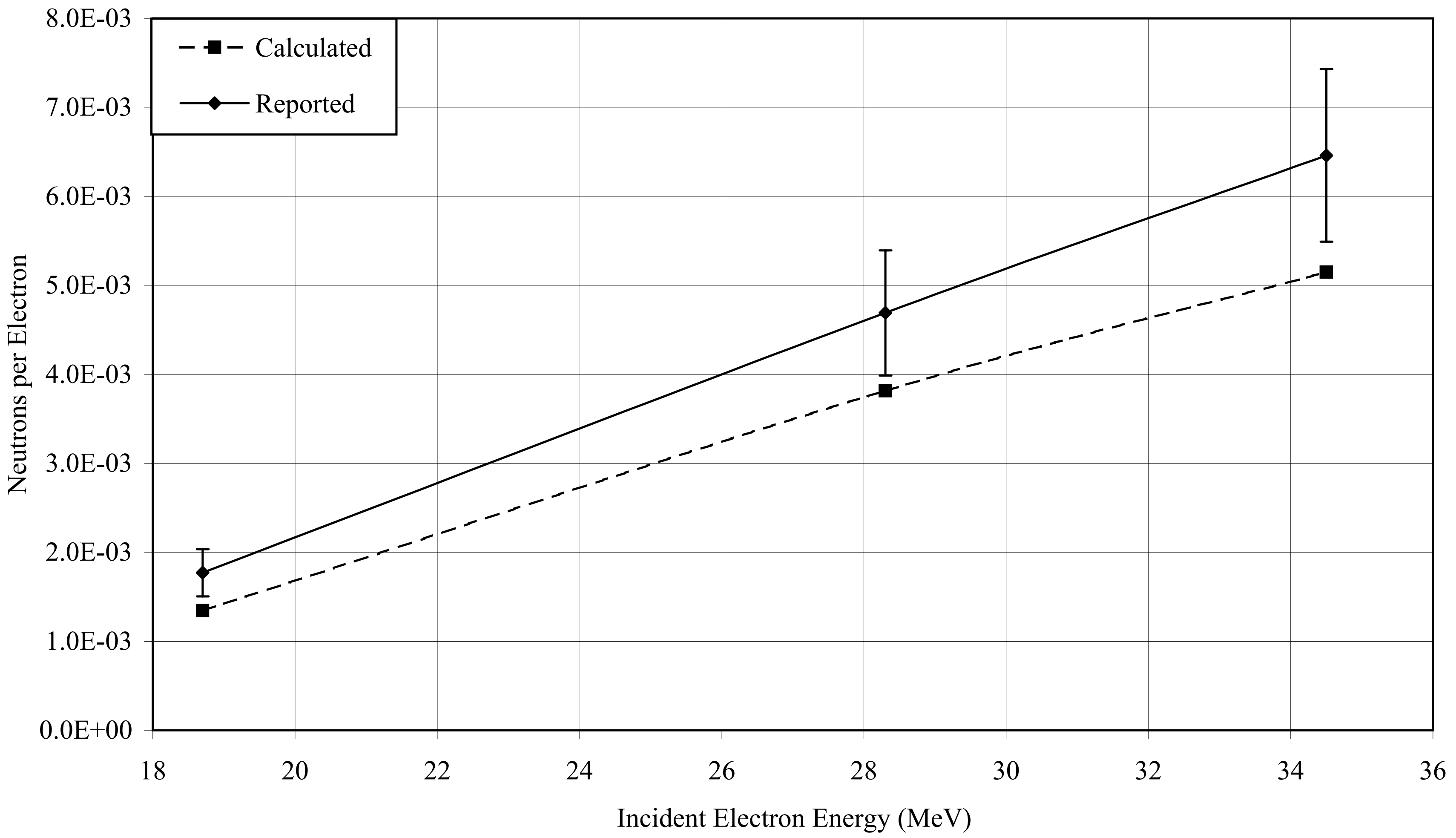

Figure 4-18. Calculated versus experimental neutron yield for electrons of various incident energy on a three radiation length thick lead target. (Reported values from Barber and George, 1959.)

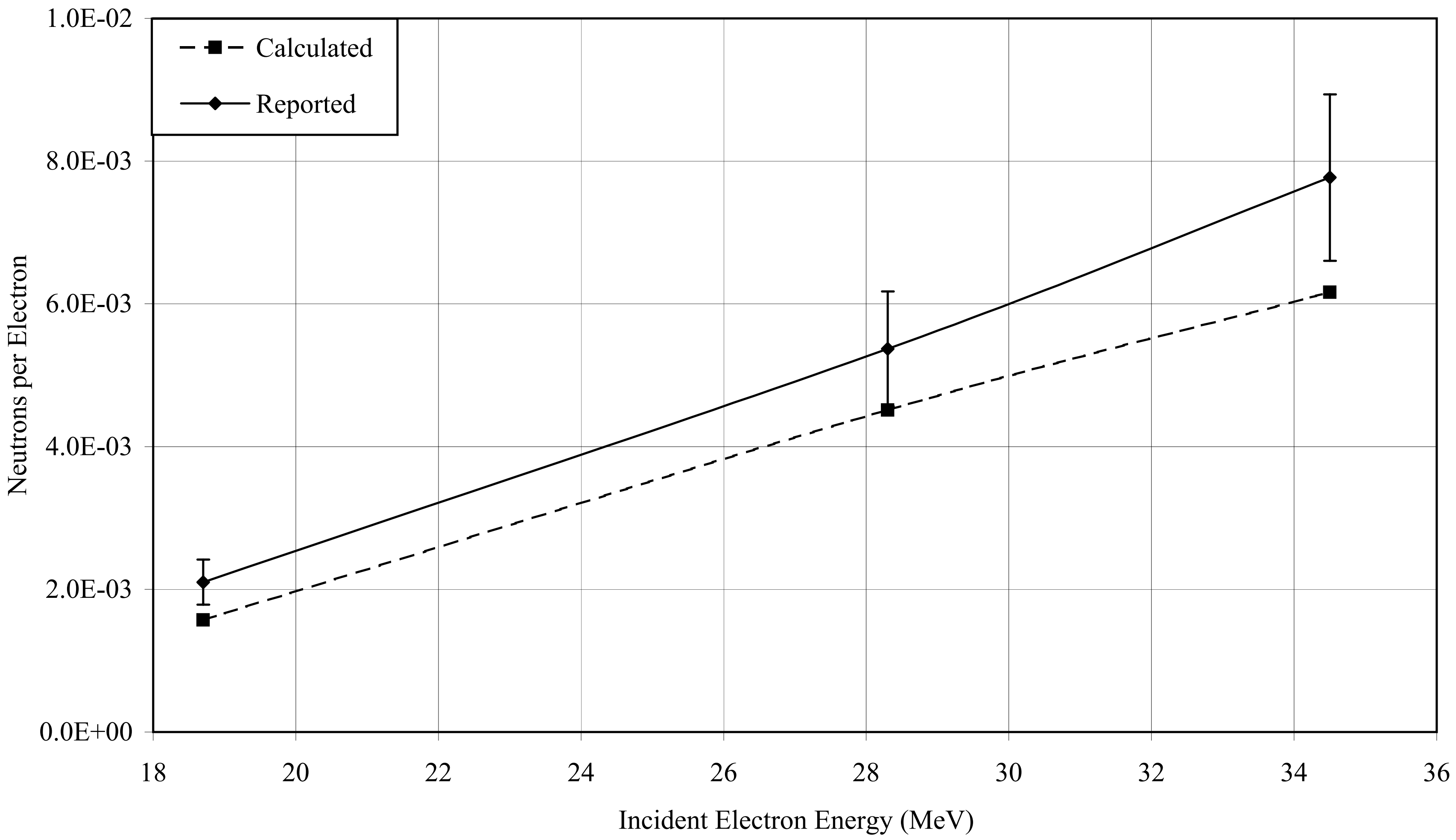

Figure 4-19. Calculated versus experimental neutron yield for electrons of various incident energy on a four radiation length thick lead target. (Reported values from Barber and George, 1959.)

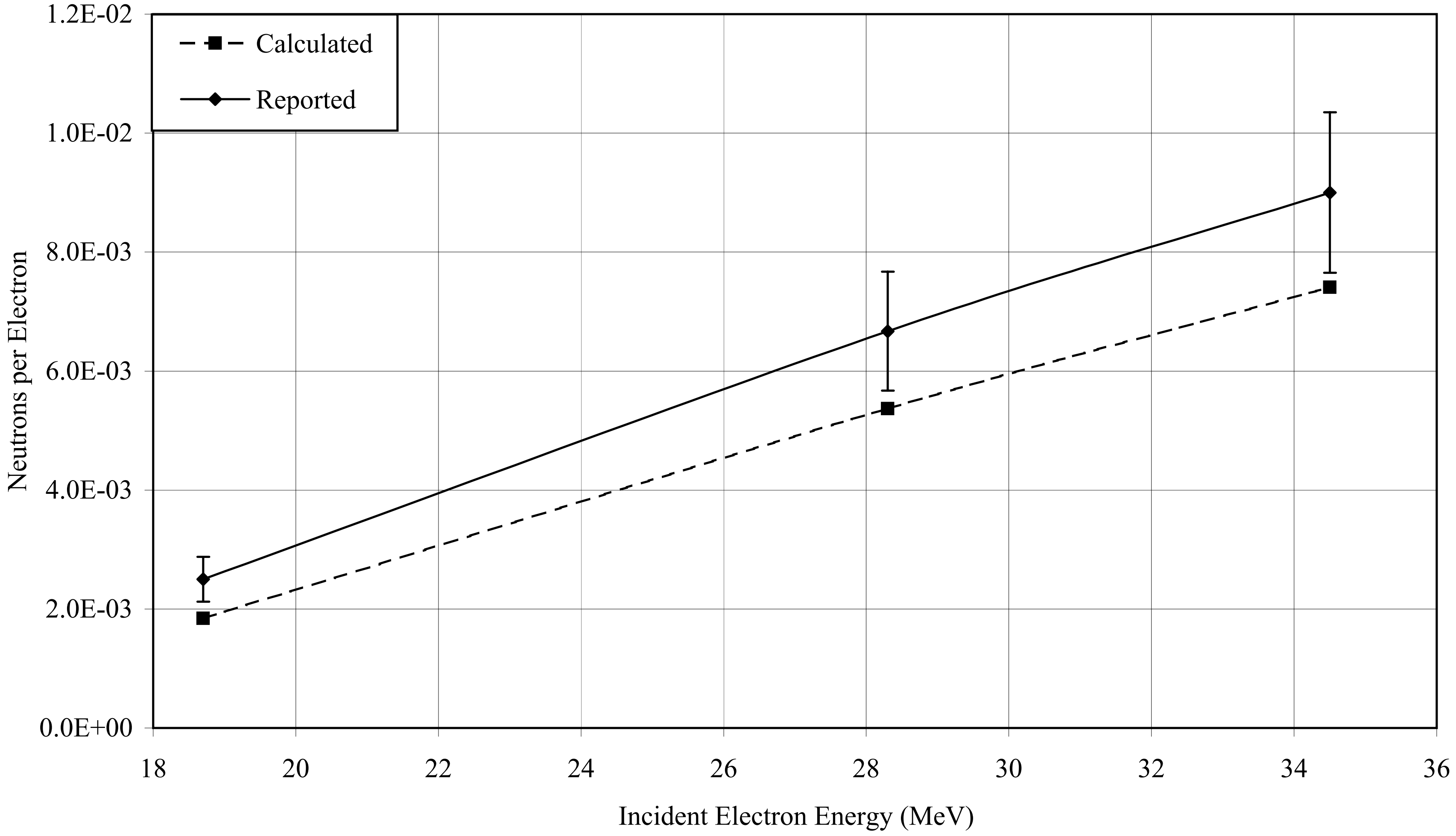

Figure 4-20. Calculated versus experimental neutron yield for electrons of various incident energy on a six radiation length thick lead target. (Reported values from Barber and George, 1959.)

Conclusions from Verification and Validation

It has been shown that the current evaluated data and the new photonuclear interaction coding provide reasonable results for simulating neutron production from materials for which an evaluated data set exists. Most important to the current work, these comparisons have covered the prominent target, filter and shielding materials used in medical electron accelerators. The overall uncertainty in the evaluated data is estimated to be less than 25 percent.

As a last note in reference to Swanson's theoretical yields, these comparisons are considered something more than simple verification but not quite true validation. Good agreement is seen between most of the data sets but those are typically based on the same underlying experimental data. It is worth noting that analytical shower theory and Monte Carlo electron-photon transport appear to provide similar results. This argues that the neutron production is not highly sensitive to the photon production mechanism.

The evaluated photonuclear data library, like all other nuclear data libraries, will evolve as practical experience using the data feeds back into the evaluations. This will be a long process. The first part of this process should involve will be the further validation of the evaluated photonuclear data as it becomes available. Hopefully, the ability to simulate this class of problems will encourage experiments worthy of becoming benchmarks.

To repeat this theme, validation data for the purposes of photonuclear physics is sorely lacking. Only aluminum, copper, tantalum and lead have been directly validated against true integral experimental data. The remainder of the evaluated data are "validated" only in the sense that they were created through the same process and have derived from the best known cross-section measurements. This also elucidates the point that further measurements of photonuclear cross sections will be necessary for isotopes previously not measured or with multiple measurements that disagree.

It should also be noted that only the photoneutron yields are directly validated by the simulations above. Emission characteristics, the energy and angle of the secondary particles, are only validated in the sense that GNASH [41] has a long standing, well-validated ability to produce such data. Experiments measuring energy and angle spectra are needed.

Last, it is recommended that once this ability is made generally available, each user community perform its own experiments to validate the data and code for their class of problem. This set of validation activities shown above has set an uncertainty only on the use of this data and coding for the calculation of neutron yields for simple geometries and materials which have photonuclear evaluated data available. Simulations involving production and transport through complex physical geometries and containing materials for which evaluated data may not be available will be considerably more difficult. It is incumbent on the user performing such work to understand the uncertainties of their specific simulation.

CHAPTER 5
APPLICATION: SIMULATION OF A MEDICAL ELECTRON ACCELERATOR

Introduction

It is now time to remember that the original motivation for this work was the enhanced understanding of the radiation environment in the vicinity of a medical electron accelerator (MEA). To this end, this chapter will discuss the previously available ability to simulate electron-photon transport within a MEA and the extension of this ability by the current work to include photonuclear physics and simulate electron-photon-neutron transport. Two major sections are presented.

The first section demonstrates the ability to simulate electron-photon and electron-photon-neutron environments by comparison to experimental data. The data were obtained from measurements around the Phillips SL Series MEA located in Treatment Room 5 of the Shands Cancer Center (SCC) at the University of Florida (UF). The validation is further divided to cover the electron-photon model and the electron-photon-neutron model. Once the simulation model has been validated, it can then be used to investigate the interesting aspects of the radiation environment.

The second section of this chapter explores two questions of interest for MEAs in general. The first question of major importance is the determination of the dose, particularly the neutron dose, to the patients and workers in the vicinity of the machine. The second question is the effect of increasing electron energy on the relative neutron to photon dose. Within both of these discussions, some fundamental lessons learned are highlighted and several issues suitable for future work are proposed.

Validating the Simulation

Background

A computer model may generally be defined as the virtual world necessary to adequately simulate the corresponding real world. The virtual world is never an exact reproduction of reality but instead attempts to capture the essential details. The major items necessary for radiation transport simulations are representing the physical environment, providing transport data for the constituent materials, describing the initial radiation source, implementing the transport algorithms to propagate the radiation source and obtaining the resultant output information. It is worth digressing for a moment to discuss in general terms what is important to each of these tasks and why.

Physical geometry. A standard MEA treatment room is an extremely complex physical space. First and foremost of concern to the simulation is the electron accelerator itself. This is typically a linear accelerator or a cyclotron consisting of a few thousand pounds of meticulously engineered parts designed to deliver an intense radiation field to a precise location. The bulk of this equipment is usually located within the treatment room in a machine closet such that only the treatment head is visible in the main room. To further complicate matters, MEAs are often designed with treatment heads capable of delivering either electron or photon fields at multiple energy levels and involving extremely complex parts. Finally, the as-installed configuration can include changes from the original specifications for particular needs including increased shielding.

Surrounding the MEA is the treatment room itself. It serves two essential purposes. First, it is there to contain the radiation for the protection of those in the vicinity. Second, it provides an esthetically-soothing, working environment in which to treat cancer patients. For the purpose of shielding the radiation, these rooms are typically constructed as concrete vaults though in some cases other shielding material may be used. For the purpose of creating a working space, they are typically finished with wall-board, drop ceilings, carpeting, cabinetry, instrumentation and associated miscellaneous items typical of a clinical environment.

For the goal of simulating the electron-photon dose to a patient, it is only necessary to include detailed modeling of the MEA treatment head. Specifically this would include the electron target or scattering foil and all materials in the immediate vicinity of the subsequent beam path. In the typical electron-photon simulation, e.g. [83,84], only the target, collimators, filters and associated items are modeled.

Jumping slightly ahead, a typical dose calculation that uses a complete room model in addition to the treatment head model gives the following characteristic results. The electron-photon dose to the patient from a typical treatment using a Phillips SL Series MEA in the high-energy photon mode consists of the following sources: approximately 80 percent directly from photons produced in the electron target; approximately 15 percent from photons produced or scattered in the filters and other materials in the beam path; approximately four percent from photons produced or scattered in the collimators; and less than one percent from room return and other sources. Hence, there is justification for ignoring anything very far outside of the electron-photon beam path for most electron-photon dose calculations.

Simulating electron-photon-neutron transport makes life much more difficult. Neutrons are notorious in the world of health physics for their ability to penetrate materials either directly or by finding streaming paths. In practical terms, this means that the exact geometry of the MEA and the room are necessary for accurately representing the simulation environment. With their ability to scatter, any path between the neutron production site and the patient or worker is a concern. However, even if complete shielding coverage could be achieved, their penetrating ability ensures that some level of neutron radiation will always be present.

Again, it is important to consider some characteristic results from later simulations here in order to set an appropriate frame of mind. The neutron dose for two standard photon field sizes, 5x5 and 30x30 sq. cm. respectively, consists of the following sources: approximately 56-36 percent from room return; approximately 16-44 percent from neutrons produced or scattered in the primary collimator; approximately 21-12 percent from neutrons produced or scattered in the secondary collimators; and the remainder from neutrons produced or scattered at other locations in the treatment head. Hence, for an accurate simulation it is necessary to account for the placement of practically all the material around the treatment head and most of the material in the room itself.

Transport data. Once the extent of the physical space is known, it is necessary to provide transport data for each material found in the geometry. Transport data are defined here as tabulated listings of interaction probabilities, i.e. reaction cross sections, and the emission spectra for the resultant particles suitable for use in Monte Carlo transport algorithms. For traditional MCNP simulations, these data are available by element for incident electrons and photons and by isotope for incident neutrons. Note that the photon data to date has only included photoatomic interactions. The current work has integrated photonuclear data, by isotope, into this mix.

Electron and photoatomic transport data exist as a complete library for the elements from hydrogen to plutonium (Z equal 1 to 94) in the standard MCNP distribution. Photoatomic data are provided over the incident energy range from 1 keV to 100 GeV [85] and electron data over the incident energy range from 1 keV to 100 MeV. This means that electron-photon problems can be simulated for any typical condition found in a MEA treatment room. The exclusion of photonuclear data from photon interactions was found to make no significant difference in the electron-photon portion of the dose calculated for a typical treatment plan.

Neutron and photonuclear transport data exist for a limited selection of isotopes and over a varying range of incident energies. In the case of neutron data, the coverage of isotopes is nearly complete in the incident energy range up to 20 MeV. There are some significant isotopes missing, e.g. any germanium isotope. There are also some materials which are represented as average "elemental" data rather than by individual isotope, e.g. magnesium. However, most of the major isotopes have neutron data available. Unfortunately, at this time photonuclear data exist for a very limited set of isotopes.

In a practical sense, this means that neutron and photonuclear simulations must make approximations in their representation of materials. Often this is trivial. For example, splitting the 0.13 atom percent of tungsten-180 among the other four isotopic data sets to represent elemental tungsten introduces negligible error into most simulations. However, serious error can occur when the missing constituent is more significant, e.g. the lack of a germanium data set to simulate neutron transport in a germanium detector. The problem of missing isotopes will play a significant role in simulations requiring photonuclear data until such time as the evaluation of the major isotopes is completed.

Radiation Source. Radiation transport starts from the creation, either artificially or naturally, of a radiation source. For a MEA operating in a photon mode, the radiation source is electrons which have been accelerated and directed onto a converter to produce bremsstrahlung photons. The final spatial-directional-energy distribution of the electrons incident on the target is system dependent. In most simulations, the final dose is not sensitive to the exact distribution of electrons and a simplified description is acceptable.

A typical simulation uses a mono-energetic beam, perpendicularly incident on a point on the target. This seems to be a fair approximation in the sense that it gives reasonable results. It can be improved by uniformly distributing the electrons over a spot rather than a point and by spreading the incident energy over a Gaussian distribution though these are expected to be rather modest gains in accuracy. To date, the author has not seen any work which methodically documents the effects of the variations in the incident electron beam distribution.

Because detailed electron-photon transport is difficult and time consuming over long distances, the radiation source is typically recomputed one or more times at intermediate locations in the geometry. Specifically, the upper section of the treatment head geometry is fixed for a treatment modality. One initial simulation of this immovable portion of the treatment head can be used to create a phase-space file [84,86] of the radiation field passing through a plane just above the first movable object. The phase-space file can be used as the radiation source for subsequent transport simulations though the remainder of the geometry. Obviously, this step can be repeated. Since there is typically only minor feedback from changing the lower geometry, this approach can save a significant amount of time while introducing relatively minimal errors.

Transport algorithms. Once the radiation source is determined and the physical world has been represented, the next step is to transport the radiation through the geometry. For neutral particle transport, e.g. of neutrons and photons, the Monte Carlo algorithms necessary to sample continuous-energy transport data are straight forward and well established. However, the algorithms used for charged-particle transport, e.g. of electrons, are a subject area still undergoing significant improvements.

Around this mix of transport algorithms, it is necessary to have a framework to handle all the other details. As the radiation propagates through the geometry, it is necessary to update the transport data for its current location. The distribution of sampling must be monitored to ensure that the phase space of the problem has been adequately covered. Summary information should be collected for later presentation. Error states should be checked and appropriate warnings issued. All of this should be as tightly coupled as possible.

It is generally accepted that separating portions of the transport tends to introduce error into the simulation. The only way to avoid these errors is to pass the next portion of the simulation a complete description of the necessary information. In the case described above where the geometry is separated, the phase-space file must contain as accurate as possible a description of the radiation source propagating into the subsequent geometry.

This same situation exists for coupling the transport algorithms. It is desirable to have one framework which includes the necessary components to handle electron, photon and neutron transport.

Obtaining output. The best simulation model in the world still has to be able to present the results in a reasonable manner. If the desired information is not conveyed to the user, or it is conveyed in a misleading manner, the simulation has not finished its work. The ideal would be to have all the details of everything that influenced the simulation available. For example, it might be useful to know the energy distribution of the photon flux as a function of spatial position over all locations. In practice, the information available is typically constrained by the amount of memory and time available to track these details. For the example, it might be enough to known the energy distribution of the photon flux within a cell or at a point. A fine balance is needed.

It is these five issues that must be kept squarely in mind when evaluating a radiation transport simulation. Understanding of these issues will lead to more thorough comprehension of how the simulation relates to the physical world. Only with adequate depth of knowledge can extrapolations be made back into the physical world.

Experimental Setup

The experimental data were obtained from the Phillips SL Series MEA in Treatment Room 5 of the Shands Cancer Center at the University of Florida. Two sets of experimental data were desired. The data desired and the basics of how they were obtained are discussed in general here. More details are provided in the following section as they relate to what was modeled in the simulation.

The layout of the treatment room is shown in Figure 5-1. The MEA treatment head is positioned as indicated and mounted on an extension such that it can rotate about a fixed point in space. That point is known as isocenter and is located on the center-line axis of the treatment head 100 cm source-to-surface distance (SSD) from the electron target. The room contains a set of laser lights that are aligned in the in-plane (wall to treatment head) and cross-plane (parallel to the maze) directions at the correct height such that they cross at isocenter. A mirror and light system can also project through the treatment head to indicate the SSD of an object in the beam path.

As part of the calibration of the MEA unit, a set of depth dose curves are taken. This procedure uses a 48x40x40 cm (width x depth x height) Lucite tank with 1 cm thick walls filled with water. The tank is placed such that it is centered directly below the treatment head and the surface of the water is 100 cm from the electron target (100 cm SSD, source to surface distance). A depth dose curve is the relative dose at each point along the central axis in the water tank starting from isocenter.

The relative dose is measured by use of an ion chamber and a positioning system. An Ion Chamber IC 10 connected to an Electrometer WP 5006 current monitor was used in this experiment. A WP600 Controller is used to control the position of the ion chamber and relay the position and current to a standard PC computer. The current and position information is stored to disk along with the details of the treatment mode, e.g. energy setting and collimator opening. A set of depth dose curves for photon field sizes of 5x5 sq. cm., 10x10 sq. cm. and 30x30 sq. cm. with the machine in the high-energy photon mode were obtained in this manner. The incident energy of the electrons in the high-energy mode of the MEA can be estimated by simulating this data.

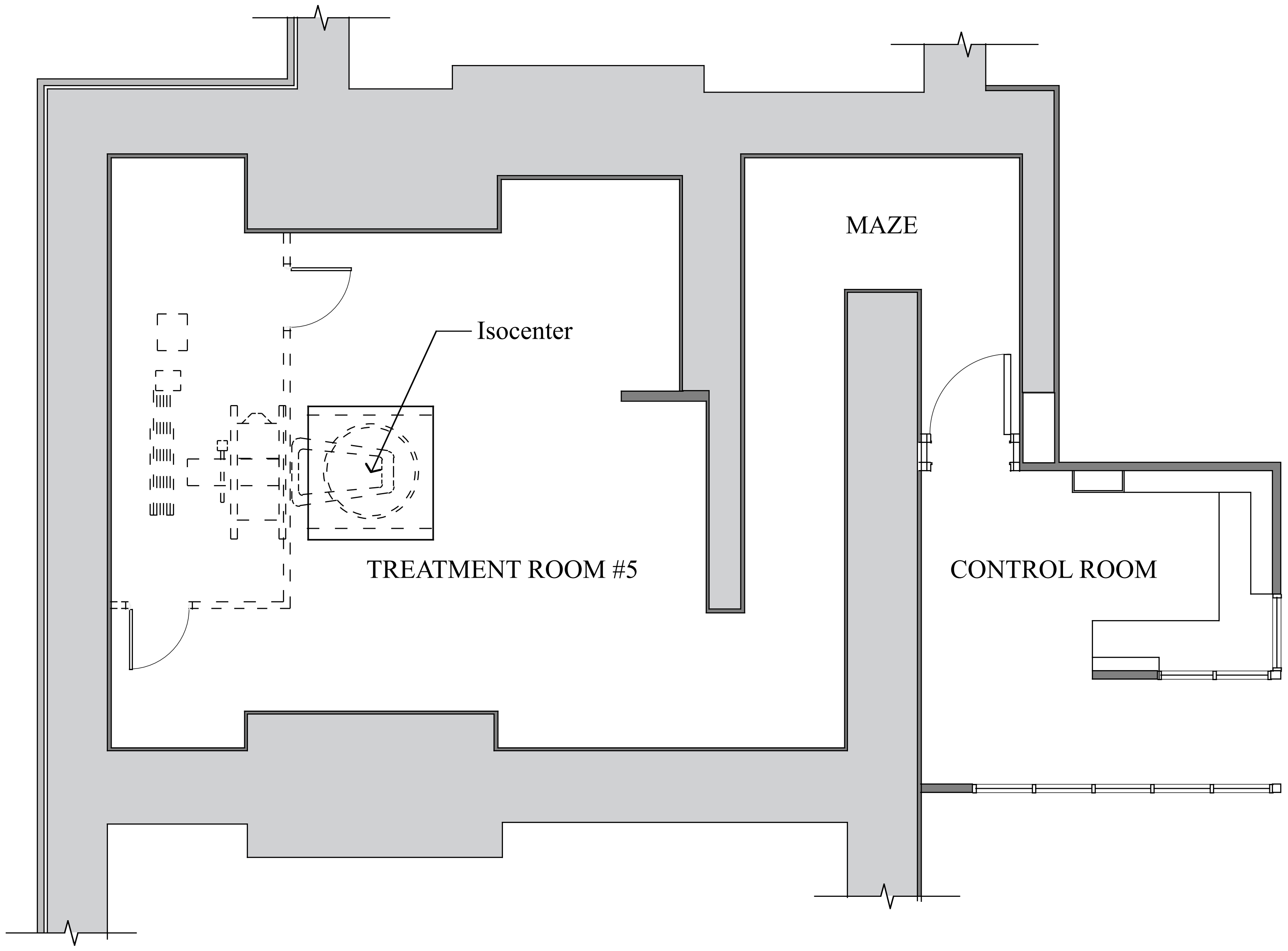

Figure 5-1. Diagram of Room 5 at the Shands Cancer Center at the University of Florida.

The second set of experimental data desired is an estimate of the absolute neutron production in the treatment room. One method to obtain this information is to measure the activation of a known sample of material in the presence of the accelerator's radiation field. Activation is the transmutation of a nucleus from one isotope to an unstable isotope. The decay of the unstable isotope is then measured and the number of such isotopes observed can be used to estimate the radiation field the original sample experienced. This technique is known as activation analysis. Many texts exist on the subject, e.g. Alfassi [87], as well as specific guidance for measurements around MEAs [88].

For the purpose of these experiments, gold was the material chosen. It has a number of useful properties. First, in its elemental form, gold is mono-isotopic. Therefore only one set of cross-section data are needed. The ^{197}Au(n,γ)^{198}Au cross section as a function of incident neutron energy is very large (98.8 barns for thermal neutrons). More importantly, it is commonly used in activation analysis and the cross section is well known, probably not more than 20 percent in error at any given energy with an aggregate accuracy of better than 5 percent. Last, the decay of ^{198}Au is well documented and easily distinguished for counting by gamma-ray spectroscopy.

Activation analysis requires a calibrated detector system capable of discriminating the radiation emission of interest. The Neutron Activation Analysis (NAA) laboratory at the University of Florida Test Reactor (UFTR) facility maintains a set of germanium detectors and associated equipment for this purpose. As their equipment was available for use, it was unnecessary to setup and certify a new system. The counting system utilizes the GammaVision software produced by EG&G Ortec.

Eleven small foils and three large ingots of gold were available for use in these experiments. Each sample was cleaned with alcohol, weighed, sealed within a plastic sleeve and numbered. The activation in the gold is a function of the number of atoms present and the number of neutrons available. Consequently, for the same neutron population, the larger the mass of gold, the more activation, i.e. ^{198}Au isotopes, produced. The original motivation for using the ingots was that their 31.1 g (one troy ounce) mass would activate quickly even in the relatively low neutron fluence of the treatment room. They could therefore be used for a set of measurements without the need for long irradiation times.

All of the samples had been subject to previous irradiation. The background counts present in all the samples were evaluated at the NAA laboratory prior to their irradiation. The foils had not been previously irradiated in more than one year and showed no significant background. The ingots had been irradiated within the treatment room slightly more than one month prior to this set of experiments. They showed a slight background which had to be subtracted off the later counts. The GammaVision software produces a report showing the energy boundaries of the gamma-ray peaks observed as well as the net counts seen for the peak and the estimated one sigma error.

Gold has a second reaction of interest for these experiments. The photonuclear (γ,n) threshold for ^{197}Au is 8.0711 MeV. Thus, in the presence of high-energy photons, ^{196}Au is produced. The ^{196}Au decay scheme is also well known and easily distinguished for counting by gamma-ray spectroscopy. The ^{197}Au(γ,n)^{196}Au cross section has been measured experimentally [69,78,89,90] and is believed accurate to within 25 percent.

While not directly a measure of the neutron production, simulation of the ^{196}Au production provides a secondary check of the photon production and transport.

Activation of the available gold samples was conducted on Sunday afternoon, April 18, 1999. Table 5-1 provides the number, mass and position of each sample. Due to the small mass of some of the foils, sets were combined into one sample such that the mass of each foil sample was about the same. Each sample was positioned to look at a different aspect of the neutron population. The accelerator was set to the high-energy photon mode with a 10x10 cm field size at 100 cm SSD throughout these experiments.

The first sample, I1, was placed bare at isocenter. It was situated on top of a cardboard box such that the long axis of the ingot was in the cross-plane direction. This was further supported by the treatment couch. The exact positioning was checked by the lighting system . The cardboard box provided separation from the treatment couch to try and lesson any effect it may have on the neutron population. The primary objective was to observe the total neutron flux seen at isocenter including the room return. The secondary objective was to observe the high-energy photon flux at isocenter.

The second sample, I2, was placed at isocenter surrounded by a moderator. A-150 plastic was chosen as the moderating material as it was readily available and easier to set up than a water tank. The plastic slabs are uniform in area, 30x30 cm, and of varying thickness. A 16 cm tall block was constructed on top of the treatment couch such that the center of the block was situated at isocenter. The sample was aligned such that the long axis of the ingot was in the cross-plane direction. The position was checked by the lighting system. The dimensions of the block were checked by ruler. The primary objective was to observe the effect of the moderator on the neutron population. The secondary objective was to observe the high-energy photon flux at isocenter.

The third sample, I3, was placed in the maze corridor. It was taped in place on the inside, i.e. nearest the room, wall such that it was 240 cm from the corner to the room and 150 cm above the floor. The long axis of the ingot was parallel to the maze corridor. The position was checked by ruler. The primary objective was to observe the neutron population in the maze. The secondary objective was to observe the high-energy photon flux from other sources within the accelerator. One possible secondary source of high-energy photons is that the energy selection slit within the Phillips bending magnet system. It constitutes a possible source of background contaminate photons and thus neutrons. The sample was aligned near the in-plane axis to check for this effect.

The remaining samples, foil samples 1-6, were placed in the moderator block similar to sample I2. They were distributed radially outward from isocenter along the cross-plane axis with the spacing indicated in Table 5-1. The position was checked by the light system and the spacing by ruler. The primary objective was to observe the neutron population in the moderator as a function of depth into the block. The secondary objective was to observe the in-beam versus out-beam high-energy photon flux.

The irradiation was carried out in 500 monitor unit (MU) increments. (The monitor unit is measured by an ion chamber within the accelerator and 1 MU nominally corresponds to 1 cGy absorbed dose at isocenter.) The start time and the total irradiation for each sample are also listed in Table 5-1. Samples I1 and I2 were each irradiated for 3000 MU. Sample numbers 1 through 6 were irradiated with all the foils in place for a total of 6000 MU. Sample I3 was expected to receive the least activation and was therefore left in place throughout the duration of the entire irradiation process for a total of 12000 MU.

A standard EG&G high-purity germanium detector and associated equipment were available in the control room at the time of the irradiation. Sample I1 was checked at the 1000, 2000 and 3000 MU irradiation levels to determine its count rate for the gamma-rays of interest. It was determined that count rate was sufficient after 3000 MU irradiation. Sample I2 was run to match sample I1. The foil samples, because of their smaller size, were irradiated for twice as long.

Final counting of the activated samples was performed at the NAA laboratory. As discussed above, background counts were taken prior to irradiation. Final counting was performed once later the same day as the irradiation and a second time the following day to ensure no false readings were observed. The 333 and 356 keV decay lines from ^{196}Au and the 412 keV decay line from ^{198}Au provided accurate assessment of the activation due to each of these isotopes. This activity is used to compute production rate of the isotopes seen while the beam was energized. The value is compared to simulation calculations of the value in the comparison discussion below.

Two certified sources were also observed during the second set of counts. This set of counts was used to determine the efficiency of the detector system for the gamma-rays of interest in the counting geometry. It also ensured that the energy calibration of the gamma-ray detection system was accurate. The final results are discussed in the comparison below.

Table 5-1. ID, mass, position, start time and length of irradiation of the gold samples.

Sample #	Radial Position (cm)	Individual ID #	Mass (g)	Start Time EST	Irradiation (MU)
I1	0	I1	31.1	14:13	3000
I2	0	I2	31.1	14:34	3000
I3	Maze	I3	31.1	14:13	12000
1	0	1	0.0664	14:51	6000
2	3	2	0.0634		
3	6	3	0.0512		
4	9	set (4,10)	0.0676		
5	11.5	set (5,6,9)	0.0782		
6	14.5	set (7,8,11)	0.0790		
-	-	4	0.0514	-	-
-	-	5	0.0159	-	-
-	-	6	0.0466	-	-
-	-	7	0.0472	-	-
-	-	8	0.0158	-	-
-	-	9	0.0157	-	-
-	-	10	0.0162	-	-
-	-	11	0.0160	-	-

Simulation Setup

The two sets of experimental data require modeling at very different levels of detail. As discussed in the Background section, electron-photon dose calculations for a patient or equivalent phantom do not require modeling of anything outside the main beam path. On the other hand, the neutron problem is influenced by every object in the room and especially the exact details of the treatment head. Likewise, the way in which the transport is run and the goals of the output are also very different.

Before describing these models, the goals of each should be stated. There are two key unknowns in the descriptions of the MEA obtained for use here: the mean electron energy and the number of electrons incident on the target. The simulation of the depth dose has as its goal the validation of the beam path geometry with its associated materials and the determination of these two unknowns. The value of these two unknowns is part of the starting point for the second set of simulations. The second simulation attempts to validate the neutron production and transport by matching the activation seen in the gold samples.

Physical geometry. Simulation of the depth dose curves is an electron-photon transport problem to solve the energy deposition in the water phantom. As such, it requires an accurate description of the area directly around the beam path. It turns out that this bit of physical geometry is the most difficult item to obtain.

Trying to obtain the exact schematics of a MEA is like trying to extract a sore tooth from a man who hates dentists. Upon asking if the tooth is sore, he answers that it might be but it’s nothing for a dentist. Upon finally going to the dentist and learning it must come out, he makes the dentist promise he won’t pull it without his permission. When the dentist asks for permission, the patient hems and hahs and talks about how the rest of his teeth are fine and only with the greatest of reluctance gives up anything at all.

When the author first asked for blueprints of the Phillips MEA in question, he was told that they existed but they were proprietary documents. This was understandable but unfortunate. At that time, the author felt is was important to remain outside of the obligations of handling proprietary data as it was desired to be able to publish full details of the model without restriction in this dissertation. This was considered an absolute must as without those details, the work presented cannot be replicated. Therefore, a compromise was reached and a hand-drawn schematic of the beam path as well as a generic diagram of the treatment head profile were made available.

During the course of this work, the author has had opportunity to speak with many researchers in this field. While many were willing to share experiences trying to model MEAs, all except one were unwilling to share exact details of their models because of obligations due to the proprietary nature of their source information. The author believes that this is a very unfortunate state of affairs as it means each group must start from scratch and no group can exactly replicate another's work as no two models will be exactly the same. It should also be noted that several persons with access to proprietary information made further statements to the effect that even with detailed blueprints, the specifications were sometimes out of date and key details had changed between the blueprints and the MEA as built.

The author owes a great debt to John Demarco and Indrin Chetty of the University of California at Los Angeles Department of Radiation Oncology. They were the one group willing to share their experiences using MCNP as a simulation tool [91] as well as their well validated models of the Phillips SL series MEA [92]. In terms of simulating electron-photon radiation transport in MEAs for calculating dose distributions, they have significantly advanced the state-of-the-art. For the purposes of this work, being able to start with their model meant no great effort was necessary to refine and validate the geometry model.

The geometry model thus obtained included the electron target, the target housing, the primary collimator, a hardening filter, two flattening filters and an ionization chamber. Dimensionally it matched the hand-drawn schematic obtained earlier with one exception. The exception was an aluminum ring just outside the second flattening filter. It was determined to be outside the main beam path and therefore ignored. The placement and size of the secondary collimators and the water tank were determined through discussions with the staff and engineers onsite at SCC.

The MCNP geometry specification was reordered and restructured during early trial runs without affecting the original specifications except to speed up calculations. MCNP geometry specifications can greatly influence run-times depending on the complexity per cell description. The reordering served the secondary purpose of describing the model in a more commented, structured manner for the sake of readability. During these same runs it was found that presence of the ion chamber made no significant contribution to the overall transport process and it was removed. A simple schematic of the final simulation geometry is shown in Figure 5-2 and the details for the MCNP input decks are given in Appendix E.

The second set of simulations has as its aim the accurate assessment of neutron production and distribution within the treatment room. As discussed in the background above, this is a considerably more difficult challenge than the depth dose simulations. In fact, as the discussion in the comparison will show, while the challenge has been met with more comprehensive techniques, the results still leave much to be desired. The following discussion will point out a number of approximations which have been made that contribute to the uncertainty in the final results.

The greatest single difficulty lies in adequately defining the physical space of consequence. The starting point used here is the treatment head as defined in the depth dose simulations. To this has been added the bare concrete walls of the treatment room. The dimensions and materials of the room were derived from the original architectural

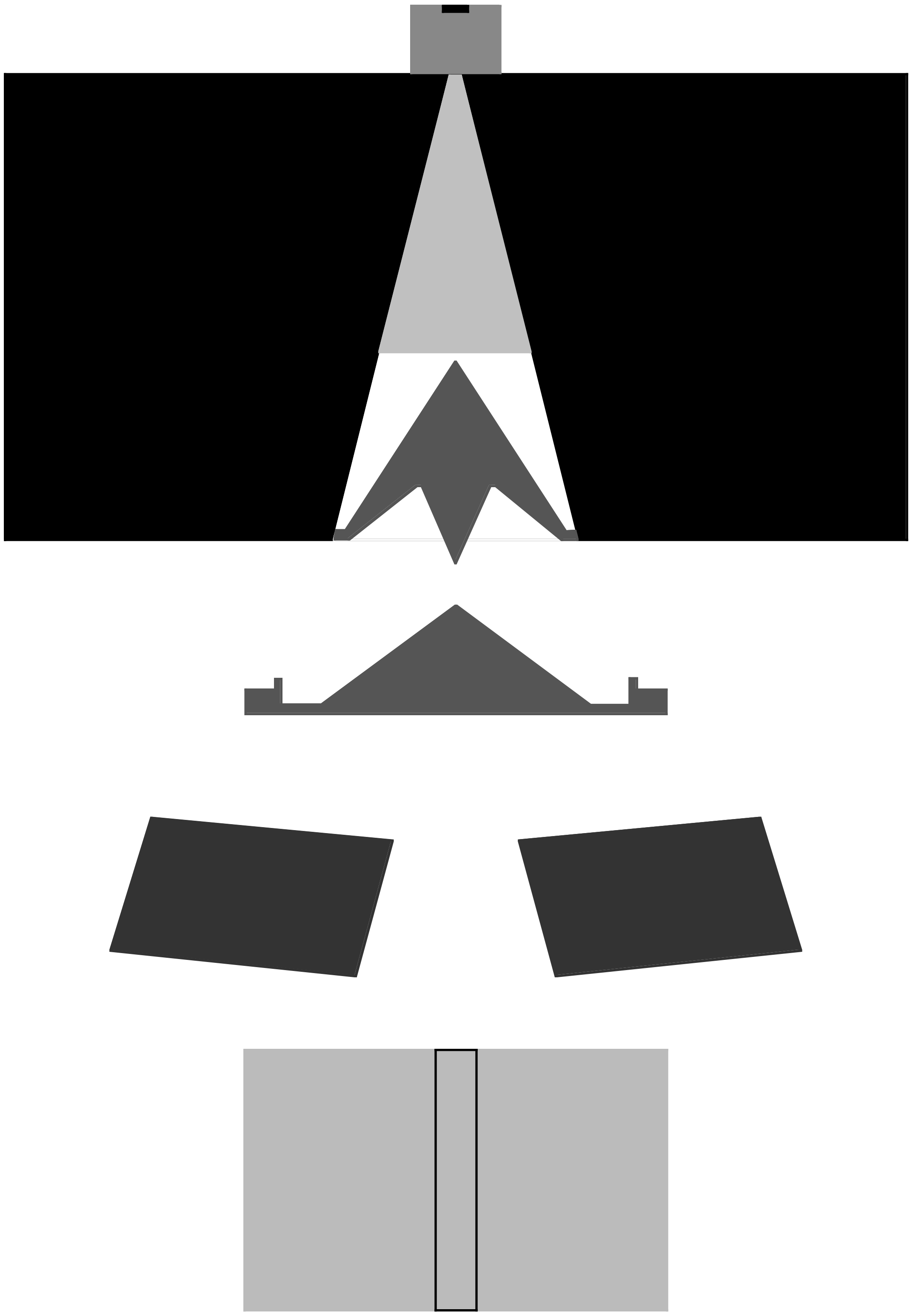

Figure 5-2. Simple schematic of the known geometry in the medical accelerator treatment head.

drawings and specifications [93] as obtained from the archives of the Shands Facility Management. For the primary simulations, nothing else was included.

It is worth spending some time on the known unknowns this approximation introduces and why it was made. First and foremost, only about one-third of the total material in the treatment head is represented. The remainder is significantly outside the primary beam path and therefore was not necessary for the electron-photon simulation and was not included in any of the available references.

A detailed representation of the lead and tungsten shielding, structural steel and other materials in the treatment head is necessary for an accurate simulation of the electron-photon-neutron problem. Without truly accurate descriptions of the locations and compositions of these materials, anything done is subject to large error. The final results presented below are obtained only with what is known. With that said, there are a significant number of variations and/or educated guesses on placement of lead and tungsten shielding that can be made to estimate the influence of this missing material.

It is recommended that future work undertake to obtain this information by direct inspection of the MEA. It might be attempted to obtain this information through blueprints or specifications but the final model should be matched against the actual dimensions as measured. It was not feasible to obtain that information for the purpose of the current study.

There is also a significant amount of material within the room that is not represented in the simulation. Again, this is done due to the difficulty in obtaining information on the placement and composition of the objects. It was not readily apparent from the specifications if the room was finished as is typical for similar facilities. If this is the case, there are aluminum or steel studs providing cable runs between the concrete and finished wall. The finished wall itself is probably gypsum board covered by paint and/or wallpaper. Cabinetry and furnishings have been provided to make the room a useful workspace. Additionally, the main bulk of the accelerator itself, the wall partitioning the machinery room and many other miscellaneous items also reside within the concrete vault.

This list of missing materials and unknowns could be continued though it certainly becomes less significant. However, of probable importance are certain construction materials and the accelerator itself. It is estimated [94] that one to two and a half tons of structural steel or aluminum reside in the walls; one and a half to two and a half tons of gypsum wall board, ceiling tiles or the equivalent coverings cover the walls and ceiling; and, several hundred pounds of cabinetry are located against the walls. The accelerator components, associated machinery and treatment couch account for hundreds, if not thousands, of pounds of additional metals. The partition wall forming the machinery closet is also left out. This lack of this material in the simulation represents an unknown error that could play a significant role in neutron scattering and absorption.

Starting from this simplified model, four variations were used to simulate the activation experimental setup. These are: the bare room; the room with detailed ingots at isocenter and in the maze; the room with the moderator block at isocenter; and, the room with the moderator block and detailed ingots at isocenter and in the maze. During the course of this study, several variations have also been explored though none have been included in the final simulations.

Transport data. Those materials in the treatment head which were specified in the simple drawing are the same as those specified in the UCLA model. The target is tungsten alloyed with 10 weight percent rhenium and has a density of 19.47 g/cc. The target housing is natural copper and has a density of 8.96 g/cc. The primary collimator is tungsten alloyed with 1.5 weight percent copper and 3.5 weight percent nickel and has a density of 18.78 g/cc. The hardening filter is natural aluminum and has a density of 2.7 g/cc. The flattening filters are both steel with a density of 7.9 g/cc. The composition was not listed on the hand drawing so the UCLA definition was taken. The steel is iron with 18 weight percent chromium, 9 weight percent nickel, 2 weight percent manganese and 1 weight percent silicon. The secondary collimators are lead. The lead may or may not be alloyed with antimony. For this model it was taken as natural lead and has a density of 11.35 g/cc.

For the transport simulation of the depth dose curves, electron and photoatomic tables were available for all of the elements specified above. Tables are also available for simulating the hydrogen and oxygen of the water tank. While available, the air was taken to be a void for these simulations. The detailed material descriptions used in the actual input decks are listed in Appendix E.

For the activation simulations, the concrete walls, air, gold and A-150 plastic have been added to the physical description. Other materials may be present in the room but are not included as they are not represented in the geometry. The definitions of those materials present and not established earlier have been taken from well established sources. Exact details of the material compositions used are once again found in Appendix E.

As with almost all MCNP simulations, tables are available for all elements of interest for electron and photoatomic interactions. For neutron interactions, tables are available for almost all isotopes and those unavailable constitute minor isotopes of natural elements (less than 5 percent in all cases; less than 1 percent in most). However, difficulty arises because only nine evaluated photonuclear tables are available for use: ^{27}Al, ^{40}Ca, ^{56}Fe, ^{63}Cu, ^{181}Ta, ^{184}W, ^{206,207,208}Pb.

The difficulty is deciding what is reasonable when no table exists for an isotope of interest. What has always been done in past is to completely ignore the photonuclear contributions. However, this virtually guarantees that the simulation will underpredict the neutron production. Therefore, it seems more reasonable to follow the example used in picking neutron tables and to use an available isotope, e.g. ^{184}W, to represent all the isotopes present in the natural element, e.g. ^{180,182,183,184,186}W. However, the reason these tables are compiled by isotope is that each individual species in a naturally occurring element has unique thresholds and reactions. Still, engineering practicality says that something is better than nothing. As a result, the original argument is extended to say that it is reasonable to use a table for an isotope of similar atomic weight if the necessary isotopic table is missing. However, beware, you get what you pay for.

For the purposes of representing the materials in this simulation, missing tables were substituted by isotope within an element or by nearest atomic neighbor. Specifically, the ^{184}W table was used to represent elemental tungsten and rhenium. The ^{63}Cu table was used to represent elemental copper and nickel. The ^{27}Al table represents elemental aluminum and was also used to represent elemental silicon. The ^{56}Fe table was used to represent elemental iron, chromium and mangenese. The ^{40}Ca table was used to represent elemental calcium. The major isotopes of lead, ^{206,207,208}Pb, have tables available leaving only 1.4 atom percent ^{204}Pb to be covered by an appropriate mixture. No photonuclear table was associated with hydrogen, carbon, nitrogen, oxygen, flourine, sodium, magnesium, sulphur or argon. The exact definitions of the materials as well as the geometry used in the simulation can be found in Appendix E.

Radiation source. The radiation source in the MEA is electrons on the target. The electrons are produced by an electron gun, formed into bunches and accelerated through a microwave chamber. They are then guided through a set of vacuum tubes to the treatment head. A system of three bending magnets [95] then spread the beam, direct it through an energy selection slit, refocus it and direct it onto the target. The actual electron distribution on the target is probably a chopped Gaussian in energy, impinging on a relatively small spot with a slight angular distribution about the normal. This distribution as modeled in the simulation is a whole Gaussian in energy having a full width at half maximum of 780 keV and perpendicularly incident on a spot size 1 mm in diameter. This description is taken directly from the original UCLA model and is believed accurate enough as it has been used to successfully match experimental data.

One of the key unknowns is the mean energy of the electrons incident on the target. It will always be dependent on the specific MEA in use as it is a function of the microwave cavity and RF tuning. The UCLA model gives this value as 22 MeV. Taking 22 MeV as a starting point, depth dose simulations were run at 1 MeV increments for 3 MeV on either side of this value, i.e. 19, 20, 21, 22, 23, 24 and 25 MeV. The details are listed in Appendix E.

The radiation source for the activation simulations is the exactly as described above. However, it is worth noting that while this is still a good approximation to the full source, it leaves out one portion which has the potential to be significant. As the electrons pass through the system of bending magnets, they pass through the energy selection slit. The bremsstrahlung photons occurring as a result of this process do not affect the electron-photon dose as a significant amount of shielding blocks their direct path to the treatment area. However, any high-energy photons from this process can contribute to the production of neutrons.

Transport algorithms. The MCNP radiation transport code is the work of hundreds of people over decades of time. One of the principal reasons the current work was performed using the MCNP code as a base was the comprehensive validation of its primary transport algorithms. This validation has been accomplished through the diligence and use of thousands of users. MCNP is one of, if not the, gold standard in neutron-photon transport and its electron-photon transport package has made great advances over the last decade. For neutron and photon transport, there are many papers in the literature validating the accuracy of the data combined with MCNP's transport algorithms.

Electron transport was added relatively recently in MCNP's lineage and is still undergoing significant improvements. However, the current set of electron transport algorithms are derived from the well established ITS code [96] and have proven to be accurate for most situations. Several papers have been published since electron transport was first added to MCNP showing its application to electron accelerator environments. Recent examples include work by Love et al. [97] and Jeraj et al. [98]. These show that while there is still work needed in this area, MCNP is capable of simulating this class of problems.

However, since it was desired to run the depth dose simulations with the best electron-photon physics transport package available, they have been run using the most up-to-date electron physics package. As part of the upcoming release of MCNP version 4C, Ken Adams of the MCNP code development team has worked to correct some of the known discrepancies in the electron transport algorithms [99]. A prototype code has been designated MCNP4BNU to indicate it is based on MCNP4B2 and includes the new electron package. The prototype was made available to the author for use in these simulations [100].

The bulk of the current work has been directed at providing the algorithms necessary to include photonuclear interactions in MCNP. The resultant prototype code is designated MCNP4BPN to indicate that it is based on MCNP4B2 and includes photonuclear physics. It has been well documented in Chapter 3 and validated in Chapter 4. The neutron, photoatomic and electron routines remain those of MCNP4B and as such have been validated as previously described. MCNP4BPN is used for the bulk of the activation simulations.

Obtaining Output. The MCNP code has a very well established, comprehensive set of output tables. The have been well tested over the years by the users and present a wealth of information about the simulation. Creation and loss tables present a summary of the overall events. If needed, details are available about the events by cell and by the type of interaction. And most important of all, a standardized tally package provides requested results along with statistical analysis of their uncertainty.

Part of the problem in simulating the depth dose calculations is doing detailed electron-photon transport over more than a meter in distance. It is trivial to obtain the bremsstrahlung spectrum from the target and transport it into the region of interest, i.e. the water tank. It is more slightly more difficult to obtain the bremsstrahlung spectrum from the other major components in the beam path and transport them to the region of interest. It is extremely difficult to transport the scattered electrons to the region of interest. Because of this, the typical approach has been to break the geometry up into distinct regions and create a phase-space file which adequately describes the electron-photon "source" at the start of each new location.

In the true spirit of engineering mentality, i.e. always use the biggest hammer available, these simulations were run from the original electron source incident on the target. This was done at great expense in terms of CPU time though it considerably eased the amount of time that would have been necessary to understand and use phase-space files. The generation and use of phase-space files is, in the opinion of the author, still an art form rather than a science. Therefore a large number of CPU cycles were facilitated by the availability of standardized variance reduction techniques within MCNP to run these simulations from top to bottom.

A dxtran sphere allows a volume of interest to receive a representative neutral particle from every collision site outside of the volume. In order to keep the Monte Carlo game fair, any particle actually reaching the boundary of the dxtran sphere is killed. For the purpose of these simulations, a photon dxtran sphere was placed around the water tank. To illustrate how effective this method is, consider that a typical example simulation shows that 8 million photons were killed at the dxtran boundary but 200 million particles entered the sphere, a significant net gain of particles interacting with the water tank.

Unfortunately, there is no method currently available to propagate electron contributions over a distance. Therefore the electron transport had to proceed the traditional way. Each electron reaching the water tank was the result of a long series of collisions which managed to penetrate the full length of the treatment head and still be going in the right direction at the bottom. Part of the requirement for long run-times derives from the need to have enough of these particles contribute to the dose very near the surface of the water tank.

The dose along the central axis of the water tank was calculated from the surface to 30 cm. A three square centimeter column was defined extending along the central axis. It was cut up into 39 vertical slices: the first five each 0.2 cm thick, the next nine each 0.5 cm thick, the next 24 each 1 cm thick and the last cell 0.5 cm thick. The energy deposited in each cell can be tallied and, with that, the absorbed dose calculated.

The standard MCNP tally package includes two methods for estimating energy deposition. The first is a heating tally. This method computes the average energy deposited in the volume of interest for each photon collision assuming all secondary energy is deposited instantaneously and locally. This requires that the region in question has reach electron equilibrium. This occurs for homogeneous regions away from boundaries. Thus it is a reasonable approximation for those points after the build-up region and peak dose. As this method depends primarily on the photon transport, it does not require as much time to achieve a converged answer as the next method.

The second method to estimate energy deposition in a volume is to measure the net energy flow through its boundaries. This is achieved simply by tracking each particle and adding its energy to the tally when it enters and subtracting off its energy when it leaves. The primary energy loss mechanism is the slowing down of electrons within the cell. Obtaining convergence for this energy deposition tally is difficult because it is necessary to have a large number of particles, particularly electrons, traverse the volume in order to obtain an accurate measure of the average energy deposited.

The energy deposition tally is also sensitive to the electron and photon energy cutoffs. Because the energy of the particle is added to the cell when it enters, any mechanism that prevents it from leaving will cause the energy to remain added to the tally. Thus if the electron and photon energy cutoffs, the energy below which no further transport is done, are too high, the tally will probably overestimate the absorbed dose. It was found through preliminary simulations that electron and photon cutoffs of 0.5 and 0.1 MeV, respectively, gave reasonable answers in acceptable run times.

All standard MCNP tally outputs include a wealth of statistical information to help the user determine the precision of the answer. These simulations were run until the energy deposition tally showed convergence at less than 3 percent relative error. The heating tallies could be run at the same time thereby making better use of the time spent. The run-times needed to achieve convergence in the energy deposition tallies corresponded to a relative error level in the heating tallies of less than 0.5 percent. However, it should be remembered that these both these error levels indicate the precision of the Monte Carlo results and not necessarily its true accuracy. There accuracy will be discussed in detail in the comparison sub-section.

Results were obtained in the manner described above for three field sizes. The secondary collimators were set such that the photon field incident on the water tank 100 cm SSD was 5x5 cm, 10x10 cm and 30x30 cm, respectively. Each of the seven incident energy distributions was considered. All other conditions were held constant such that 21 variations were run.

A debt of gratitude is owed to the Advanced Computing Laboratory (ACL) at Los Alamos National Laboratory (LANL). They operate the world's fastest integrated computer (at least for today), the Blue Mountain SGI cluster [101]. At the time these simulations were performed, the machine was severely underutilized. After about 4,000 hours of time to obtain some preliminary results, the final set of simulations took 35,000 hours of CPU time. However, this time was cheap in comparison to the learning curve necessary to understand and utilize phase-space files.

Two standard MCNP tallies were used to obtain estimates of the activation in the gold foil. The track length estimate evaluates the particle flux in a volume. A point detector evaluates particle flux at a point. Either can be multiplied as a function of energy with a production cross section and an atomic density to obtain the production rate of an isotope per source electron per volume. Track length estimators were used to evaluate the isotopic production rate in the ingots as measured over finite volumes. Point detectors were used to evaluate the isotopic production rate in the foils as approximated at a point. Again, the statistical analysis package provides useful information for evaluating the precision of the results.

Several techniques were used by the activation simulations to reduce the CPU time required. A dxtran sphere, as discussed above, was used to surround the volume where a track length estimate of particle production was made. Since electrons and photons with an energy below the lowest photonuclear threshold are no longer capable of producing neutrons they are not transported. This is done by setting the particle energy cutoff to remove them from the simulation when they fall below 5.7 MeV, the lowest photonuclear threshold in the simulation. The electron energy cutoff represents the most substantial time savings as electron transport becomes much more CPU intensive at lower energies. Photonuclear biasing was used such that the neutron production from every photon collision would be evaluated. Finally, a single weight window was used for each particle type to ensure that particle weight due to the biasing schemes did not cause unnecessary fluctuations in the tallies.

It should be noted that dxtran spheres are not used in conjunction with point detectors. Point detectors are also known as next event estimators. They work in a manner analogous to dxtran spheres. A contribution is made to the flux at the point detector from every particle collision. Therefore, dxtran spheres are used in those simulations with a finite ingot defined and point detectors are used otherwise.

The activation simulations included the production rates of both ^{198}Au and ^{196}Au. The (n,γ) cross section necessary for estimating production of ^{198}Au was available in the ZAID 79197.60c data set found in the standard ENDF60 continuous-energy neutron library [102]. The (γ,n) cross section necessary for estimating production of ^{196}Au was taken from the Saclay ^{197}Au photoneutron cross-section evaluation [69] as available electronically in the Atlas of Photonuclear Cross Sections [17].

There were five simulation setups of interest: the bare ingot at isocenter; the moderated ingot at isocenter; the ingot located in the maze; the foils in moderator; and, the foils without the moderator. The point detector estimate used for the foils without the moderator is useful for comparison purposes even though the equivalent experiment was not performed. Considering the seven energy distributions, this lead to 35 simulations.

Each of these simulations was duplicated, for ^{196}Au production only, using MCNP4BNU to determine if the enhanced electron physics would significantly change the production rate. The simulation thus run is not quite identical to the MCNP4BPN simulation in that MCNP4BNU does not include the photonuclear cross section. Inclusion of the photonuclear cross section will shorten the photon mean free path slightly but due to the relatively small change, it has very little effect on the gross photon transport.

As with the depth dose simulations, it was desired to run until the statistical package reported convergence for all the tallies. This was indeed the case for almost all of the volume tallies. The few discrepancies involved warnings although other indications showed that the tally had converged. The point detectors had a more difficult time.

Point detectors are best used in a vacuum outside the main region of transport. Because they are next event estimators, most contributions to them tend to be of low weight as the particle has had to manage to scatter in the right direction and traverse a significant amount of material. Unfortunately, when used in a material they suffer from the occasional particle which collides relatively close by and has a high probability of scattering in the direction of the detector. These particles, to use a technical term, clobber the tally by introducing a sample with much higher weight than the average.

The activation simulations used point detectors residing within materials, in some cases within dense materials. As such, many of the these tallies did not converge due to the problem described above. After many attempts to remedy the situation and long run-times to see if enough normal particles could overcome the occasional offending particle, the final results were taken despite some continuing problems. User judgement is used to evaluate those results which did not converge and estimate their uncertainties in light of those results which were converged.

As discussed above, the author holds with the notion that all credible scientific work must be reproducible. In order to facilitate the reproduction of the data presented here, Appendix E contains the information necessary to reconstruct the input decks. The coding for the algorithms added to MCNP4B2 has also been provided as well as the coding necessary to reproduce the cross-section library used. The cross-section data are available in the ENDF format from the LANL T-2 web site [103]. The only set of information which would have been useful to include, but is not, is the actual MCNP output files. Unfortunately, they form several hundred megabytes worth of text files and their inclusion was not practical.

Discussion of the Results

Two sets of experimental data and simulation results have been described. This section will discuss how well the simulations match the experimental data and make suggestions about where to concentrate future work to improve these results. The comparison of the depth dose data is presented first, followed by the activation data.

Depth dose. The experimental data exists as three relative depth dose curves for photon field sizes of 5x5, 10x10 and 30x30 cm at 100 cm SSD. The original experimental data is presented in Figure 5-3. It was provided at 1 mm intervals without an estimate of the error bars though they are expected to be small. The data was averaged over each cell in order to facilitate comparison to the simulation results.

The heating tally results for each of the seven incident electron energies and for each of the three field sizes are shown in Figure 5-4 compared to their respective ion trace. Since this tally is only valid once electron equilibrium is achieved, the comparison is only for those points after and including the peak value of the ion trace. The sum of the squares of the difference between each tally value considered to the average of the ion trace in the cell volume was then minimized. Remember that the ion trace is a set of relative values. The general conclusion based on this graph is that the simulations appear to be in the right neighborhood. However, despite the fact that they meet the "eyeball norm", no further conclusions can be readily made strictly from this graph.

Three difference plots between each heating tally result and the ion trace are presented in Figure 5-5. This set of graphs provides much better insight into the true measure of each simulation. The first conclusion to be drawn is that the simulation model has fairly accurately modeled the true experiment. None of the results are more than four percent different. However, there are still some discrepancies.

Figures 5-4 and 5-5 are presented without error bars. Error bars are not included on these graphs as they would obscure the information being conveyed. The error bars are not available for ion chamber trace. The error bars for the heating tallies are all less than 0.3 percent of the tally value.

The first area of concern is the large slope in the difference plots just after the peak value. This is most probably due to electron equilibrium not having been fully

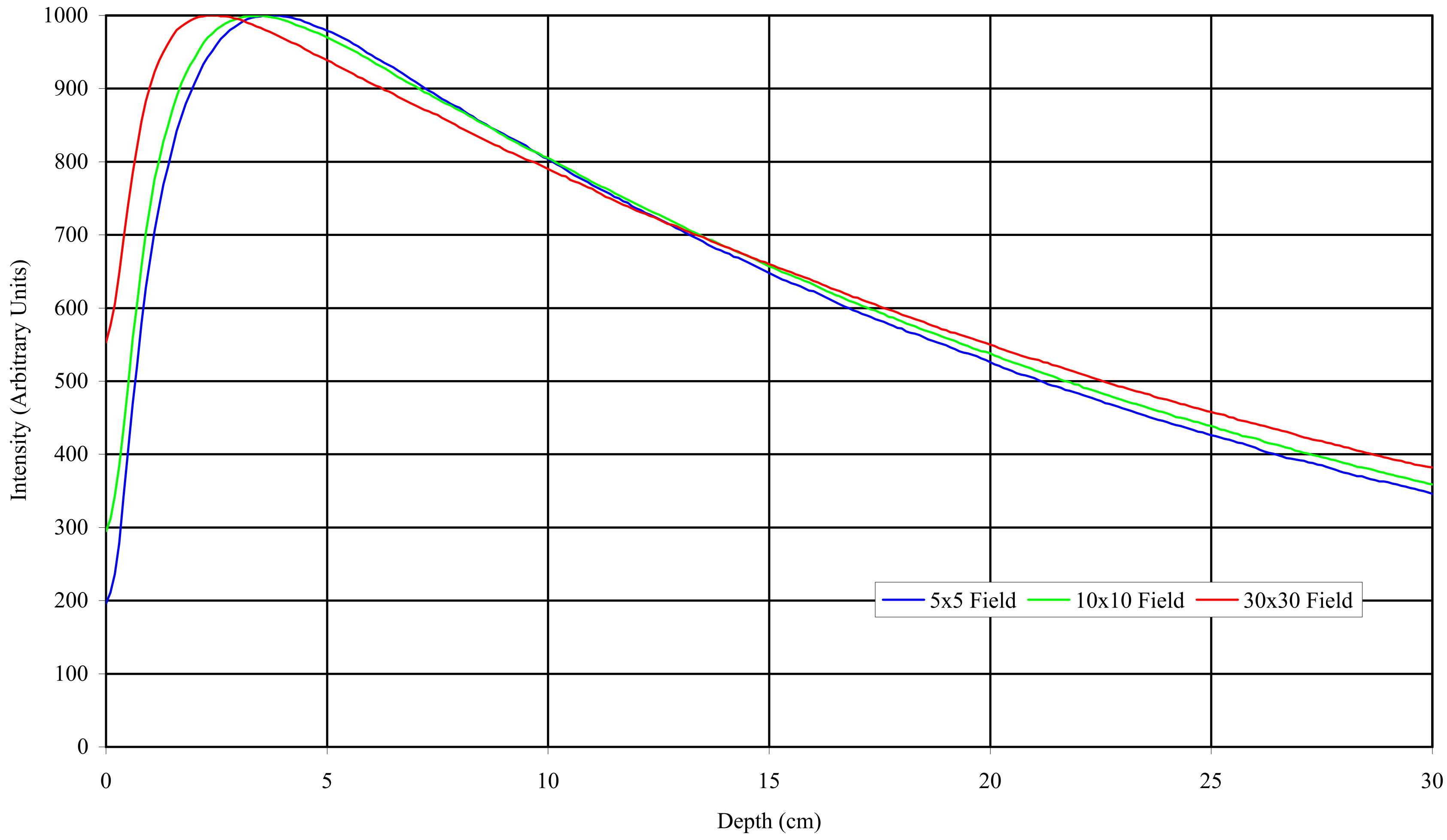

Figure 5-3. Ion chamber trace plots from a standard calibration of the Phillips SL25 in Room 5 of Shands Cancer Center.

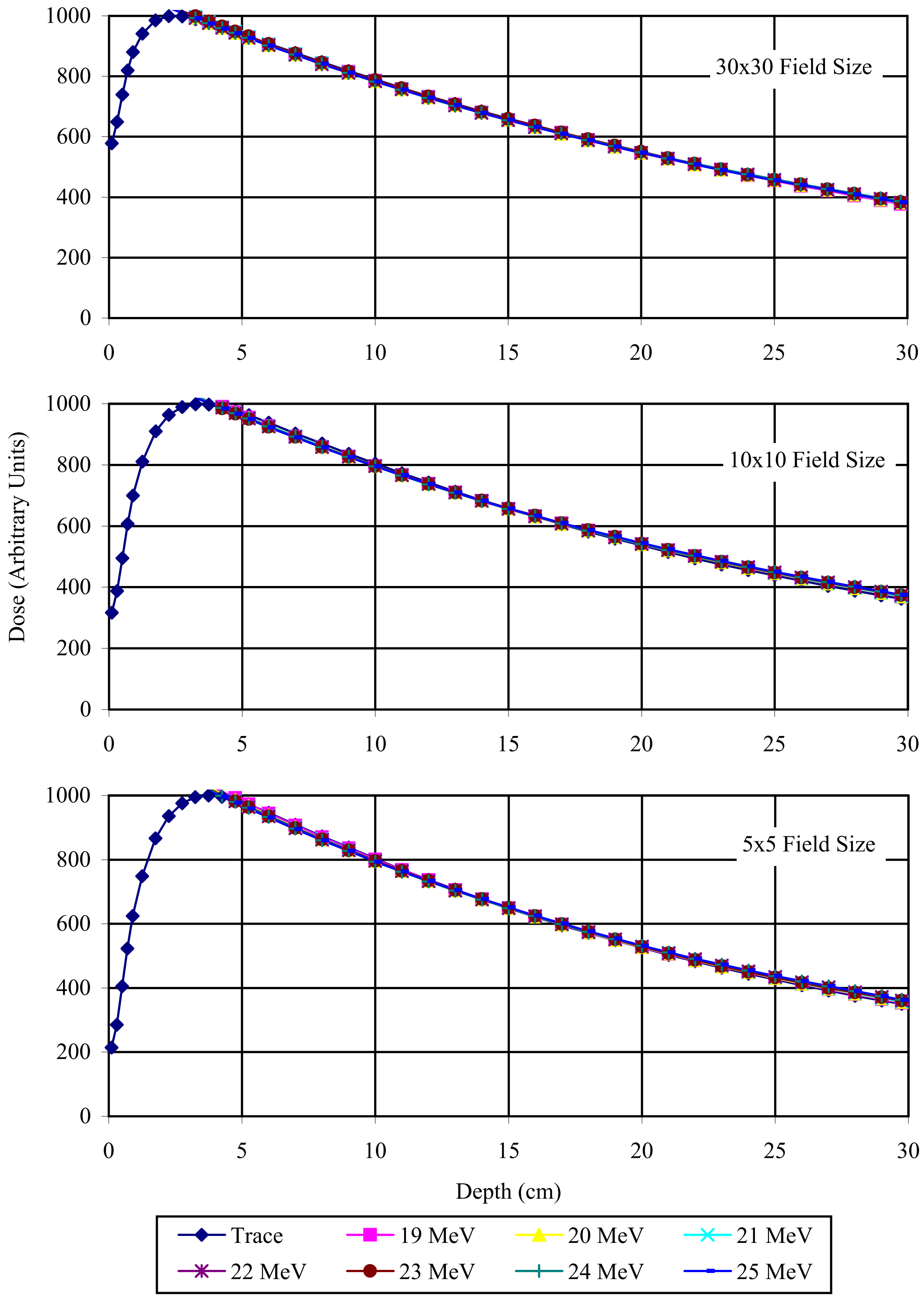

Figure 5-4. Comparison of ion chamber trace with calculated heating tally for a) a 30x30 field; b) a 10x10 field; and c) a 5x5 field.

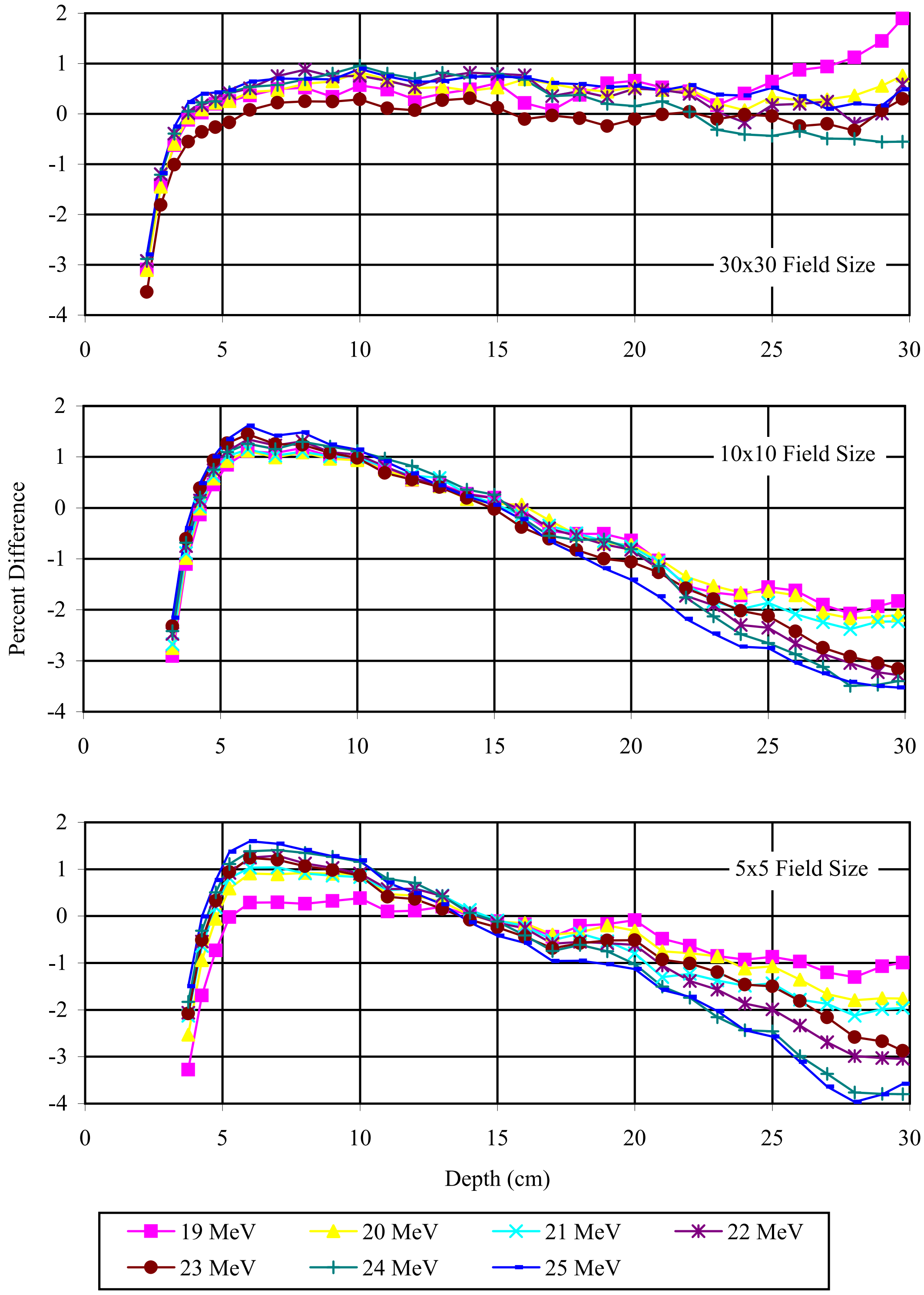

Figure 5-5. Percent differences between ion chamber trace and calculated heating tally for a) a 30x30 field; b) a 10x10 field; and c) a 5x5 field. (Percent difference is computed as (trace-calculation)/trace.)

reached until just after the peak value. Remember that the heating tally is not valid near boundaries where electron populations are in a non-equilibrium state.

The second area of concern is the increasing difference seen between the simulations and the ion trace for the 10x10 and 5x5 cm field sizes. One hypothesis for this difference is that the relative size of the tally volume is a much larger percentage of the total photon field size as that field size is decreased. The area of the central column used for defining the tally is 3 cm^2. This corresponds to 1/3, 3 and 12 percent of the 30x30, 10x10 and 5x5 cm field sizes, respectively. It may be that the tally is being influenced by the edges of the field boundary. Further study is needed.

Overall, the simulation geometry seems to have captured the essence of the physical space within the beam path. Further, from the difference graphs, the energy of the incident electrons would appear to be in the 20 to 21 MeV range. This value is based on user judgement in evaluating the curves. If the 30x30 cm field size simulation is taken as the most accurate, it would appear that the incident electron energy is above 20 MeV. Granting that the 10x10 and 5x5 cm field size simulations are not as accurate they still indicate that the higher energies are becoming more divergent especially above 21 MeV.

The energy deposition tally results for each of the seven incident electron energies and for each of the three field sizes are shown in Figure 5-6 compared to their respective ion trace. Again a least squares fit was used to match each simulation result to the ion trace. Similar to the heating tally results, these graphs seem to indicate that the simulation geometry is a fairly accurate representation of the treatment head along the beam path. The discrepancy near the peak is discussed below.

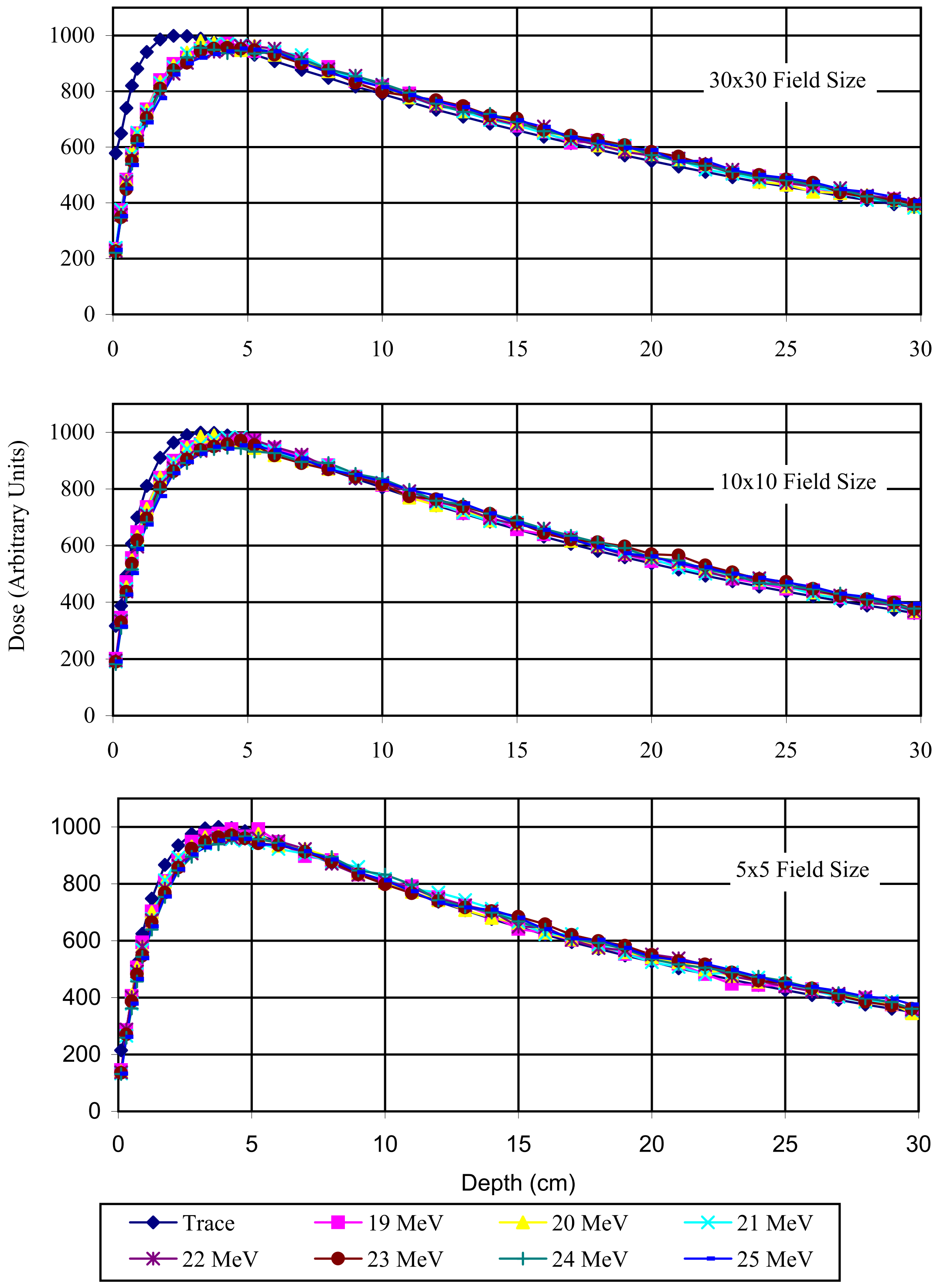

Figure 5-6. Comparison of ion chamber trace with calculated energy deposition for a) a 30x30 field; b) a 10x10 field; and c) a 5x5 field.

The percent differences between each energy deposition result and the ion trace are presented in Figure 5-7. Due to the larger relative error in these results, they do not provide as much insight as the heating tallies. Again no error bars are provided on the graphs themselves as they would obscure the information to be conveyed. The error bars for the simulation results are less than 3 percent. No error bars are available for the ion traces.

The first discrepancy causing concern is the increasingly poor match between the build-up region in the simulation versus the experiment. The build-up region is volume near the surface where the electron population has not reached equilibrium. The results for the simulation show discrepancies of approximately 30, 40 and 60 percent difference in the surface cell for the 5x5, 10x10 and 30x30 cm field sizes, respectively. This is most probably due to the lack of air in the simulation model. The air would provide a source of Compton scattered electrons impinging on the surface of the water tank. This explanation seems reasonable as the effect is worse for increasing field size where more electrons would be produced by this mechanism.

The area beyond the build-up region seems to substantiate the results of the heating tally. The overall agreement in this region is on the order of 5 percent or less. Though it is more difficult to observe, the same divergence seen in the heating tallies seems to be apparent here. Due to the larger relative errors, no conclusions about the appropriate incident electron energy can be drawn directly from these graphs though nothing seems to refute the conclusions drawn from the heating tallies.

The original goals of the depth dose simulation were to assess the incident electron energy and the number of electrons on target per MU. The mean energy has

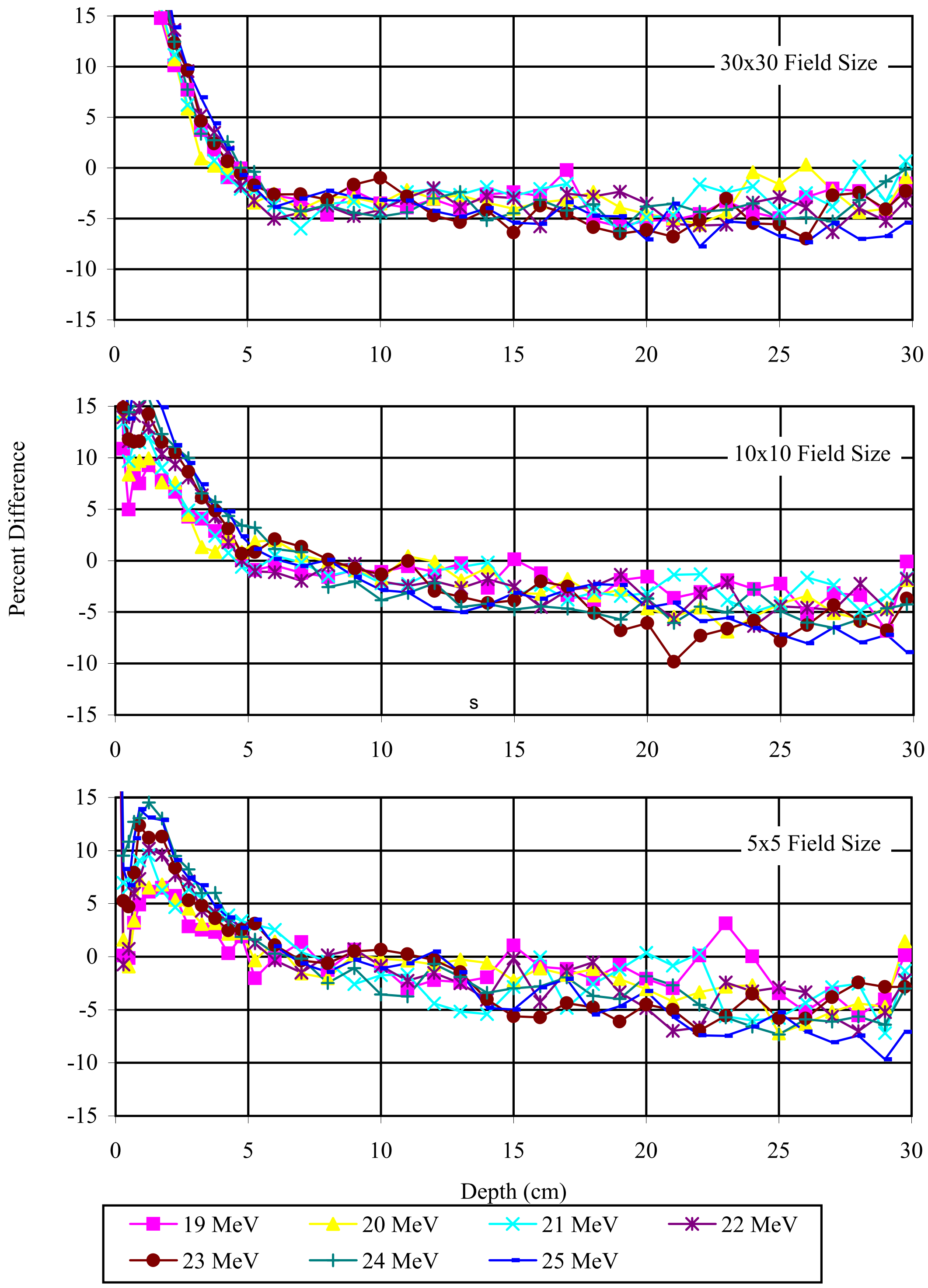

Figure 5-7. Percent differences between ion chamber trace and calculated energy deposition for a) a 30x30 field; b) a 10x10 field; and c) a 5x5 field. (Percent difference is computed as (trace-calculation)/trace.)

been estimated to be in the 20 to 21 MeV range. Remembering that one monitor unit corresponds to one centigray of absorbed dose at the peak of the depth dose curve for the 10x10 photon field size, Figure 5-8 shows the estimate of the number of electrons per MU as a function of energy for each of the two tally approximations. Based on the errors seen to this point, 10 percent error bars are included on these values. Taking the center of the expected energy range, the estimate of $(1.36\pm0.14)\cdot10^{13}$ electrons incident on the target per MU is predicted.

The overall conclusion drawn by the depth dose comparison is that the simulation's treatment of the physical space in the area of the beam path is substantially correct. Based on the results of simulations the mean incident energy is estimated to be (20.5 ± 0.5) MeV corresponding to $(1.36\pm0.14)\cdot10^{13}$ electrons on target per MU. Given an average dose rate of 400-425 MU per minute, this represents an average of approximately 15 microamps of beam current. This number is a good sanity check for the work so far as it makes physical sense.

Activation. The experimental data consists of an estimate for both ^{196}Au and ^{198}Au production for four different configurations. Several of these production rates have been simulated via two different methods. The final estimates of the production rate for both ^{196}Au and ^{198}Au by experiment and by simulation are given in Tables 5-2 and 5-3, respectively. How these numbers were calculated is the subject of the following discussion.

Each activation simulation reports the production rate in atoms produced per electron incident on the target per cubic centimeter of original atoms. The number of electrons per MU was estimated in the previous comparison. Thus the production rate

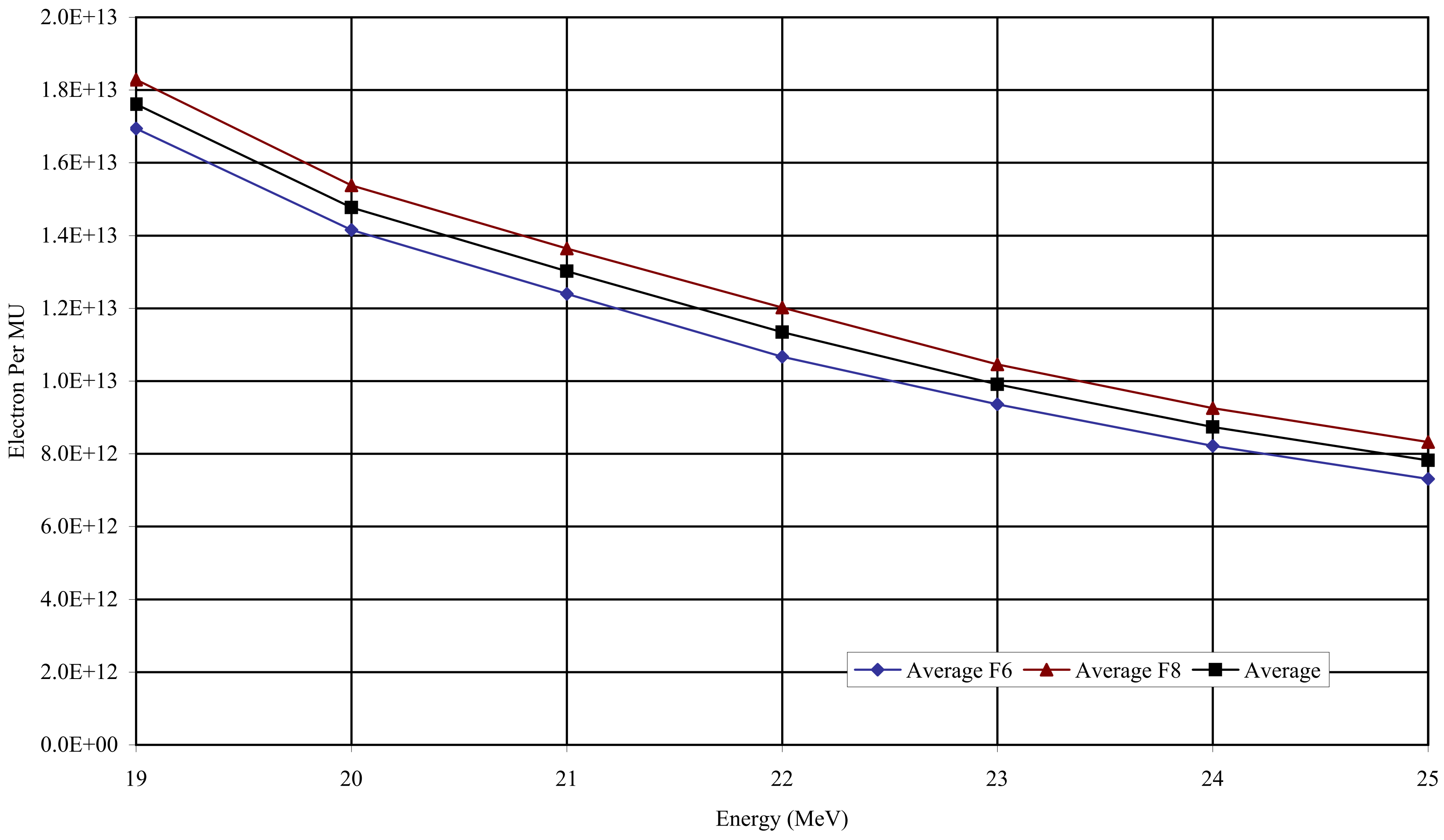

Figure 5-8. Estimate of electrons incident on target per MU (1 cGy) for the energy range of interest.

expressed in atoms produced per MU per cubic centimeter encapsulates the integral result of the simulations. For the purpose of comparison, it is desired to express the experimental data into this same format.

The raw data from the experiments consisted of several sets of counts for specific decays as determined by gamma-ray spectroscopy. The two strongest emission lines from ^{196}Au decay are a 333 keV gamma from 24.4 percent of the decays and a 355.72 keV gamma from 93.6 percent of the decays. The strongest emission line from ^{198}Au decay is a 411.8 keV gamma from 95.53 percent of the decays. Note that the decay data used throughout this section is taken from the Nuclide Navigator program [104]. The GammaVision software [105] used to control the counting provides an estimate of the net counts for each peak observed. All three of these peaks were present and well defined in each counting session. Other possible peaks of interest were not as well defined and therefore ignored.

Assuming the production rate is constant in time over the length of the irradiation, the number of counts seen is a function of that one value. Therefore, a system of equations can be written to express the count rate as a function of the production rate in atoms produced per MU per volume of sample atoms. Simple algebraic manipulation can be used to solve for the production rate in terms of the count rate and the count rate can be fed into these equations to estimate the production rate seen by each sample.

The number of counts seen by the detector can be calculated from Equation 5-1. Here, C is the number of counts observed and D is the true number of decays that occurred during the counting session. The dead time is accounted for by multiplying by the ratio of LT, the live time of the detection system, to RT, the real time elapsed during the counting session. BR is the branching ratio, i.e. the number of gamma-rays of a specified energy seen per decay. The times and branching ratios are assumed to have negligible error.

The last three factors in Equation 5-1 require more significant explanation. Because the dimensions of the ingots are significant in comparison to the mean free path of the gamma-rays, self shielding (SS) occurs. The correction factors used here have been computed by Monte Carlo simulation. Photons are produced uniformly within the volume of a mock ingot and the average number which escape the boundary is tallied. Given an ingot size of 4.1 x 2.4 x 0.1636 cm and a total mass of 31.1 g, the self-shielding factors are 0.445, 0.477 and 0.541 for 333, 355.72 and 411.8 keV photons, respectively. Although the Monte Carlo simulations for self shielding were run to convergence and negligible simulation error, the distribution of the photons in the ingot remains an unknown and the self-shielding factors are assigned a 10 percent uncertainty.

Self shielding in the foils was assumed to be negligible and assigned a factor of unity. Although self-shielding effects might be present, they should be minimal. An uncertainty of 5 percent should be assigned to this factor.

The detector efficiency (E_D) is a function of the gamma-ray energy and the position of the photon source in relation to the detector. Two certified check sources, ^{133}Ba and ^{137}Cs, were available to determine the absolute efficiency of the counting system. Four decay lines were of interest. The decay of ^{133}Ba includes 302.71, 355.86 and 383.7 keV gamma rays. The decay of ^{137}Cs includes a 661.62 keV gamma ray. Based on the counts rates observed from these four lines, detector efficiencies of 0.0504, 0.0484 and 0.0437 were used for the 333, 355.72 and 411.8 keV photons, respectively. The uncertainty in these efficiencies is estimated to be 10 percent.

At the time the counts were taken, it was not contemplated that the large finite size of the ingot samples would have an effect other than self-shielding. During the final analysis, it was discovered that the values estimated by the foils and the ingots differed significantly. The only reasonable suspect for such a difference was the much larger size of the ingots leading to a different detector efficiency than a point source. Therefore a finite size (FS) factor has been added to the equation.

$$C = D \cdot \frac{LT}{DT} \cdot E_D \cdot BR \cdot SS \cdot FS \tag{5-1}$$

Working back towards the production rate, the next step is to relate the true number of decays (D) to the number of atoms present at the start of the counting session (N_T). The fraction of atoms that survive the counting session is given by the familiar exponential decay term. T_{SC} and T_{EC} are the start and end time for the counting session, respectively. The decay constant λ is found in many sources and the values 1.30E-7 and 2.97E-7 per second are used for ^{196}Au and ^{198}Au, respectively. The relation describing the number of decays is written in full in Equation 5-2. The times and decay constants are assumed to have negligible error.

$$D = N_T \cdot \left(1 - e^{-l \cdot (T_{SC} - T_{EC})}\right) \tag{5-2}$$

The total number of atoms of interest in the sample at the beginning of the counting session is a function of the number of atoms produced in each irradiation and their subsequent decay. The total present at the start of the counting sessions is the sum of each of the individual contributions as shown in Equation 5-3. $N_{P,i}$ is the number produced during irradiation i. The exponential decay term accounts for the number that decayed between the end of the irradiation ($T_{E,i}$) and the start of the counting session(T_{SC}). Again, the times and decay constants are assumed to have negligible error.

$$N_T = \sum_i N_{P,i} \cdot e^{-l \cdot (T_{EP,i} - T_{SC})} \tag{5-3}$$

Finally, the number of atoms produced from the irradiation is a direct function of the production rate as shown in Equation 5-4. The production rate is given in units of atoms produced per volume(V) of sample per dose(AD_i) as measured in MU. The rate is in terms of MU, as opposed to electrons, as that is the experimental measure of the amount of irradiation. The decay of atoms during the irradiation time is ignored. This is justified by the fact that less than one-quarter of one percent of the atoms produced during any given irradiation decay before the end of the irradiation. The volume and the dose are assumed to have negligible error.

$$N_{P,i} = P \cdot V \cdot AD_i \tag{5-4}$$

The linear system of equations relating the production rate to the count rate has now been established. It is a trivial matter to solve for the production rate in terms of the count rate. The uncertainty of the count rates varies by gamma-ray and by counting session but on the whole is less than one percent for the ingots and a few percent for the foils. The issue of the finite size factor has not been resolved and is left for further discussion below.

The five configurations for the simulations have been described above. The final results are production rates as a function of incident electron energy given in units of atoms produced per electron per volume of sample. These are changed to units of atoms produced per MU per volume by multiplying the number of electrons per MU as estimated in the previous set of simulations. The uncertainty in the number of electrons per MU is estimated to be 10 percent. The uncertainty in the simulations is discussed in more detail below.

The first experimental configuration simulated was the bare ingot at isocenter. Simulations were run to calculate production of ^{198}Au using MCNP4BPN and production of ^{196}Au using MCNP4BPN and MCNP4BNU. The results of the simulations are shown in Figures 5-9 through 5-11, respectively.

For all of the simulations calculating the production rate of ^{196}Au using both MCNP4BPN and MCNP4BNU, almost no difference was seen. This can be attributed to the fact that the gross photon transport characteristics are unaffected by the presence or absence of the photonuclear portion of the photon cross section. It also indicates that the changes in electron transport and bremsstrahlung production from version 4B to 4BNU do not significantly affect this simulation. Therefore, while both sets of results will continue to be shown, their results are discussed without differentiating between the two.

Similar to the depth dose simulation, the production rate of ^{196}Au depends mainly on the electron-photon transport through the treatment head in the vicinity of the beam path. The point estimates of the production rate in the beam path indicate that the high-energy photon flux in this region is fairly uniform. The experimental values also confirm this. It is also evident from the simulations that self shielding in the ingots has an effect though in this case it changes the results by less than 10 percent. The best estimate of the production rate for ^{196}Au is $1.49 \cdot 10^{-7}$ atoms/e/cc from the volume estimate (the track

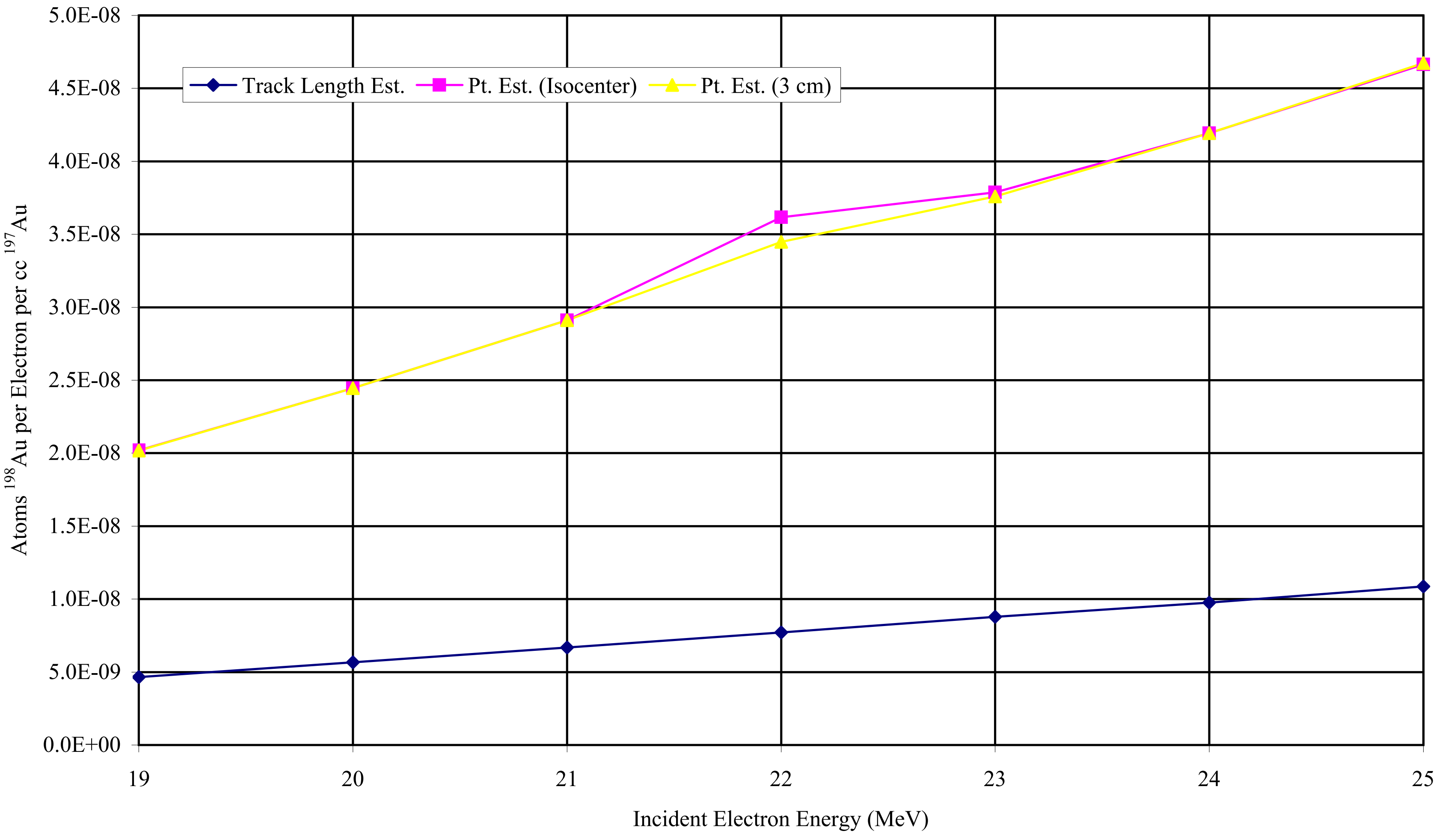

Figure 5-9. Calculated production rate of ^{198}Au in the ingot located at isocenter.

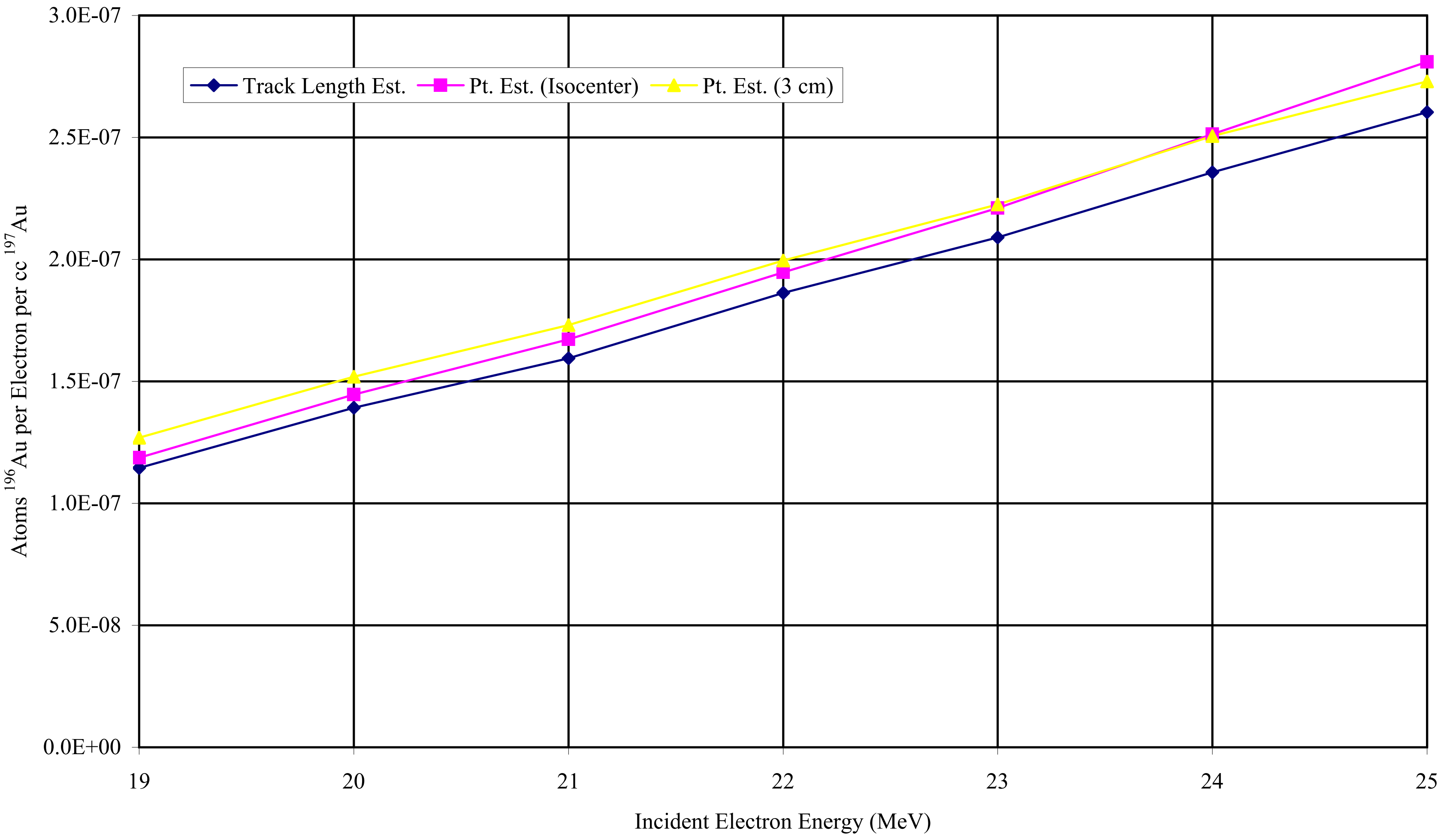

Figure 5-10. Calculated production rate of ^{196}Au in the ingot located at isocenter using MCNP4BPN.

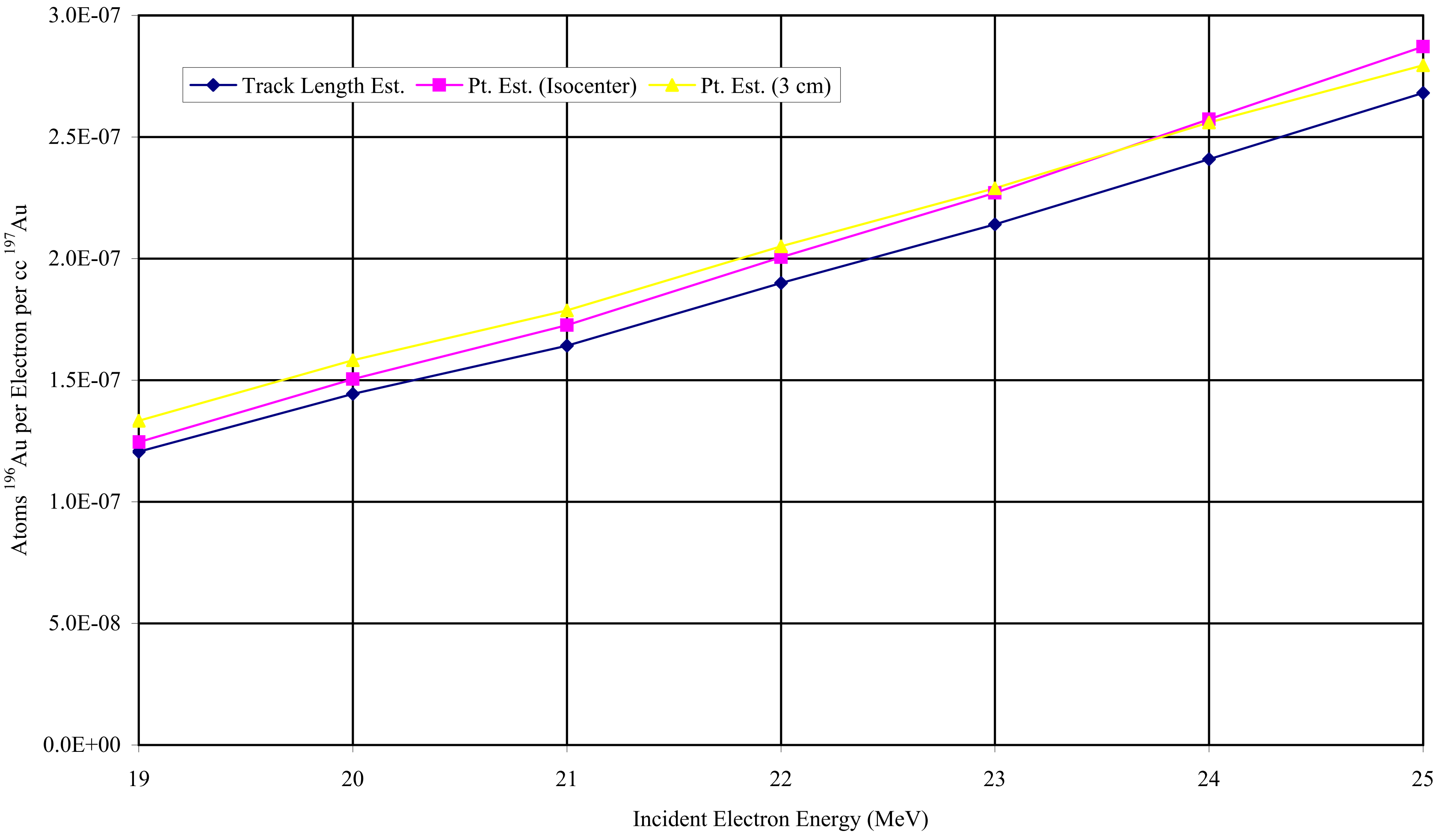

Figure 5-11. Calculated production rate of ^{196}Au in the ingot located at isocenter using MCNP4BNU.

length tally) and its uncertainty is estimated to be 25 percent. This uncertainty derives mainly from uncertainties in the $^{197}Au(\gamma,n)^{196}Au$ cross section.

The production rate of ^{198}Au is very sensitive to the production and transport of the neutrons. For the reasons discussed above, all the simulations attempting to simulate the production of ^{198}Au have a high level of uncertainty. It would be fair to say that these simulations represents an accurate solution of the problem described but that the problem described is incomplete. It is still worth discussing the results and drawing some conclusions.

For production of ^{198}Au in the bare ingot at isocenter, both the simulation using point estimates and the simulation using a volume estimator give reasonable results. However, these results are a factor of four different. This can be attributed to self shielding in the gold ingot causing less neutron flux to be seen within the larger volume. The volume estimate is clearly the best choice due to the self shielding in the sample. The production rate of ^{198}Au is estimated to be $6.18 \cdot 10^{-9}$ atoms/e/cc and the uncertainty is estimated to be a factor of three.

The second experimental configuration simulated was the moderated ingot at isocenter. Simulations were run to calculate production of ^{198}Au using MCNP4BPN and production of ^{196}Au using MCNP4BPN and MCNP4BNU. The results of the simulations are shown in Figures 5-12 through 5-14, respectively.

The presence of the moderator block reduces the high-energy photon flux available for production of ^{196}Au. However, the results are very similar to those obtained for the bare ingot. One difference is that the point detectors are subject to more high weight variations in this region and therefore more subject to error. Self shielding in the

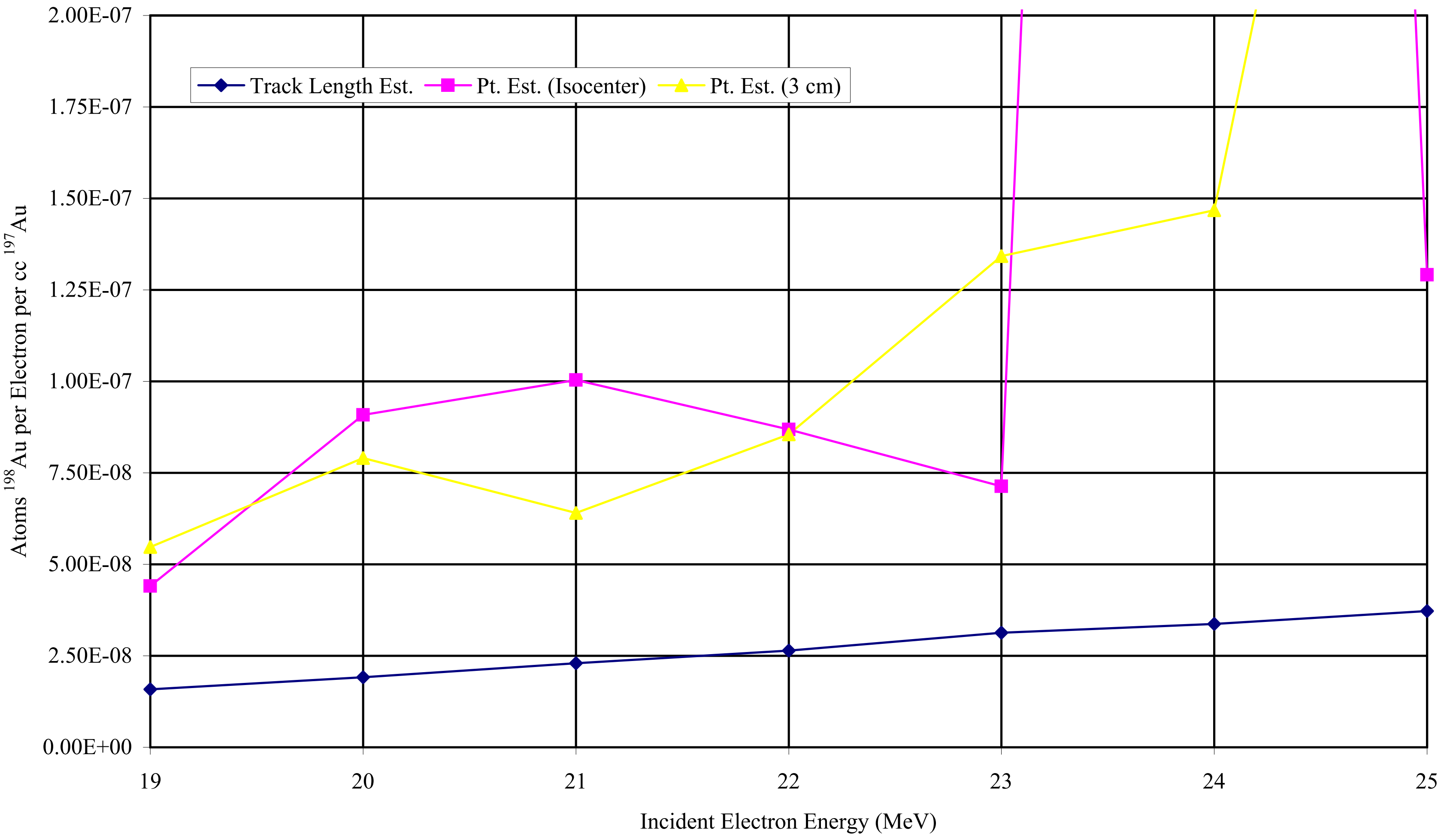

Figure 5-12. Calculated production rate of ^{198}Au in the ingot located at isocenter surrounded by A-150 plastic.

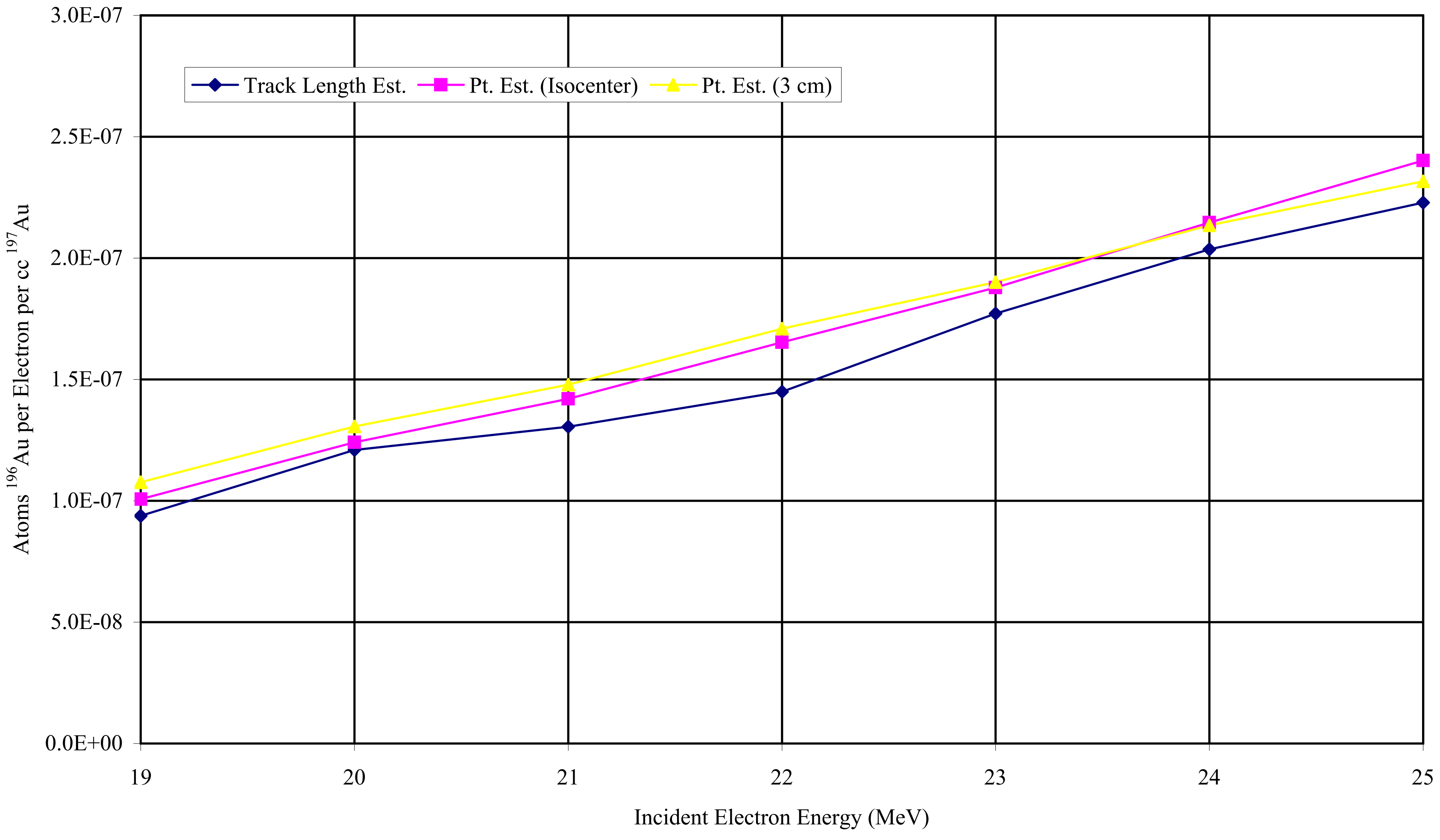

Figure 5-13. Calculated production rate of ^{196}Au in the ingot located at isocenter surrounded by A-150 plastic using MCNP4BPN.

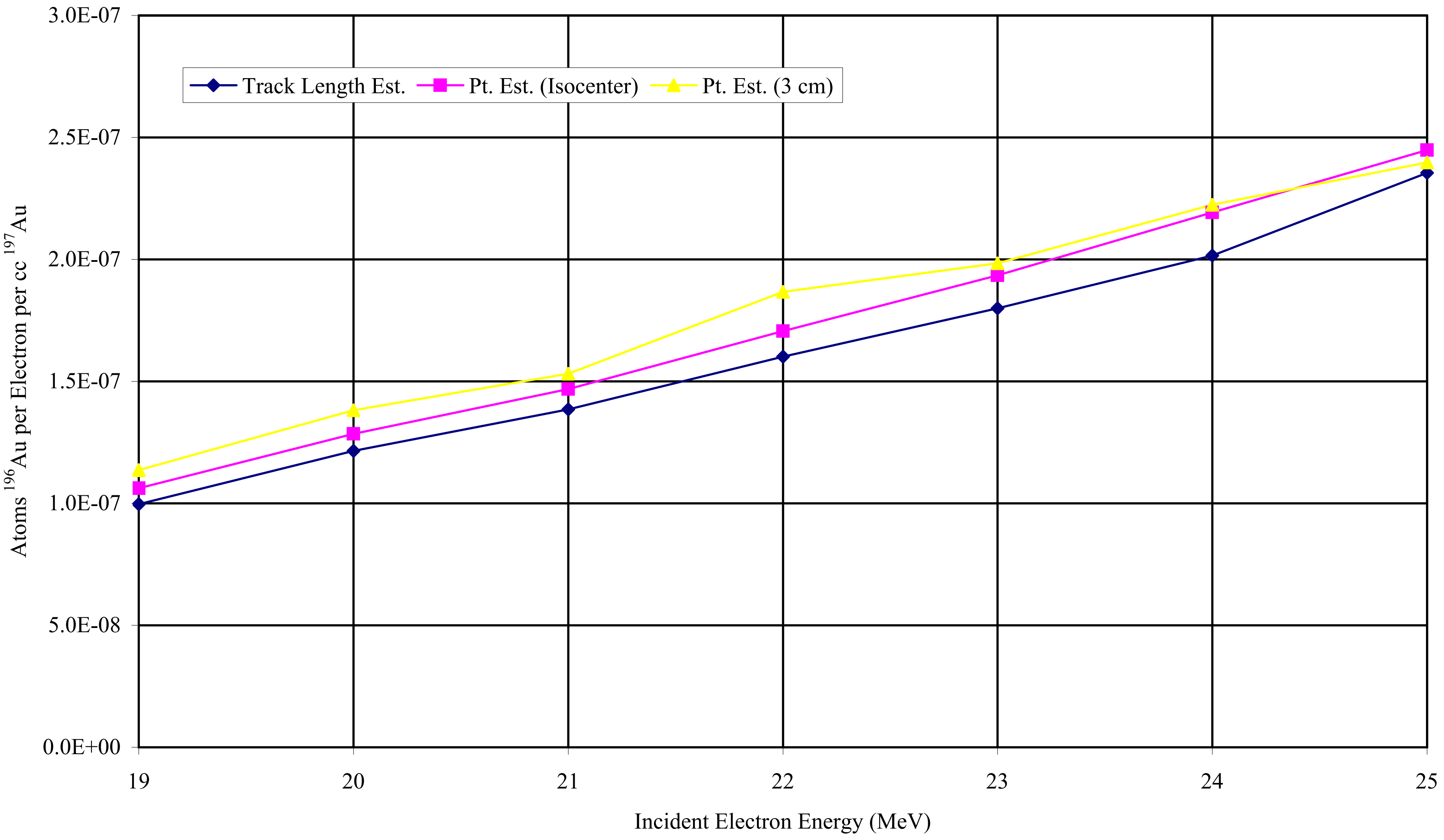

Figure 5-14. Calculated production rate of ^{196}Au in the ingot located at isocenter surrounded by A-150 plastic using MCNP4BNU.

sample volume still reduces the activation by something less than 10 percent. The volume estimate gives a value of $1.22 \cdot 10^{-7}$ atoms/e/cc for the ^{196}Au production rate and the estimated uncertain is about 25 percent.

The presence of the moderator block dramatically alters which neutrons are causing activation in the sample. In the bare ingot, thermal neutrons scattered within the room are readily available for absorption within the gold. The mean from path for the average neutron in the moderator block is about 0.5 cm. This means that any room scattered neutron must penetrate a minimum of 16 mean free paths to contribute to production of ^{198}Au. However, high-energy source neutrons have a longer mean free path and can more readily penetrate to the center of the moderator. In this same process, they are thermalized such that they are more easily absorbed in the gold. Again the volume estimate is used for the final results. The production rate of ^{198}Au is estimated to be $2.10 \cdot 10^{-8}$ atoms/e/cc and the uncertainty is estimated to be a factor of three.

The third experimental configuration simulated was the moderated foils distributed in the cross-plane. Simulations were run to calculate production of ^{198}Au using MCNP4BPN and production of ^{196}Au using MCNP4BPN and MCNP4BNU. The results for all the foils were obtained using point detectors. Unfortunately, point detectors are subject to large errors as described above. Therefore, these results did not prove as generally useful as those from the ingots. However, they did provide some insight into the modeling and, most importantly, insight into the probable value for the finite size factor.

The production rate of ^{196}Au in the foils drops dramatically outside the region of the photon field. In fact, due to the low exposure of those foils outside this region the experimental data is virtually meaningless. However, the two foils located in the primary beam path provide some useful data. The simulation results for these foils are presented in Figures 5-13 and 5-14. The point estimate at isocenter and 3 cm radially outward along the cross-plane show essentially the same result. These estimates give a value of $1.23 \cdot 10^{-7}$ atoms/e/cc for the ^{196}Au production rate and the estimated uncertain is about 25 percent.

It should be remembered that small, thin foils are typically used for activation analysis because their size closely resembles the calibration sources used to determine detector efficiency. Self shielding and finite size typically are not considered as significant factors in computing the activation in the sample based on the decays counted.

It can be shown that the foils used in this experiment do not appear to suffer from self shielding or finite size effects. Taking the self shielding and finite size factors to be unity for the foils, the experimental value for the ^{196}Au production rate in samples 1 and 2 is approximately $1.64 \cdot 10^{6}$ atoms/MU/cc with an uncertainty of 12 percent. The point detector estimation of the production rate is $1.36 \cdot 10^{-7}$ atoms/e/cc with an uncertainty of about 25 percent. Using the electron per MU value from above, this corresponds to a production rate of $1.85 \cdot 10^{6}$ atoms/MU/cc. This is a satisfactory match between the experimental and simulation values. It continues to enforce the conclusion that the electron-photon transport through the treatment head to the region around isocenter is accurately modeled.

Knowing that the simulation model appears to be accurately representing the electron-photon transport, the ^{196}Au production calculated can be taken as sufficiently near truth. The moderated foil and moderated ingot simulations give results that are within 10 percent of each other. This indicates that the experimental production rates should be within that same error margin. Taking self-shielding as computed above, the experimental value from the ingot is a factor of 1.6 too high when compared to the value from the foil. Based on the match between the foil estimate and the simulation, the simulation for the ingot should be substantially correct. Therefore a value of 1.6 is estimated for the finite size factor and used in calculating the experimental production rate from the ingot samples. As the estimate of the finite size value is based only on this one data point, it is assigned a 50 percent uncertainty. At a later date, this study should be redone using only foils in order to avoid having to use a value like this.

The production rate of ^{198}Au in the foils as estimated from the simulation is nearly worthless. The point detectors used for this estimate were subject to large fluctuations due to the problems described earlier. Using a healthy dose of user judgement, the production rate of ^{198}Au is estimated to be $8.36 \cdot 10^{-8}$ atoms/e/cc and the uncertainty is estimated to be a factor of three.

Though the remaining foils have been eliminated from the major comparison, they are worth considering for a moment longer. The experimental data show a trend in the moderator block that the ^{198}Au production is highest in the main beam path where high-energy neutrons can traverse the treatment head with little downscatter. Figure 5-15 shows that the simulation reproduces this trend though not as well defined as the experimental data.

The last experimental configuration simulated was the bare ingot located in the maze corridor. Simulations were run to calculate production of ^{198}Au using MCNP4BPN

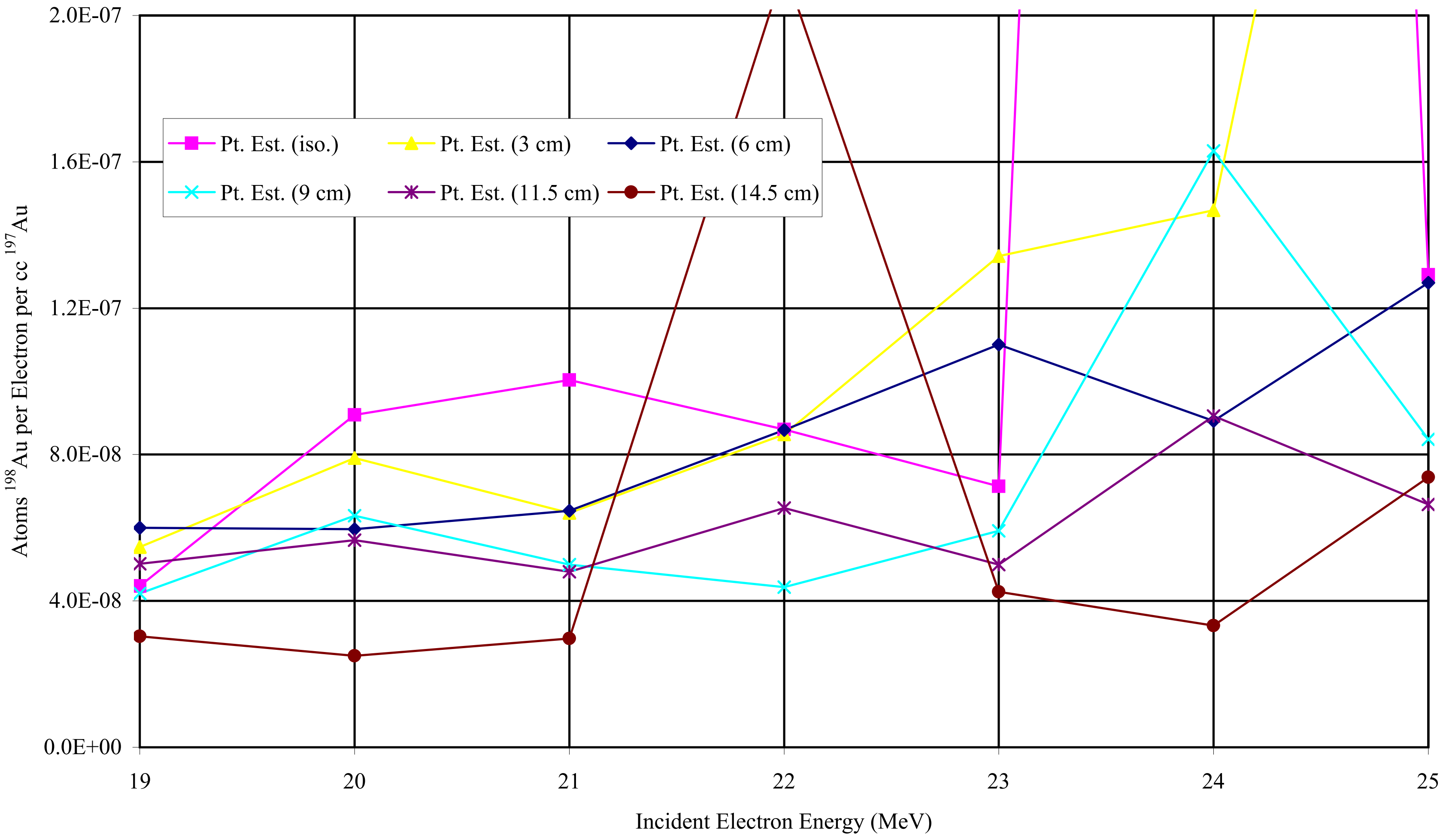

Figure 5-15. Calculated production rate of ^{198}Au in the foils distributed radially outward in the cross-plane direction from isocenter surrounded by the A-150 plastic.

and production of ^{196}Au using MCNP4BPN and MCNP4BNU. The results of the simulations are shown in Figures 5-16 through 5-18, respectively.

The production rate of ^{196}Au in the ingot located in the maze is anticipated to be very low. It is expected that the only significant contributions from the radiation source as modeled will be for electrons that have scattered such that they can produce bremsstrahlung heading towards the sample. The volume estimate gives a ^{196}Au production rate of $4.65 \cdot 10^{-13}$ atoms/e/cc and an estimated uncertain of 25 percent.

The production rate of ^{198}Au in the ingot located in the maze is anticipated to be significant. Neutrons scatter off concrete very well and the maze corridor presents an ideal streaming path. The volume estimate gives a ^{198}Au production rate of $7.83 \cdot 10^{-10}$ atoms/e/cc and the uncertainty is estimated to be a factor of three.

With all of the assumptions going into the experimental and simulation production rates now documented, the final values can be compared and conclusions drawn. Tables 5-2 and 5-3 present the final values for the production rates of ^{196}Au and ^{198}Au, respectively. The values are tabulated in units of atoms produced per monitor unit per cubic centimeter of gold sample. The ratio of the values is given to aid in comparison.

Table 5-2. Experimental and simulated production rates of ^{196}Au for four configurations.

Configuration	Production Rate (^{196}Au/MU/cc^{197}Au)		
	Experiment (E)	Simulation (S)	S/E
Sample I1 Bare Ingot at Isocenter	$(1.88 \pm 0.94) \cdot 10^6$	$(2.03 \pm 0.51) \cdot 10^6$	1.080
Sample I2 Moderated Ingot at Isocenter	$(1.63 \pm 0.82) \cdot 10^6$	$(1.71 \pm 0.43) \cdot 10^6$	1.049
Samples 1 & 2 Moderated Foil Near Isocenter	$(1.64 \pm 0.18) \cdot 10^6$	$(1.85 \pm 0.46) \cdot 10^6$	1.128
Sample I3 Bare Ingot in Maze	$(1.84 \pm 0.92) \cdot 10^4$	$(6.33 \pm 1.58) \cdot 10^0$	$3 \cdot 10^{-4}$

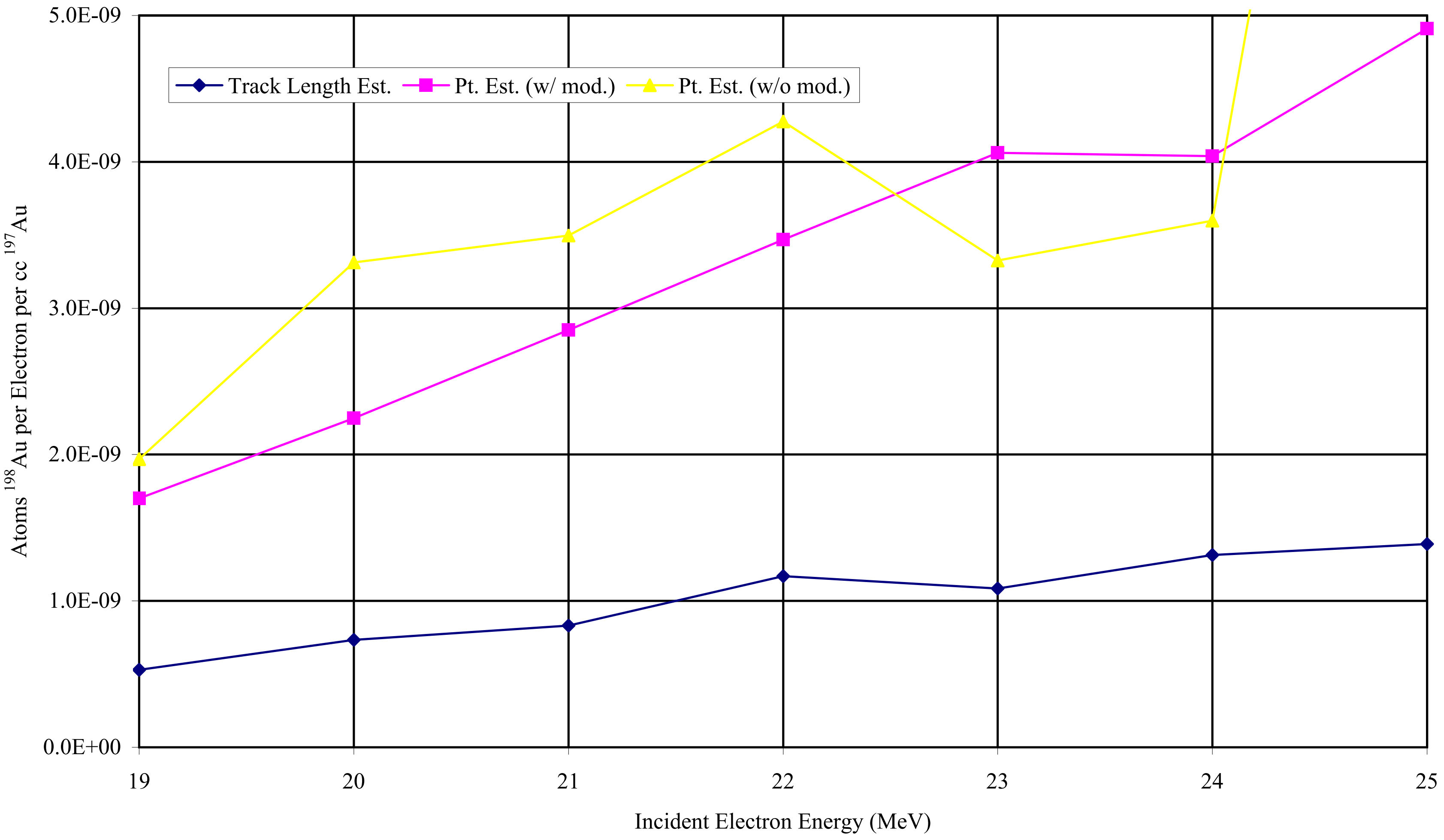

Figure 5-16. Calculated production rate of ^{198}Au in the ingot located in the maze.

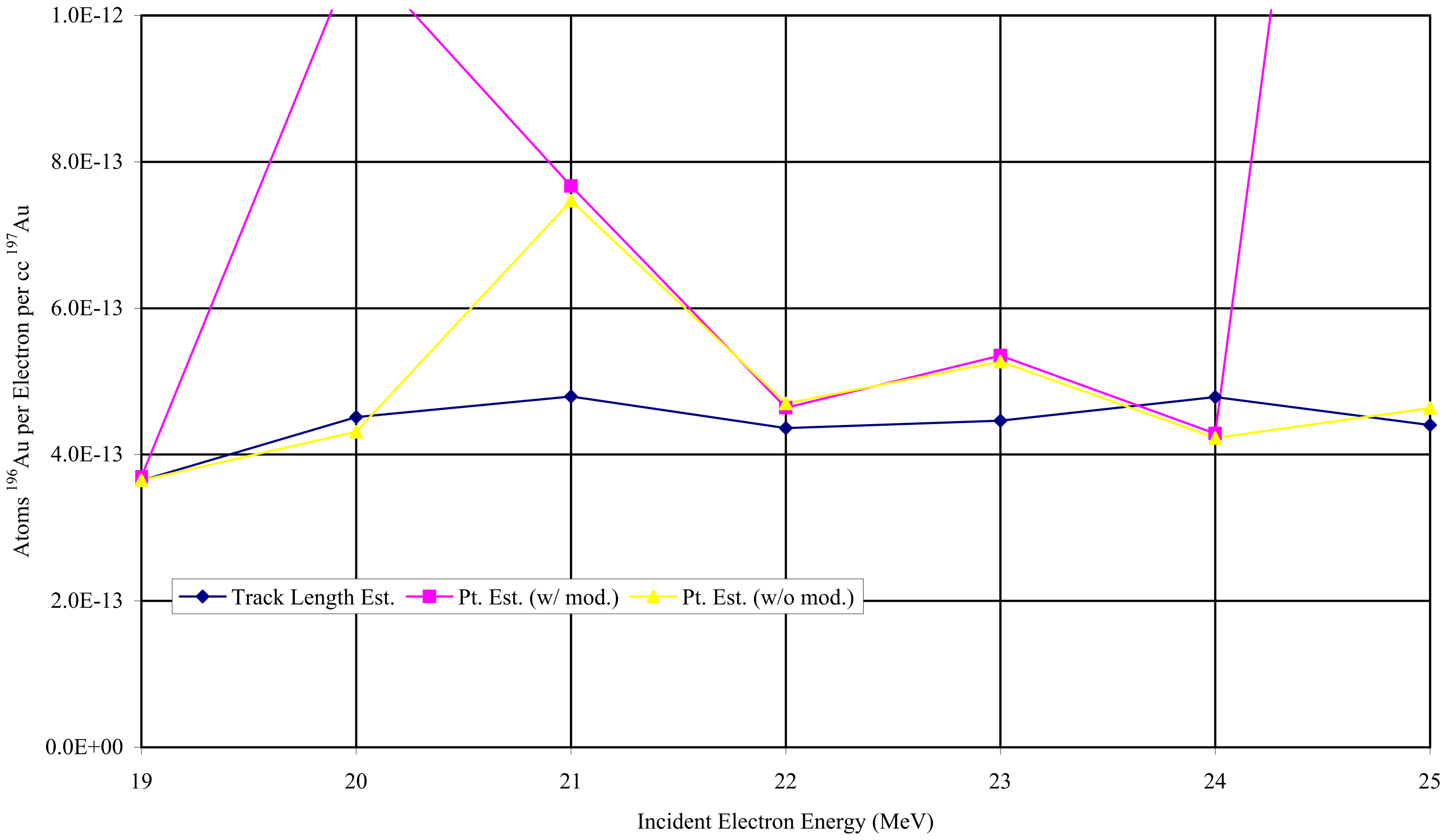

Figure 5-17. Calculated production rate of ^{196}Au in the ingot located in the maze using MCNP4BPN.

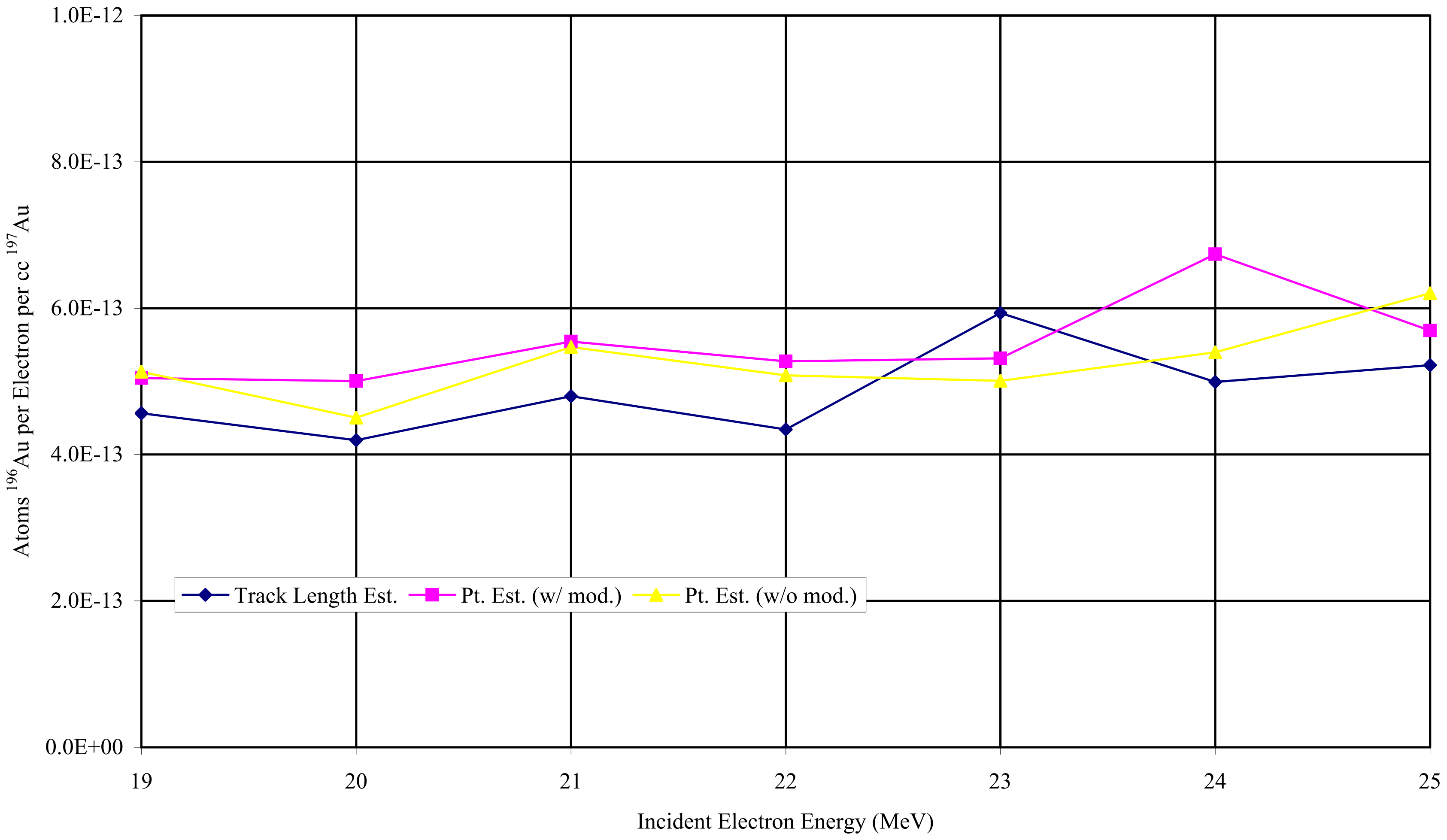

Figure 5-18. Calculated production rate of ^{196}Au in the ingot located in the maze using MCNP4BNU.

The production rate of ^{196}Au supports the conclusion that the electron-photon transport through the primary beam path in the treatment head is accurately simulated by this model. The comparison shows three matches within 15 percent difference which is in turn well within the accuracy of the individual values. The only anomaly is the production rate in the sample in the maze.

From the discussion above, it should be remembered that the secondary objective of the sample in the maze was to determine if high-energy photons were being produced outside of the target area. The experimental production rate for that sample is four orders of magnitude greater than the simulation. This is a clear indication that there is either an unknown streaming path through the primary collimator, an unlikely situation, or that there is another source of high-energy bremsstrahlung photons. It is believed that there is a secondary source and that the source is the energy selection slit in the bending magnet system. Due to their initial direction and the shielding in their path, photons produced at the energy selection slit would not significantly affect the photon flux in the beam path near isocenter. However, if such photons are being produced, they could represent a significant source of high-energy photons, and thus photoneutrons, that are not included within this model. The experimental observation of high-energy photons in the maze supports this hypothesis.

The production rate of ^{198}Au supports the conclusion that the electron-photon-neutron model is inadequate to simulate neutron estimates with an uncertainty of less than a factor of three. This uncertainty was derived from these final numbers although it has been quoted in the discussion above. With that said, this methodology still represents

Table 5-3. Experimental and simulated production rates of ^{198}Au for four configurations.

Configuration	Production Rate (^{198}Au/MU/cc^{197}Au)		
	Experiment (E)	Simulation (S)	S/E
Sample I1 Bare Ingot at Isocenter	$(4.67\pm2.34)\cdot10^{4}$	$(8.40\pm$x.xx$)\cdot10^{4}$	1.799
Sample I2 Moderated Ingot at Isocenter	$(5.37\pm2.69)\cdot10^{5}$	$(2.86\pm$x.xx$)\cdot10^{5}$	0.533
Samples 1 & 2 Moderated Foil Near Isocenter	$(2.28\pm0.25)\cdot10^{6}$	$(1.14\pm$x.xx$)\cdot10^{6}$	0.500
Sample I3 Bare Ingot in Maze	$(3.30\pm1.65)\cdot10^{3}$	$(1.07\pm$x.xx$)\cdot10^{4}$	3.242

a giant leap forward in the state-of-the-art for simulating these quantities in that coupled simulations can be run and an accurate assessment of their uncertainty is available. Further, the data make clear the areas of highest concern and help to prioritize future work to improve the accuracy of this simulation model.

The final conclusion from the ^{196}Au production data above indicate the possible existence of a photoneutron source not modeled in the current simulation. The comparison for the moderated foil and the moderated ingot both indicate that the neutron production is too low, possibly by a factor of two. These two bits of evidence lend credence to the hypothesis that the energy selection slit is a significant source of high-energy photons and therefore photoneutrons. This should be one of the first areas addressed by future work.

Further, from the ^{198}Au production data it is concluded that the simulation of neutron scattering and absorption is inadequate. During the course of this study, many variations of physical geometry and materials were explored. It was found from these studies that the scattered neutron flux was most sensitive to the materials, in particular tungsten, in the treatment head outside the primary beam path. With no other changes, the ^{198}Au production in the bare ingot at isocenter could be reduced by a factor of three by realistic increases in the size of the primary collimator. This is primarily due to the fact that tungsten has an thermal absorption cross section two orders of magnitude greater than lead. Placement of lead throughout the treatment head had a lessor, though still significant, effect. One of the recommendations discussed above is to visually inspect the treatment head in order to obtain an accurate model of the shielding outside the beam path.

Several other factors also have a significant, though smaller, impact on the neutron scattering and absorption. Once the two questions raised above have been answered, the lesser problems will become more important to solve. Included in this is an accurate description of the finished wall. Every scattered neutron traverses this geometry numerous times. If any material with a significant capture cross section exists within the general wall-board, it will have an impact. The main body of the accelerator and the partition wall of the machine closet represent a significant amount of material available to scatter or absorb neutrons. These and possibly other materials in the room will eventually have to be modeled. Last, as new cross-section data becomes available, the description of the neutron production will improve.

Implications

Now that the region of applicability for this simulation model is known, it can be applied to the general problems at hand. In particular, two pressing questions exist. The first is to estimate the dose due to photoneutrons around the MEA. The second is evaluate the relative contributions from the photon and neutron components of the dose as the mean electron energy is increased. These two questions are addressed simultaneously in this section.

For the purpose of addressing these questions, the physical model used for the activation simulations is used without change. In reality, the accelerator would be redesigned for any significant change in the incident electron energy in order to provide an appropriate photon field in that target area. However, that is beyond the scope of this study.

For the purpose of calculating the dose, the electron-photon dose and the neutron dose are calculated separately. Similar to the studies performed above, MCNP4BNU is used for the electron-photon transport problem and MNCP4BPN is used for the electron-photon-neutron transport problem. The dose is estimated using point detector tallies multiplied by appropriate flux-to-effective-dose conversion factors. The conversion factors are taken from a tabulation by Rogers [106] for photons and the tabulation by the ICRU [107] for neutrons. A hidden assumption necessary for this comparison is that the number of electrons on target stays constant ($1.36 \cdot 10^{13}$) throughout these calculations.

Figure 5-19 shows the dose at isocenter over the mean electron energy range 10 to 100 MeV. Note that both the photon and neutron dose are not strongly dependent on the field size. Field size is therefore ignored in the following discussion despite the fact that it is included for each of the graphs. It can be observed that the ratio of the photon to neutron dose remains fairly constant in this comparison.

The neutron dose at isocenter is approximately three orders of magnitude below the photon dose. This is as it should be. Many state regulations specify an upper limit of 0.1% of the dose from neutrons. However, the figure shows that the two curves converge

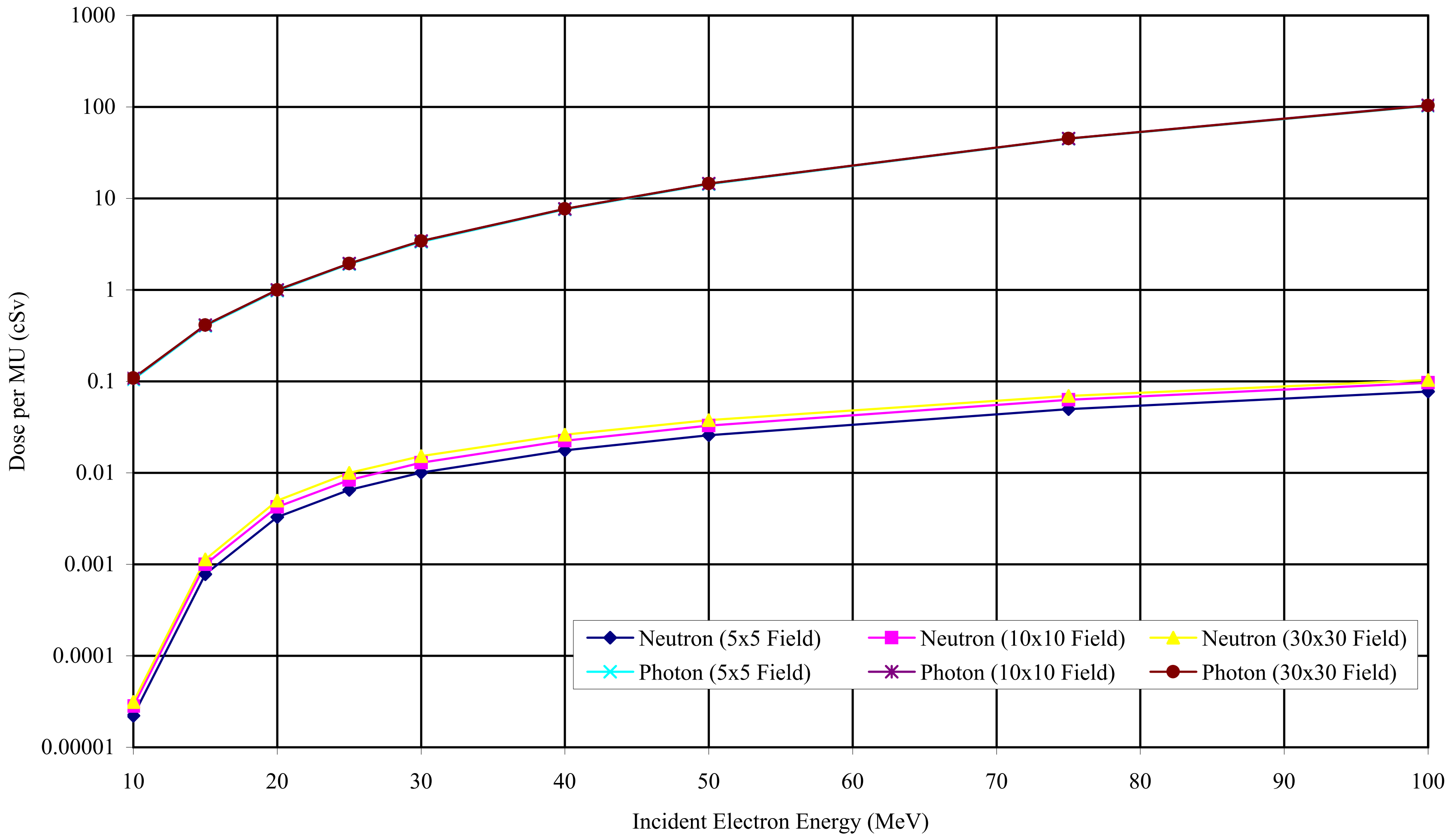

Figure 5-19. Theoretical neutron and photon dose per monitor unit at isocenter.

somewhat in the 15 to 50 MeV range with the point of closest approach around 20 to 25 MeV. The ratio in this energy range is around 0.0035 to 0.005, slightly above what is desired. While this is of concern, it should be reviewed with the understanding that there is a large uncertainty in the calculated value.

Figures 5-20 through 5-22 present the dose at one meter above, in front of and to the side of the electron target, respectively. As might be expected due to the symmetric nature of the treatment head model, they show very similar results. Of significant note is that the neutron dose quickly exceeds the 0.1% limit desired. This is due to the lack of shielding around the treatment head and can be remedied.

It is also worth noting that the photon dose is relatively flat in these locations. In fact, the photon dose is relatively flat for all locations except at isocenter. This is useful because it indicates that photon shielding in place is adequate for this entire energy regime.

The neutron dose becomes the primary shielding problem in the maze corridor. This is not unexpected. The neutron's ability to scatter and utilize streaming paths has already been discussed. The purpose of the maze is to create a longer distance between their inception and their leakage into occupied areas. Figures 5-23 and 5-24 show the dose at both extremities of the maze corridor.

The effective dose calculated at the entrance door for a typical 2000 MU treatment is 0.001 cSv (1 millirem). This calculation is too high and again argues that more work is needed to create a more detailed simulation model. The error is probably due to the large uncertainties in the neutron shielding around the treatment head and the uncertainties in modeling possible neutron absorbers in the room and maze.

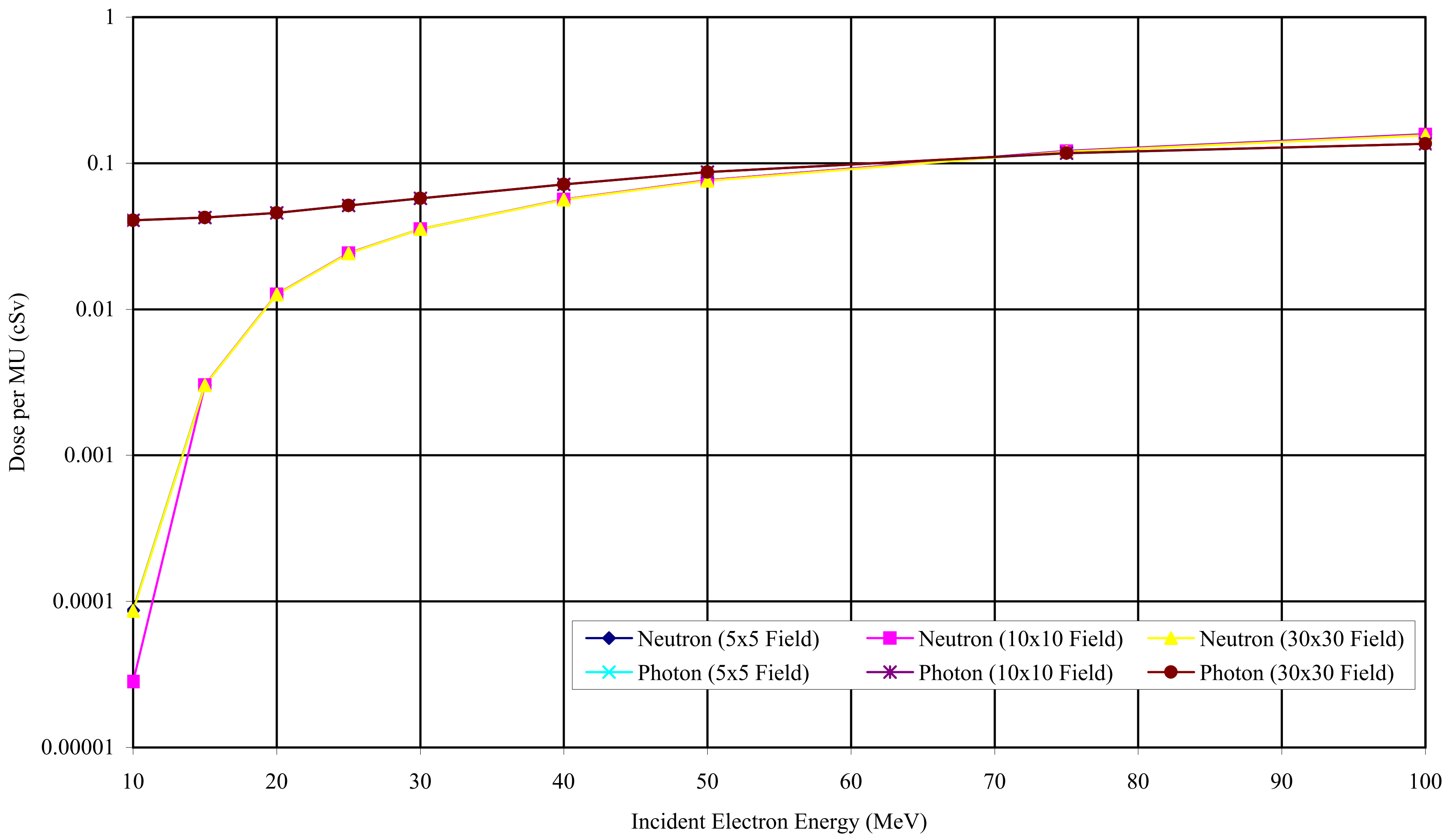

Figure 5-20. Theoretical neutron and photon dose per monitor one meter above the target.

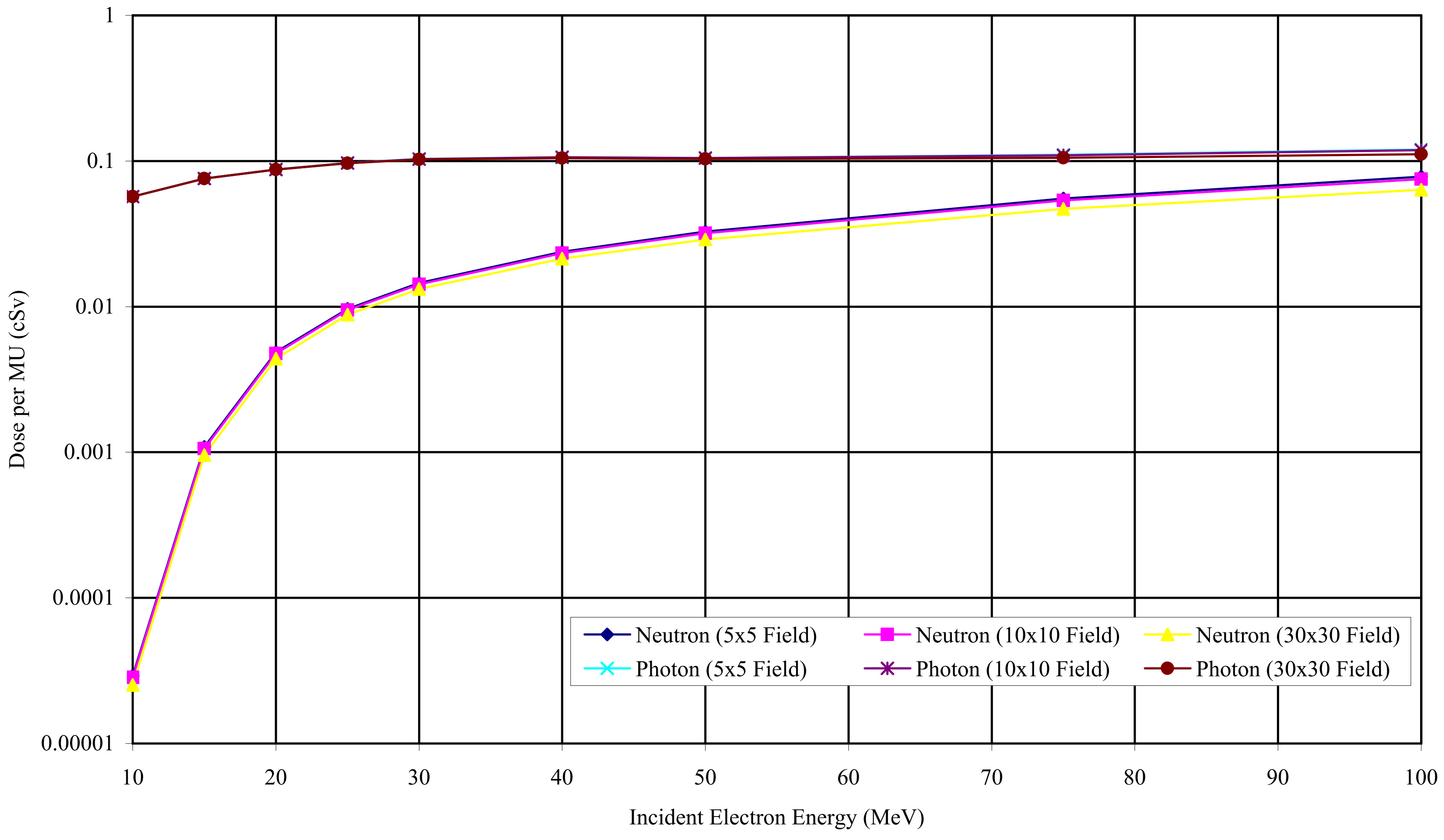

Figure 5-21. Theoretical neutron and photon dose per monitor unit one meter from the target in the cross-plane.

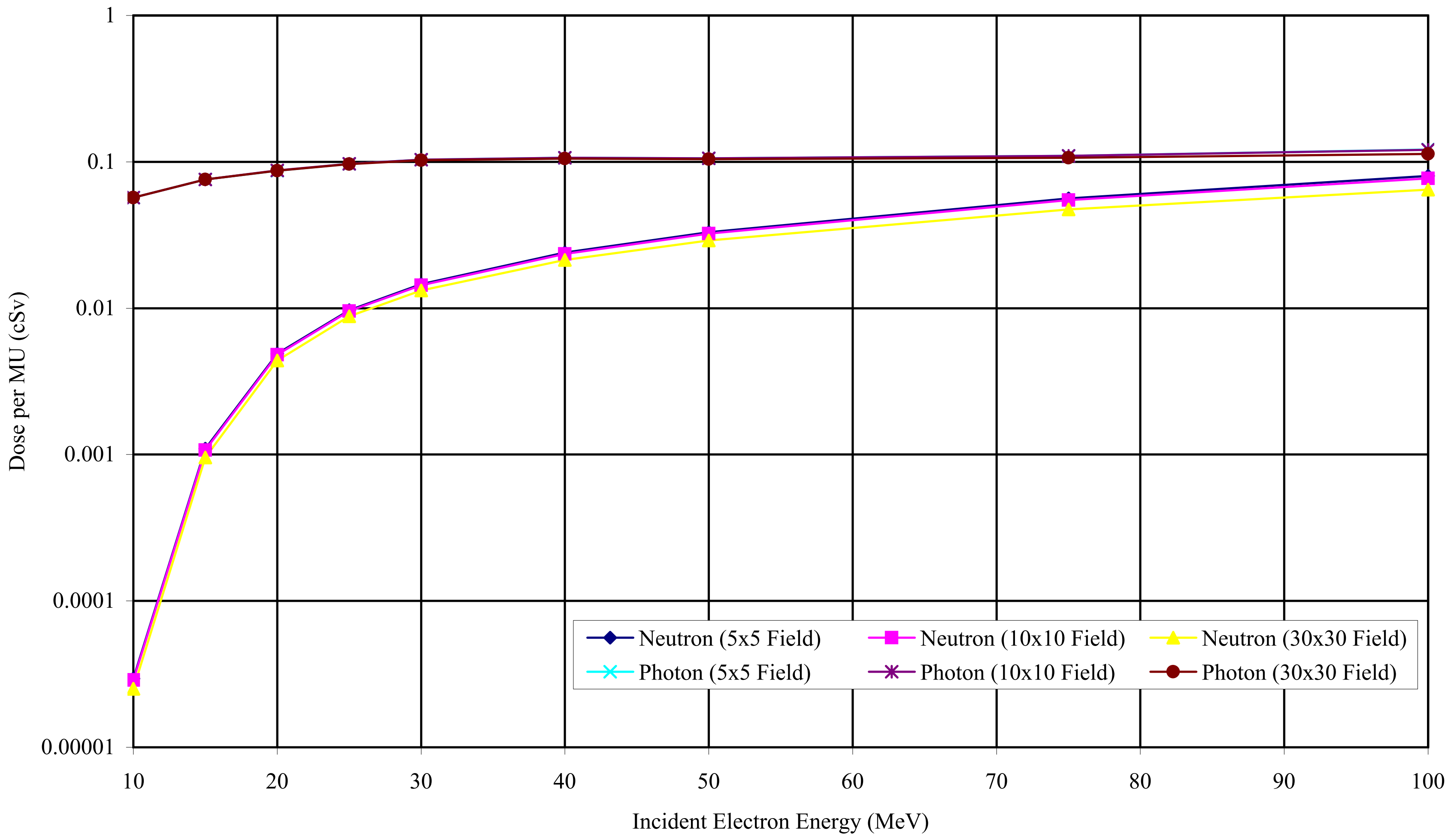

Figure 5-22. Theoretical neutron and photon dose per monitor unit one meter from the target in the in-plane.

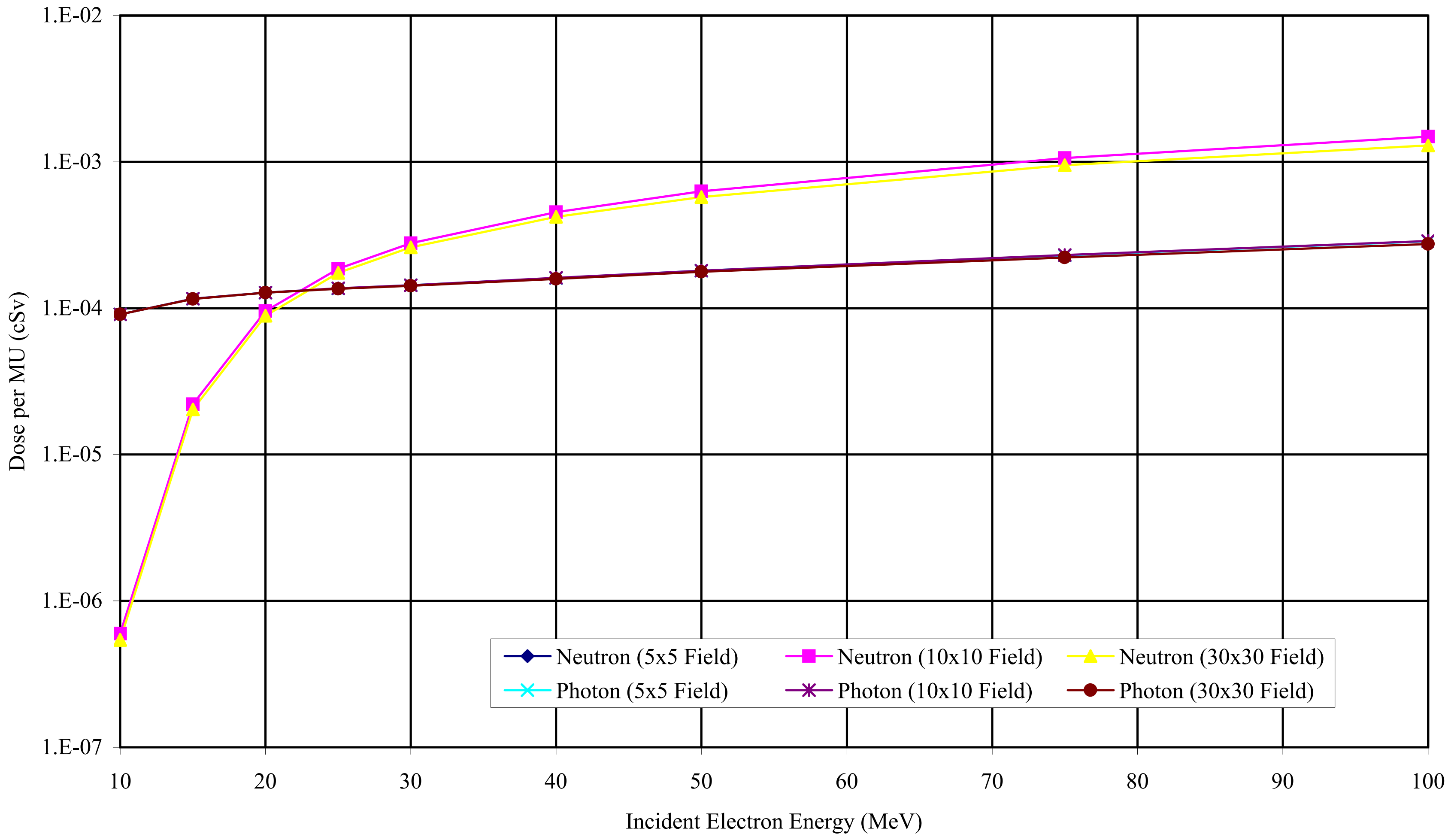

Figure 5-23. Theoretical neutron and photon dose per monitor unit just inside of the maze.

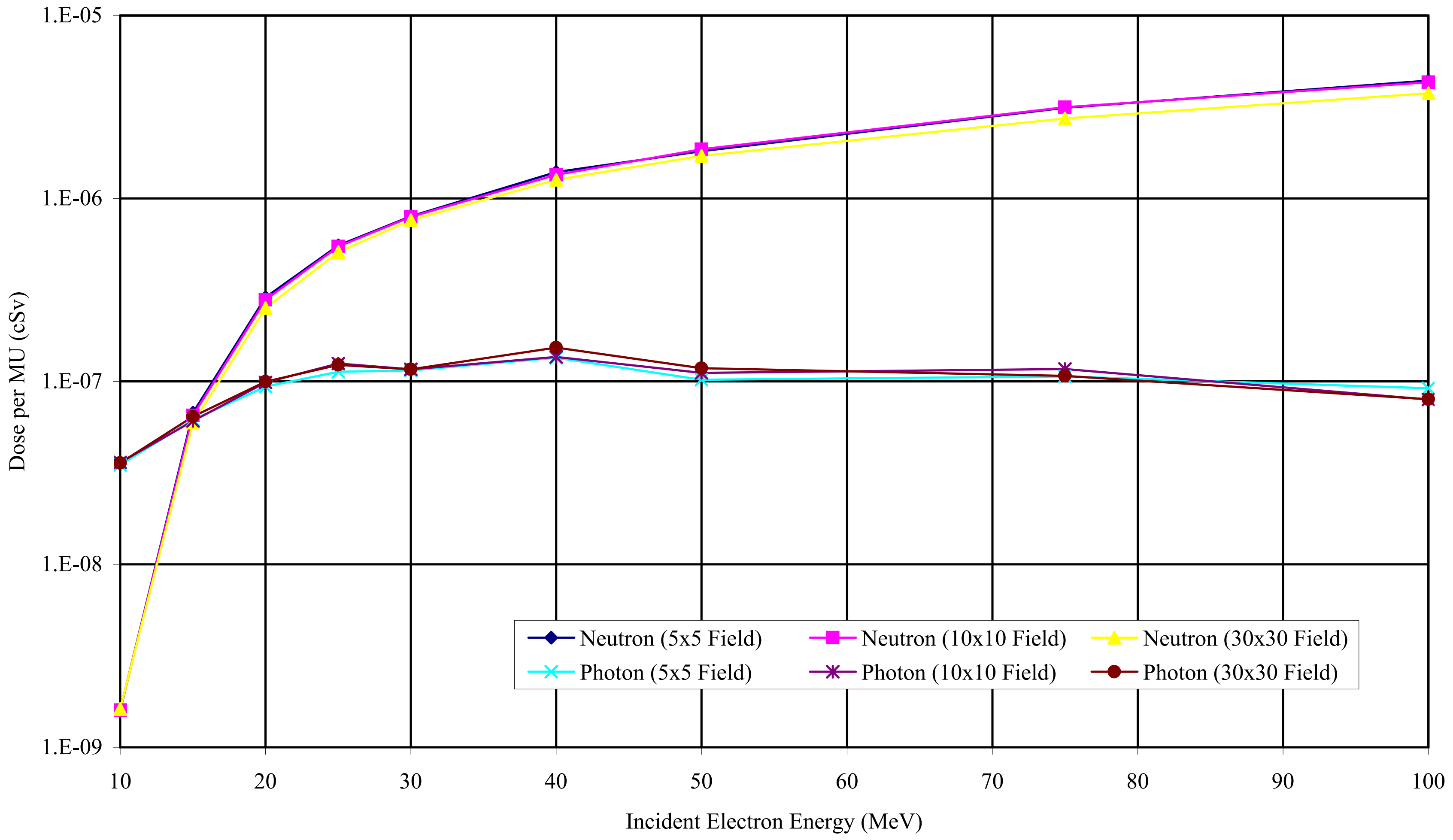

Figure 5-24. Theoretical neutron and photon dose per monitor unit at the door to the maze.

CHAPTER 6
SUMMARY AND CONCLUSIONS

The development and implementation of a systematic treatment of photonuclear physics for use in coupled photon-neutron simulations has been presented. This capability is based on the use of evaluated photonuclear data to enable Monte Carlo sampling of tabulated data describing photonuclear interactions and the resultant products. As such, it represents the state-of-the art in simulation capability using the most accurate data available. These new developments have been assessed through the process of verification and validation. Additionally, an initial application to the simulation of medical electron accelerators (MEAs) has been presented.

The Evaluated Nuclear Data File (ENDF) format is the international standard for representation of nuclear cross-section data in a complete manner. The Cross Section Evaluation Working Group (CSEWG) of the National Nuclear Data Center (NNDC) maintains ENDF/B-VI data library containing the recommended values for the United States. For the first time ever, evaluated photonuclear data have been made available in this format and they are undergoing review for inclusion into the ENDF/B library.

The Nuclear Theory and Applications Group of the Los Alamos National Laboratory has created the Los Alamos LA150 cross-section evaluation library that includes evaluated data up to 150 MeV incident energy for photonuclear interactions with selected materials. The evaluated data provided are complete descriptions of all possible photonuclear reactions. That is, the absorption of the photon and the subsequent emission of all secondary particles is handled in a self-consistent manner to describe the spectra of all emission products, not just neutrons. It is expected that these new photonuclear evaluations will be formally accepted for inclusion in the ENDF/B-VI library by CSEWG at a forthcoming meeting.

Further, the International Atomic Energy Agency (IAEA) has maintained a Coordinated Research Project over the last three years with the goal to establish an internationally accepted evaluated photonuclear library. That library and its associated report are expected to be released later in 2000 and will include the evaluations in ENDF format for the major isotopes of interest for photoneutron production. The newly available evaluated data from both T-2 and IAEA represents a first-of-a-kind advancement in this field that will help bring the accuracy to photonuclear simulations that was previously available only to neutron, electron and photoatomic simulations.

The Monte Carlo N-Particle (MCNP) radiation transport code has long represented the state-of-the-art in neutron, electron and photoatomic transport simulations. The reputation of this code is in large measure due to the quality of the data underlying the calculations. The MCNP code uses tabular data that include interaction probabilities with complete descriptions of the resultant products. These data are maintained in A Compact ENDF (ACE) format derived from evaluated data in ENDF format. The current work has defined a new class of ACE table for the inclusion of photonuclear data and presented a simple data processing code for the conversion of ENDF evaluated photonuclear data into this new ACE format. The LA150 photonuclear data available have been processed in this manner.

The implementation of photonuclear interactions into MCNP has also been presented. This included the definition of new user interface options for specifying the photonuclear data to be used in a transport simulation. The standard definitions available in MCNP have been extended to accept the specification of photonuclear tables and libraries. In addition, since evaluated photonuclear data may not exist for the corresponding neutron data, provisions have been added to allow the code user to specify the most accurate neutron and photonuclear data separately. The setup and storage sections of the code have been updated such that the tabular data specified for use is read into memory in the standard manner.

With the introduction of complete evaluated data and their conversion into ACE formatted tables, the sampling of emission particles from photonuclear absorption can be performed using the existing ACE sampling routines. Slight modifications to these routines were necessary to ensure the appropriate handling of new incident and emitted particle types. These modified routines have the additional benefit of immediate use for sampling nuclear interaction from new tabular proton data within the MCNPX code.

The photon collision routines within MCNP have been updated to include sampling of photonuclear interactions. This includes accounting for the photonuclear cross section in the distance-to-collision calculation and appropriately sampling secondary particles produced from a photonuclear collision. This coding is fully integrated such that coupled simulation of photon-neutron transport uses the standard MCNP framework of routines. This means that tallies, variance reduction techniques and summary information all reflect the coupled simulation.

Since the photonuclear interaction is a rare event, a biasing scheme has been introduced to increase the sampling of these events. The code user now has the ability to simulate the photonuclear contribution to the radiation field from every photon collision. Further, the photonuclear collision routine has been integrated with the standard MCNP weight-window scheme to ensure that particle weights from photonuclear production do not unnecessarily introduce large variations in the tally results. These enhancements significantly reduce the computational run-times necessary to achieve statistically valid results.

The accuracy of the results from the new ability to simulate coupled photon-neutron radiation transport has also been appraised. Verification of the coding was done to ensure that the newly implemented algorithms performed as expected. Validation of the new data was achieved by comparison of simulation results with two sets of data found in the literature.

The National Council on Radiation Protection and Measurement (NCRP) provided recommendations for assessing the neutron production and transport around medical electron accelerators [2]. This report recommends the method developed by Swanson [31,32] for estimating neutron production. Swanson's work documents neutron yields from electrons incident on selected materials. Comparison of calculated yields from the current work to Swanson's revised [32] values shows that the current methodology is able to accurately assess the neutron production within the uncertainty of the underlying experimental data.

The experimental measurements of Barber and George [58] are the defining benchmark for neutron production from electrons incident on selected materials.

Comparison between the current calculated yields and the reported values shows agreement to better than 25 percent. These results directly validate those materials for which tabular data and experimental measurement were available. They indirectly validate the methodology used to create the tabular data and provide a basis for hypothesizing that all the available data are probably accurate to 25 percent for the prediction of neutron production. Further benchmark data will be necessary to directly conclude this. It should be noted that the difficulties in assessing nuclear interactions are well known and for this problem 25 percent uncertainty is considered an excellent first step.

An initial study has been prepared to determine the course future work should take for the accurate assessment of the neutron environment around medical electron accelerators. During the course of this study, it was demonstrated that the electron-photon component of the MEA treatment beam is accurately modeled using current simulation techniques.

However, despite the availability of a coupled simulation code and evaluated photonuclear data, the current simulations are unable to reproduce experimental measurements of neutron flux around the MEA with an accuracy better than a factor of three. It is concluded that this is primarily due to limitations in the simulation model's description of the materials and their placement within the treatment head of the MEA and to a lesser extent the placement of materials within the surrounding room. The lack of certain isotopic evaluations in the current photonuclear library also introduces a source of error though contributions from this source are expected to be less than those mentioned above. Therefore, the first task that must be completed by future work is the more accurate modeling of the physical space around the medical electron accelerator than has been necessary for electron-photon simulation.

Some interesting results have been obtained from the medical electron accelerator simulation model currently available. It was shown that the neutron dose to a patient during a typical radiotherapy treatment using high-energy photons (20 to 25 MeV) is on the borderline of what is generally considered acceptable. This was expected and has been a known concern for this type of treatment. This new capability provides a tool which can be used to refine the estimation of the neutron dose and explore the possibilities for reducing it. Of significant note, it was found by this study that tungsten is an excellent neutron shielding material for use in MEAs due to its larger neutron capture cross section. On the other hand, lead was seen to influence the neutron energy distribution but have negligible effort on the population.

The simulation methodology was also shown to be able to assess the health physics concerns around a typical medical electron accelerator. The model is capable of estimating the direct neutron dose to technicians working near the MEA though further refinement is necessary to reduce the large uncertainties. Future work should first concentrate on refining the simulation model to improve this assessment. Other work may also be undertaken to simulate the photon dose due to neutron activation of materials within the treatment room and their subsequent gamma-ray decay.

The development and implementation of a systematic treatment of photonuclear interactions in coupled photon-neutron simulations paves the way for many follow-on studies. As the effort to integrate the new evaluated data into the transport codes has been carried out under the supervision of staff from the Los Alamos National Laboratory, these developments are expected to make their way into publicly released versions of the MCNP and MCNPX codes. The data processing capability for creation of photonuclear ACE files has been integrated into the NJOY99 nuclear data processing code and released last December (1999).

In addition to the recommendations above, many other possibilities have been discussed with researchers in this field. Future studies may include the assessment of the charged particle dose to patients undergoing radiotherapy from light-ions produced by photonuclear reactions in tissue; the assessment and placement of neutron shielding for both personnel and equipment around electron accelerators used for radiotherapy and radiography; the assessment of the neutron source created by in-flight annihilation photons impinging on a heavy water target; the assessment of material compositions for unknown samples by photon interrogation; and many other interesting concepts. The capability to simulate photonuclear interactions is therefore offered up with great hope and expectation on what the future will bring.

APPENDIX A
PHOTONUCLEAR ACE TABLE FORMAT

Introduction

This appendix documents the class ‘u’ ACE table format as used by this work. ACE tables are compact versions (A Compact ENDF) of Evaluated Nuclear Data Files (ENDF). The MKPNT (MaKe PhotoNuclear Table) data processing code (described in Chapter 4 and listed in full in Appendix B) converts ENDF formatted data into the ACE ‘u’ table format as described here. Limitations on the formats MKPNT will handle are discussed in Chapter 4. Data in the ACE class ‘u’ format can then be used by the photonuclear version of MCNP (described in Chapter 4 and listed in Appendix C) for the simulation of radiation transport.

As the class ‘u’ format used here derives from the standard ACE format, the descriptions of the data presented have been adapted from those in Appendix F of the MCNP Users Guide [3]. However, many changes have been made to both streamline and augment the old format. Comments are included to document where changes have been made and to give guidance on appropriate uses of various representations. This appendix was written to be suitable for inclusion in the appropriate location in the MCNP [3] and MCNPX [42,108-110] user guides. Because of this, there is some information that is redundant to Chapter 4.

Table Layout

There are no changes from the standard ACE table layout. It is presented here in Table A-1. The format as shown describes an ASCII text file. The binary version of the file contains the data in the same order except stored in the machine dependent format for integers, reals and characters, respectively. There is one generally accepted exception from this format. Integers in the XSS array are typically written to the ASCII file in the (I20) format for readability although they are stored as real value numbers in the binary file and within the MCNP program.

A standard data library file contains multiple data tables, e.g. the MCPLIB02 library contains one photoatomic data set for each element from hydrogen to plutonium. A specific table within the library file is found by looking up its starting line (whose value is IRN) and referencing the data relative to the appropriate starting line. Similarly, when stored as binary data the address of the first entry for the table in question is the absolute starting point.

Table A-1. Standard table description for the photonuclear class ‘u’ ACE format.

Line Address		Contents	Format (Fortran Standard)
Relative	Absolute		
1	IRN	ZAID, Atomic Weight, Temperature, Date Processed	A10, 2E12.0,1X, A10
2	IRN+1	Comment	A80
3 – 6	IRN+2 – IRN+5	Inherited fields currently unused (Fill with zeros or leave blank)	4(I7, F11.0) per line
7 – 8	IRN+6 – IRN+7	(NXS(I):I=1..16)	8(I9) per line
9 – 12	IRN+8 – IRN+11	(JXS(I):I=1..32)	8(I9) per line
13 - …	IRN+11 – …	(XSS(I):I=1..LXS)	4(E20.0) per line

NXS Array Elements

Only data common to the whole table are stored in the NXS array. Specifically, it contains the information necessary to understand the details within the remainder of the table. Examples of this type of information include the number of energy points used, the number of reaction cross sections listed and the number of secondary particles with emission data. Because the format of this table was modified over several iterations, key details about the table format itself are also included here. The significance of each entry in the NXS array is documented in Table A-2.

Two elements of the NXS array have standard definitions. The first element of the NXS array is always the length (number of entries) of the XSS array. This standardization makes it possible to read in a generic table without knowing the details of the sub-arrays. The second element of the NXS array is typically the target identifier.

This document pertains to format version one (TVN=1) of the ACE class 'u' table. For this version of the table, the number of parameter entries in each IXS array is two (NPIXS=2), the number of entries in each IXS array is twelve (NEIXS=12).

Table A-2. Description of the NXS Array elements in a photonuclear class 'u' ACE format.

Entry	Parameter	Fixed numeric descriptive
NXS(1)	LXS	Length of the XSS data block
NXS(2)	ZA	Atomic and mass number of the target isotope ZA = Z*1000 + A
NXS(3)	NES	Number of energy entries in the main energy grid
NXS(4)	NTR	Number of MT entries in the reaction-type listing
NXS(5)	NTYPE	Number of secondary particle types with IXS information
NXS(6)	NPIXS	Number of parameter entries (fixed values) in the IXS array
NXS(7)	NEIXS	Number of entries (fixed values and locators) in IXS array per secondary particle
NXS(8-15)		Unused (Fill with value zero)
NXS(16)	TVN	Table Format Version

Parameters are fixed values listed first in the array. Other entries are assumed to be locators and their values updated as the table is shifted in memory. The maximum number of secondary particles (NTYPE) for which emission data can be given is eight. This structure and values of this table are subject to revision at which time the table version number will be incremented.

JXS Array Elements

The JXS array elements contain locators to global data contained within the XSS data block. Similar to the NXS elements, global applies to the main energy grid, the cross-sections, additional information associated with each reaction and a pointer to the secondary data array IXS. Locators are offsets into the XSS array. Descriptions of the JXS locators are given in Table A-3. For example, the first value for the main energy grid is located at XSS(ESZ).

This format has deviated from the traditional style in that it references all secondary particle emission data through the use of the IXS construct. The use of the IXS construct was first done for the LA150 neutron library [47,48]. This library was constructed for use in MCNPX where emission descriptions for protons, deuterons, tritons, helium-3 and alphas were desired in addition to neutrons and photons. The table presented here completes the transition in that all emission data (including photon and neutron) are referenced through the IXS array. This was done so that the table is now consistent in its treatment of all secondary particle emission data. As a result, only data general to the whole table should be referenced from the JXS array.

Another major change is the addition of the locators TOT, NON, ELS and THN. In neutron type ‘c’ tables, the locator ESZ does quintuple duty. That is the energy grid, Table A-3. Description of the JXS Array elements in a photonuclear class ‘u’ ACE format.

Entry	Locator	Offset to array of…
JXS(1)	ESZ	Main energy grid
JXS(2)	TOT	Total cross-section data
JXS(3)	NON	Total non-elastic cross-section data
JXS(4)	ELS	Elastic cross-section data
JXS(5)	THN	Total heating number data
JXS(6)	MTR	MT reaction numbers
JXS(7)	LQR	Q-value reaction energy data
JXS(8)	LSIG	Cross-section locators (relative to SIG)
JXS(9)	SIG	Primary locator for cross-section data
JXS(10)	IXSA	First word of IXS array
JXS(11)	IXS	First word of IXS block
JXS(12-32)		Unused (Fill with zeros)

the total cross-section, the absorption cross section, the elastic cross section and the heating numbers are referenced through the ESZ locator. These have now been separated. The absorption cross section has been replaced by the non-elastic cross section.

XSS Block

XSS Array

The XSS array is the generic container for the data. Because of this, it is also referred to as the XSS Block. Descriptions of each of the arrays and their associated values as located within the XSS block are presented here. It should be noted that the ACE format uses only one energy grid for all cross-section data and that all cross sections use linear-linear interpolation to determine intermediary values.

ESZ Array

The ESZ array contains the data entries for the main energy grid. It represents a superset of all energies used by any reaction cross section listed in the table. Energy values are given in units of MeV. The entries should consist of a series of monotonically increasing, positive values located at (XSS(I): I=ESZ..ESZ+NES-1). Duplicate entries are not allowed. Sharp transitions should be represented as occurring over a finite transition region rather than a true step change. Error checking should be done to ensure the conditions specified.

TOT Array

The TOT array contains the data entries for the total cross section. Cross-section values are given in units of barns. There must be an entry corresponding to each entry in the ESZ array and they should be located at (XSS(I): I=TOT..TOT+NES-1). Error checking should be performed to ensure that these values are equivalent to the sum of the elastic and non-elastic cross sections. This array must be present.

NON Array

The NON array contains the data entries for the total non-elastic cross section. Cross-section values are given in units of barns. There must be an entry corresponding to each entry in the ESZ array and they should be located at (XSS(I): I=NON..NON+NES-1). Error checking should be performed to ensure that these values are equivalent to the sum of all partial cross sections excluding the elastic and any sub-totals. This array must exist if any non-elastic cross-section data are present.

The non-elastic cross-section is listed rather than the absorption for convenience. If the elastic cross section has not been included, the non-elastic cross section is identical to the total cross section and NON should be set equal to TOT and only one set of cross-section entries is needed. One justification for not including the total absorption cross section is that it does not make physical sense for the photonuclear process. Gamma rays are emitted for all reactions that do not transition directly to a ground state.

ELS Array

The ELS array contains the data entries for the elastic cross section. Cross-section values are given in units of barns. There must be an entry corresponding to each entry in the ESZ array and they should be located at (XSS(I): I=ELS..ELS+NES-1). For photonuclear physics, this cross section is negligible and typically is not included in the original evaluation data file. If it is not included, ELS must be set to zero and no entries are included in the XSS array.

THN Array

The THN array contains the data entries for the average heating numbers, i.e. the average energy deposited per collision. Heating-number values are given in units of MeV per collision. There must be an entry corresponding to each entry in the ESZ array and they should be located at (XSS(I): I=THN..THN+NES-1). If no data have been calculated for heating numbers, THN and each PHN entry (see discussion below) must be set to zero and no entries are made in the XSS array.

The total heating number has recently undergone a revision. Since the MCNPX code is capable of transporting most particles of interest, it is necessary to be able to adjust the total heating numbers appropriately. Specifically, at the time of the simulation, the total heating number should be adjusted to represent the average amount of energy deposited per collision in a given material for all particles that are not transported.

In order to obtain an "average" heating number, several assumptions are necessary. Particles that are extremely penetrating, e.g. neutrinos, are assumed to deposit their energy elsewhere. Particles that are of "limited" range, including neutrons and photons, are not considered extremely penetrating. All particles of "limited" range are assumed to deposit their energy instantaneously at the collision site. This is a poor assumption for any situation that does not approximate an infinite, homogeneous medium with a steady-state source.

In order to accurately represent energy deposition, particles for which secondary distribution data exist and should be transported and their contribution to the total heating number subtracted off the total before beginning the simulation. Again, if heating tallies are used, it is essential to transport all particles for which instantaneous, local energy deposition is not a good assumption. Note that the sum of the partial heating numbers (PHNs) given in the table may not add up to the total unless all possible secondary particles (including the recoil particles) are included.

MTR Array

The MTR array contains the data entries for the reaction type MT numbers. Reaction type MT numbers are taken directly from the ENDF-102 File Format Manual [45]. There must be one MT value in the array for every reaction cross section to be listed and they should be located at (XSS(I): I=MTR..MTR+NTR-1). The entries should be in ascending order according to their numeric value.

Production cross sections for reaction products of interest may also be listed in the MT array by using the ZA number in place of the ENDF MT number. An isotope's ZA number is defined as the atomic number (Z) times one-thousand plus the atomic mass number (A). For example, ^{9}Be would have ZA equal to 4009. There is not a conflict with MT numbers as the currently defined ENDF MT numbers end at 1000. Note that production cross sections, i.e. all reactions with a MT value greater than 1000, are not valid for transport and are used only as tally multipliers.

LQR Array

The LQR array contains the data entries for the Q-value associated with each reaction. Q-values are given in units of MeV. There must be one entry corresponding to each MT array entry and they should be located at (XSS(I): I=LQR..LQR+NTR-1). For reactions which are not physical events, e.g. production and summation listings, the Q-value should be given as a zero (0) entry.

LSIG Array

The LSIG array contains the entries for the cross-section locators. Cross-section locators are the array index to the first word of the corresponding MT reaction data relative to the SIG locator. There must be one entry corresponding to each MT array entry and they should be located at (XSS(I): I=LSIG..LSIG+NTR-1).

SIG Array

The SIG locator is the primary reference for finding the reaction cross-section data. By tradition and for convenience, all reaction cross-section data are listed sequentially at one location within the XSS array. Cross-section values are given in units of barns. Each cross-section locator must point to a valid reaction cross-section.

Reaction cross-section data are given over a defined range of energies on the main energy grid. The entries follow the format IE, NE, VALUES where IE is the starting index corresponding to an entry on the main energy grid and NE is the number of entries. Thus the entries VALUES(1..NE) are the reaction cross-section values corresponding to the energies ESZ(IE..IE+NE-1). The cross-section entries should be located at ((XSS(I): I=SIG+LSIG(K)-1..SIG+LSIG(K)+NE): K=1..NTR) where LSIG(K) is the offset value from the LSIG array that corresponds to the Kth reaction MT as listed in the MTR array. Error checking should be done to ensure that IE is not less than one, that NE is not greater than NES and that IE+NE-1 is less than or equal to NES.

For cross-section data that do not cover the entire energy range of the table, the value of the last entry is assumed to be constant for the remainder of the main energy grid. That is, for all energies up to the first entry, the cross-section value of the first entry is used. Therefore, reactions with threshold values must start with a zero value entry. Similarly, reactions which are negligible after a certain energy should contain a zero value as their last entry.

IXS Block

The IXS block is a conceptual figment created to equate the secondary particle information storage structure to the general storage model of the NXS/JXS/XSS block. It is described as its own set of parameters, locators and data in order to help separate it conceptually and make it easier to understand. In reality all the components of the IXS array and its associated data are stored in the XSS block. To stress the point, references to IXS(i) are equivalent to XSS(i). The secondary data should be listed sequentially by particle type and not spread throughout the XSS block.

IXS Array

The IXS array emulates the parameter/locator concept of NXS/JXS for secondary particle information. Since a full set of IXS elements is needed for each secondary particle, there are typically multiple IXS arrays in a table. They are listed sequentially located at ((XSS(I): I=(IXSA+NEIXS*(J-1)) .. (IXSA+NEIXS*(J-1))+(NEIXS-1): J=1..NTYPE). The elements of the IXS array are described in Table A-4.

The photonuclear table differs from the neutron and proton table versions in that parameters specific to a secondary particle are also included in the IXS array. Thus, IXS elements perform functions similar to NXS (parameter) and JXS (locator) elements. In practice, they are used in an analogous manner to their conceptual equivalents. For example, for the third (3) emission particle the parameter NTRP(3) (stored as IXS[2,3]) and the locator MTRP(3) (stored as IXS[5,3]) can be used to find the array of MT reactions that produce that particle located at (XSS(I): I=MTRP(3)..MTRP(3)+NTRP(3)-1).

Table A-4. Description of the IXS Array elements in a photonuclear class 'u' ACE format.

Entry	Parameter	Fixed number descriptive
IXS(1,J)	IPT(J)	Particle IPT number
IXS(2,J)	NTRP(J)	Number of MT reactions producing this particle
Entry	Locator	Offset to array of…
IXS(3,J)	PXS(J)	Total particle production cross-section data
IXS(4,J)	PHN(J)	Particle average heating number data
IXS(5,J)	MTRP(J)	Particle production MT reaction numbers
IXS(6,J)	TYRP(J)	Reaction coordinate system data
IXS(7,J)	LSIGP(J)	Reaction yield locators (relative to SIGP)
IXS(8,J)	SIGP(J)	Primary locator for reaction yield data
IXS(9,J)	LANDP(J)	Reaction angular distribution locators (relative to ANDP)
IXS(10,J)	ANDP(J)	Primary locator for angular distribution data
IXS(11,J)	LDLWP(J)	Reaction energy distribution locators (relative to DLWP)
IXS(12,J)	DLWP(J)	Primary locator for energy distribution data

This table also differs from neutron and proton tables in that all secondary particle information is referenced through the relevant IXS elements. This is a change in that the previous tables still referenced the incident particle type emission data (e.g. neutron in, neutron out) through the JXS array. Photophoton emission data in a photonuclear table are referenced through the IXS array exactly as photoneutron or photoproton emission data.

The secondary particles for which data are supplied are identified by the index IPT. These numbers were originally defined by the MCNP code for neutrons, photons and electrons. They have been extended by the MCNPX code to cover the particles of interest in high-energy accelerator environments. At the current time, only those particles listed in Table A-5 have emission data available in the table. This is the source of the maximum value limit for the parameter NTYPE.

PXS Array

This array contains the data entries for the total secondary particle-production cross section. Production cross-section values are given in units of barns. The data values follow the IE, NE, VALUES format as described in the SIGP array section above

Table A-5. Association of particles with their symbol and IPT index number as defined in MCNP(X).

Particle Name	Symbol (from mode card)	IPT
neutron	n	1
photon	p	2
electron	e	3
proton	h	9
deuteron	d	31
triton	t	32
helium_3	s	33
alpha	a	34

and are referenced to the main energy grid. The array must exist if any reaction data exist for the particle type and is located at ((XSS(I): I=PXS(J)..PXS(J)+NE+1): J=1..NTYPE). Error checking should be done to ensure that the value of each entry corresponds to the sum of the relevant reaction yields.

PHN Array

This array contains the data entries for the particle average heating numbers. The data values follow the IE, NE, VALUES format as described in the SIGP array section above and are located at ((XSS(I): I=PHN(J)..PHN(J)+NE+1): J=1..NTYPE). Particle average heating-numbers are given in units of MeV per collision. As described in the discussion of the THN array above, these values are the contribution to the total heating number by this particle type assuming that the particle's average emission energy is deposited locally. Error checking should be done to make sure that PHN is not greater than THN.

MTRP Array

This array contains the data entries for the MT reaction-type numbers that produce this secondary particle. MT-reaction numbers are specified in the same manner here as for the MTR array and are located at ((XSS(I): I=MTRP(J)..MTRP(J)+NTRP(J)-1): J=1..NTYPE). The entries should be in ascending numeric order. Error checking should be done to ensure that all MTRP entries correspond to a MTR entry at the JXS level.

TYRP Array

This array contains the data entries for the coordinate system of the reaction producing the secondary particle. The emission-coordinate-system parameter indicates either the lab system (value = 1) or the center-of-mass system (value = -1) and the entries are located at ((XSS(I): I=TYRP(J)..TYRP(J)+NTRP(J)-1): J=1..NTYPE). Error checking should be done to ensure that an entry exists for each reaction and that it contains one of the two allowed values. This array is different than the TYR array for neutron tables in that multiplicity data are not included in TYRP but instead utilize the SIGP array.

LSIGP Array

This array contains the entries for the reaction yield locators. Reaction yield locators are the relative location of the corresponding MT reaction data in the SIGP array. There must be one entry corresponding to each MTRP array entry and they should be located at ((XSS(I): I=LSIGP(J)..LSIGP(J)+NTRP-1): J=1..NTYPE). The notation LSIGP(K,J) indicates the Kth entry (XSS(LSIGP(J)+K-1)) for the Jth secondary particle.

SIGP Array

The SIGP locator is the primary reference for finding the reaction yield data. All reaction cross-section data for this secondary particle are listed sequentially within the SIGP array. Reaction yields are given either as production cross sections or as multiplicity data. There must be one set of data for each reaction specified in the MTRP array and it is located as described in the relevant table below.

Production cross-section data are the simpler of the two yield descriptions. Production cross-section values are given in units of barns. Data of this type are typically

Table A-6. Reaction yield data in the form of a production cross-section.

Location in XSS	Parameter	Description
SIGP(J)+LSIGP(K,J)-1	MFTYPE	13 – Production cross-section
SIGP(J)+LSIGP(K,J)	IE	Starting index on main energy grid
SIGP(J)+LSIGP(K,J)+1	NE	Number of consecutive entries
SIGP(J)+LSIGP(K,J)+2 .. SIGP(J)+LSIGP(K,J)+NE+1	PXS(I) I=1..NE	Production cross-section values for corresponding MT reaction (linear-linear interpolation)

derived from File 13 of an ENDF evaluation and are therefore labeled with the MFTYPE equal 13. The entries for this reaction yield are described in Table A-6. The average "multiplicity" for the reaction as a function of energy can be calculated from the data by dividing the production cross-section value by the corresponding MT reaction cross-section value.

Alternatively, the multiplicity of the reaction may be used in conjunction with the corresponding MT reaction cross section to determine the production cross section. Multiplicity is unitless and implies the number of particles emitted per collision. Multiplicities can be constant, e.g. for fixed reactions like (γ,2n), or they can be variable, e.g. for fission nubar values. Note that the general reaction, MT 5, can include any combination of true reactions in a variable multiplicity. Also note that fission nubar data are now included generally in this array rather than specifically in their own array.

Yield data of this type are typically found in File 6 or 12 of the ENDF-6 format and hence are assigned the MFTYPE of 6 or 12. MFTYPE 16 is also allowed due to a backwards compatibility issue arising from the fact that the value 16 has been used in past to indicate MF File 6 yield data in neutron tables. Yield data for MFTYPE 6, 12 and 16 are described in Table A-7. This table and many of those to follow use the INT interpolation parameter as defined by ENDF. Defined INT values and their associated formalism are listed in Table A-8. Error checking should be done to ensure that all values of MTMULT match a reaction in the MTR listing.

This description provides a concise method to define a varying yield. For example, the MT 5 general reaction has a set of energy/yield pairs which is typically shorter than the defining the corresponding production cross section. The disadvantage of this format is that it requires the lookup of the cross section and the yield followed by a multiplication of the two to obtain the production cross section.

Table A-7. Reaction yield data in the form of reaction multiplicity.

Location in XSS	Parameter	Description
IXS+SIGP(J)+LSIG(K,J)-1	MFTYPE	6,12 or 16 – Reaction multiplicity
IXS+SIGP(J)+LSIG(K,J)	MTMULT	MT reaction to which multiplicity applies
IXS+SIGP(J)+LSIG(K,J)+1	NR	Number of interpolation regions for multiplicity data (If NR = 0, NBT and INT are omitted and linear-linear interpolation is assumed across all points)
IXS+SIGP(J)+LSIG(K,J)+2 .. IXS+SIGP(J)+LSIG(K,J)+1+NR	NBT(I) I=1..NR	Starting index to which the corresponding interpolation parameter applies
IXS+SIGP(J)+LSIG(K,J)+2+NR .. IXS+SIGP(J)+LSIG(K,J)+1+2*NR	INT(I) I=1..NR	ENDF defined interpolation parameters
IXS+SIGP(J)+LSIG(K,J)+2+2*NR	NE	Number of energies at which the multiplicity is defined
IXS+SIGP(J)+LSIG(K,J)+3+2*NR .. IXS+SIGP(J)+LSIG(K,J)+2+2*NR+NE	E(I) I=1..NE	Energy grid on which multiplicities are defined
IXS+SIGP(J)+LSIG(K,J)+3+2*NR+NE.. IXS+SIGP(J)+LSIG(K,J)+3+2*NR+2*NE	Y(I) I=1..NE	Multiplicity (production cross-section = reaction MT cross-section * multiplicity)

LANDP Array

This array contains the entries for the angular distribution locators. An angular distribution locator is the location of the angular distribution data for the corresponding MT reaction relative to the ANDP locator. There must be one entry corresponding to each MTRP array entry and they should be located at ((XSS(I): I=LANDP(J).. LANDP(J)+NTRP-1): J=1..NTYPE). LANDP(K,J) is the Kth entry for the Jth secondary particle type.

Several LANDP array values have special meanings. A zero (0) locator value indicates a reaction where all particles are emitted isotropically in the reference frame defined by the corresponding entry in the TYRP array. Correlated energy/angle data are indicated by a negative one (-1) locator value. In this case, the angular distribution data are included with the energy emission distribution data in the DLWP array. For both cases, no angular data are entered in the ANDP array. All other locators must be positive integer values and indicate that the angular distribution data are contained in the ANDP array.

ANDP Array

The ANDP locator is the primary reference for finding angular distribution data. All angular distribution data for this secondary particle are listed sequentially in this array. Three types of angular distribution table are currently allowed: isotropic, 32 equi-probable bin or tabulated angular-bin data. There must be one set of distribution data for each reaction specified in the MTRP array that is neither isotropic nor correlated energy/angle. The angular distribution data is located as described in the relevant table.

Table A-8. Interpolation schemes as defined for the ENDF-6 format.

Interpolation Scheme	INT Value	Interpolation Equation
Histogram	0	y(x) = y(x$_0$)
Linear-Linear	1	$y(x)=\frac{y_1-y_0}{x_1-x_0}(x-x_0)+y_0$
Log-Linear	2	$y(x)=\frac{y_1-y_0}{\ln\left(\frac{x_1}{x_0}\right)}\ln\left(\frac{x}{x_0}\right)+y_0$
Linear-Log	3	$y(x)=\exp\left(\frac{\ln\left(\frac{y_1}{y_0}\right)}{x_1-x_0}(x-x_0)+\ln(y_0)\right)$
Log-Log	4	$y(x)=\exp\left(\frac{\ln\left(\frac{y_1}{y_0}\right)}{\ln\left(\frac{x_1}{x_0}\right)}\ln\left(\frac{x}{x_0}\right)+\ln(y_0)\right)$

If all reactions are isotropic or correlated energy/angle (i.e. no data are present in the ANDP array), ANDP should be set to zero.

The angular distribution data are a set of tables comprising the average-emission angles for the Jth emission particle having the Kth reaction. These table are located using the angular distribution header information is described in Table A-9. An example of how to find a particular header in the ANDP array is given here. The appropriate angular distribution header information for the second reaction listed in MTRP producing the third secondary-emission particle starts at the array location (XSS(ANDP(3)+LANDP(2,3)-1). ANDP(3) is the ninth IXS value for the third emission particle (IXS(9,3)) and LANDP(2,3) is the value of the second entry in the LANDP array for the third emission particle (XSS(IXS(10,3)-1+2)).

Table A-9. Angular distribution header information.

Location in XSS	Parameter	Description
ANDP(J)+LANDP(K,J)-1	NE	Number of energies at which angular distributions are tabulated
ANDP(J)+LANDP(K,J) .. ANDP(J)+LANDP(K,J)+NE-1	E(L) L=1..NE	Energy grid for the Kth reaction angular distribution
ANDP(J)+LANDP(K,J)+NE .. ANDP(J)+LANDP(K,J)+2*NE-1	LC(L) L=1..NE	Locators for the angular data corresponding to energy grid

Once the angular distribution header information is located, the incident energy is used to find the locator for the appropriate angular data table. Positive locators indicate 32 equi-probable binned data, zero locators indicate isotropic distributions and negative locators indicate tabulated angular data. Isotropic distributions have no further data entries.

Angular Law 1 is 32 equi-probable binned cosine angles. It has been the traditional method used to represent angular distributions. A positive value for the angular data locator indicates it contains Angular Law 1 data. This data consists of 33 cosine angle bin-boundaries which mark the points 1/32 apart in cumulative probability density. There location in the ANDP array is described in Table A-10. The cosine of the scattering angle is chosen by linear-linear interpolation of a randomly chosen point in the cumulative density space. This method's primary advantages are its speed of execution and small memory requirements. As memory and CPU power are much more readily available today than when this method was first conceived, it is no longer recommended for use.

The Angular Law 2 tabulated angular distribution was recently introduced [111] to more accurately reproduce highly anisotropic scattering distributions. It is generally

Table A-10. Description of Angular Law 1 32 equi-probable bin angular distribution table.

Location in XSS	Parameter	Description
ANDP(J)+LC(L)-1	N/A	First word of angular distribution data for incident energy point L LC(L) is greater than zero

ANDP(J)+LC(L)-1 ..
ANDP(J)+LC(L)+31
	CAB(M)
	M=1..33
		Cosine angle boundaries of the 32 equi-probable scattering bins

accepted that distributions with very anisotropic behavior, in particular very forward peaked distributions, are not well represented by Angular Law 1. Specifically, the detail of the high-probability area is well represented at the expense of the remainder of the distribution. Angular Law 2 distributions remedy this by allowing a tabular distribution of points. The description for Angular Law 2 data is given in Table A-11. The scattering angle is chosen based on the random sampling of the cumulative probability density with proper interpolation of its corresponding cosine value.

LDLWP Array

This array contains the entries for the energy distribution locators. An energy distribution locator is the location of the emission law data for the corresponding MT reaction relative to the DLWP locator. There must be one entry corresponding to each MTRP array entry and they should be located at ((XSS(I): I=LDLWP(J)..LDLWP(J)+NTRP-1): J=1..NTYPE). The elastic collision is now explicitly included here if the data are included. All locators must be positive integer values and indicate that emission distribution data are contained in the DLWP array.

Table A-11. Description of Angular Law 2 tabulated angular distribution table.

Location in XSS	Parameter	Description
ANDP(J)+|LC(L)|-1	N/A	First word of angular distribution data for incident energy point L LC(L) is less than zero
ANDP(J)+|LC(L)|-1	JJ	Interpolation parameter for cosine distribution ENDF defined interpolation parameters (Only histogram or linear-linear is allowed.)
ANDP(J)+|LC(L)|	NP	Number of points in the distribution

ANDP(J)+|LC(L)|+1 ..
ANDP(J)+|LC(L)|+NP
	CA(M)
	M=1..NP
		Cosine of the scattering angle

ANDP(J)+|LC(L)|+1+NP ..
ANDP(J)+|LC(L)|+1+2*NP
	PDF(M)
	M=1..NP
		Probability density function

ANDP(J)+|LC(L)|+2+2*NP ..
ANDP(J)+|LC(L)|+1+3*NP
	CDF(M)
	M=1..NP
		Cumulative density function

DLWP Array

The DLWP locator is the primary reference for finding emission distribution data. All emission distribution data for this secondary particle are listed sequentially in the DLWP array. Typically, the emission data described here are the energy spectra for the secondary particle. However, many new data evaluations are taking advantage of the correlated, energy and angle, emission distributions. If the angular distribution data are contained in the emission distribution, the corresponding LANDP entry must be negative one (-1). For all other cases, there must be a corresponding set of entries, as located by LANDP and ANDP, to describe the appropriate angular distribution. There must be at least one set of emission data for each reaction specified in the MTRP array.

Law Header. Each reaction has at least one emission distribution associated with it as described in the relevant law header information data. The entries in the law header information data are described in Table A-12. This header exists to facilitate describing reactions that require more than one sampling law to describe the emission parameters correctly. An example of this would be second chance fission. The law header containing the appropriate emission distribution(s) for the second reaction producing the third emission particle starts at the array location (XSS(DLWP(3)+LDLWP(2,3)-1). Here DLWP(3) is the twelfth IXS value for the third emission particle (IXS(12,3)) and LDLWP(2,3) is the second entry in the LDLWP array for the third emission particle (XSS(IXS(11,3)+2-1)).

Table A-12. Emission parameter law header information.

Location in XSS	Parameter	Description
DLWP(J)+LDLWP(K,J)-1	LNW_i	Location of next law header relative to DLWP(J) If LNW_i = 0, then this law is used regardless
DLWP(J)+LDLWP(K,J)	LAW_i	Name (#) of this law
DLWP(J)+LDLWP(K,J)+1	$IDAT_i$	Location of law dependent data relative to DLWP(J)
DLWP(J)+LDLWP(K,J)+2	NR	Number of interpolation regions; if NR = 0, NBT and INT are omitted and linear-linear interpolation is assumed for (E,P) pairs

DLWP(J)+LDLWP(K,J)+3 ..
DLWP(J)+LDLWP(K,J)+2+NR
	NBT(I)
	I=1..NR
		Starting index to which the corresponding interpolation parameter applies

DLWP(J)+LDLWP(K,J)+3+NR ..
DLWP(J)+LDLWP(K,J)+2+2*NR
	INT(I)
	I=1..NR
		ENDF defined interpolation parameter in each region

DLWP(J)+LDLWP(K,J)+3+2*NR	NE	Number of energies

DLWP(J)+LDLWP(K,J)+4+2*NR ..
DLWP(J)+LDLWP(K,J)+3+2*NR+NE
	E(I)
	I=1..NE
		Tabular energy points

DLWP(J)+LDLWP(K,J)+4+2*NR+NE ..
DLWP(J)+LDLWP(K,J)+3+2*NR+2*NE
	P(I)
	I=1..NE
		Probability of law validity

…	…	…
DLWP(J)+$IDAT_i$-1	LDAT	First word of law dependent data for LAW_i
…	…	…
DLWP(J)+LNW_i-1	LNW_{i+1}	First word of next law header
…	…	…

Once the appropriate law header is located, the specific emission distribution is determined based the probability of its validity at the incident energy. Because of the number of laws that can be used, the description of the remainder of this array can be daunting. The key to remember is that each reaction producing a given particle has a law header that provides the location of the appropriate sampling law. Each of the laws and their associated data is described in discussion below. The variable J always indicates the Jth emission particle and the variable K always indicated the Kth reaction producing that particle.

Energy Law 1. Energy Law 1 uses an equi-probable energy bin structure for sampling emission energies. Its data format is described in Table A-13. It is similar in nature to the Angular Law 1 and suffers from the same lack of fidelity for distributions with groupings of high-probability regions. It is recommended to use Energy Law 4, tabulated-energy-distribution, instead. Error checking should be done to ensure that each E_{out} table is a set of monotonically increasing real values ending with a value less than the corresponding E_{in}.

Energy Law 2. Energy Law 2 is primarily for discrete photon-emission lines produced by neutron interactions. Its data format is described in Table A-14. Its use is discouraged with photonuclear reactions though it is possible to use it as a discrete line emission, i.e. E_{out} = EG, for LP=0 or LP=1. Use of the LP=2 option is strongly discouraged for photonuclear interactions as it assumes simple neutron kinematics for computing the emission energy, i.e. E_{out} = EG + (AWR/(AWR+1)) * E_{in}. Error checking should be done to ensure that LP is in the range 0 to 2 and that EG is a positive real value.

Table A-13. Law dependent format for Energy Law 1 (Tabular Equi-probable Energy Bins).

Location in XSS	Parameter	Description

DLWP(J)+IDAT$_i$-1	LDAT(M)
	M=1..L
		Primary reference for law$_i$ dependent data (from law header)

LDAT(1)	NR	Number of interpolation regions (If NR = 0, NBT and INT are omitted and linear-linear interpolation is assumed)

LDAT(2) ..
LDAT(1+NR)
	NBT(N)
	N=1..NR
		Starting index to which the corresponding interpolation parameter applies

LDAT(2+NR) ..
LDAT(1+2*NR)
	INT(N)
	N=1..NR
		ENDF defined interpolation parameter in each region
		Only histogram or linear-linear

LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E_{in}(N) N=1..NE	List of incident energies for which E_{out} is tabulated
LDAT(3+2*NR+NE)	NET	Number of outgoing energies listed in each E_{out} table
LDAT(4+2*NR+NE) .. LDAT(3+2*NR+(NET+1)*NE)	E_{out1}(N) N=1..NET; E_{out2}(N) N=1..NET; … E_{outNE}(N) N=1..NET	E_{out} table have NET energies listed comprising the boundaries of (NET-1) equi-probable bins. Sampling uses a linear-linear interpolation between bin boundaries.

Energy Laws 3 & 33. Energy Law 3 and 33 are inelastic level scattering. The data format is described in Table A-15. Law 3 indicates neutron incident, neutron emission. Law 33 indicates any combination of particles incident and emitted. Its use is allowed for photonuclear interactions though the parameters must be chosen for photonuclear kinetics instead of neutron kinetics. Sampling of this law follows the simple formula of E_{out} = LDAT(2) * (E_{in} – LDAT(1)) in the center-of-mass system. Error checking should be done to ensure that the corresponding TYRP entry is negative one.

Table A-14. Law dependent format for Energy Law 2 (Discrete Emission Energy).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..2	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	LP	Indicator of whether the emission particle is primary or non-primary
LDAT(2)	EG	Emission energy (if LP=0 or LP=1) Binding energy (LP=2)

Table A-15. Law dependent format for Energy Law 3/33 (Level Scattering).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..2	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	MT	For neutron scattering $((A+1)/A) * \|Q\|$
LDAT(2)	CR	For neutron scattering $(A/(A+1))^2$

Energy Laws 4, 44 & 61. Energy Laws 4, 44 and 61 are tabular energy distributions. The common portion of the data format for this set of laws is described in Table A-16. The tabular energy distribution provides the most flexibility of the energy laws. Any energy-emission-spectral shape can be formed provided enough grid points are used. Energy sampling is accomplished by determining the two closest incident energy grid points, sampling a random cumulative probability to find the emission energy from each grid and using histogram or linear-linear interpolation between them. If discrete lines are used in the emission grid, correspondence of these lines must be maintained between grids. Error checking should be done to ensure only histogram or linear-linear interpolation between distributions is used and to ensure that discrete lines are correctly handled. The format of the tabular distribution itself is dependant on which law is specified.

Table A-16. Law dependent format for Energy Laws 4, 44 and 61 (Tabular Energy Distributions).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions (If NR=0, NBT and INT are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	ENDF defined interpolation parameter in each region; only histogram and linear-linear interpolation are allowed
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE) .. LDAT(2+2*NR+2*NE)	LTB(N) N=1..NE	Locators for tabular distributions relative to DLWP(J)

Energy Law 4 contains only energy-emission information. Its data format is described in Table A-17. Angular distribution data must be included using the ANDP array. Sampling is achieved by choosing a random number between zero and one, finding the cumulative bin in which the random number falls and taking the corresponding energy, appropriately interpolated. Error checking should be done to ensure that the largest emission energy for any endothermic reactions is not more than its corresponding incident energy, that the probability density function integrates to the cumulative and that the cumulative density is monotonically increasing from zero to one.

The tabular distribution format for Energy Law 44 expands the Law 4 format to include the Kalbach parameters for each emission energy. Its data format is described in Table A-18. The parameters are used to compute the angular distribution based on the

Table A-17. Tabular distribution format for Energy Law 4 (Tabular energy distribution).

Location in XSS	Parameter	Description
DLWP(J)+LTB(N)-1	N/A	First word of tabular distribution data for incident energy point N
DLWP(J)+LTB(N)-1	INTT'	Overloaded variable: Interpolation scheme for distribution mod(INTT',10) = 1 -> Histogram mod(INTT',10) = 2 -> Lin.-Lin. Number of discrete points in distribution ND = int(INTT' / 10)
DLWP(J)+LTB(N)	NP	Number of points in the distribution
DLWP(J)+LTB(N)+1 .. DLWP(J)+LTB(N)+NP	E_{out}(O) O=1..NP	Emission energy grid
DLWP(J)+LTB(N)+1+NP .. DLWP(J)+LTB(N)+2*NP	PDF(O) O=1..NP	Probability density function
DLWP(J)+LTB(N)+1+2*NP .. DLWP(J)+LTB(N)+3*NP	CDF(O) O=1..NP	Cumulative density function

Table A-18. Tabular distribution format for Energy Law 44 (Kalbach correlated energy/angle distribution).

Location in XSS	Parameter	Description
DLWP(J)+LTB(N)-1	N/A	First word of tabular distribution data for incident energy point N
DLWP(J)+LTB(N)-1	INTT'	Overloaded variable: Interpolation scheme for distribution mod(INTT',10) = 1 -> Histogram mod(INTT',10) = 2 -> Lin.-Lin. Number of discrete points in distribution ND = int(INTT' / 10)
DLWP(J)+LTB(N)	NP	Number of points in the distribution
DLWP(J)+LTB(N)+1 .. DLWP(J)+LTB(N)+NP	E_{out}(O) O=1..NP	Emission energy grid
DLWP(J)+LTB(N)+1+NP .. DLWP(J)+LTB(N)+2*NP	PDF(O) O=1..NP	Probability density function
DLWP(J)+LTB(N)+1+2*NP .. DLWP(J)+LTB(N)+3*NP	CDF(O) O=1..NP	Cumulative density function
DLWP(J)+LTB(N)+1+3*NP .. DLWP(J)+LTB(N)+4*NP	R(O) O=1..NP	Kalbach pre-compound fraction r
DLWP(J)+LTB(N)+1+4*NP .. DLWP(J)+LTB(N)+5*NP	A(O) O=1..NP	Kalbach-Chadwick angular distribution slope value a

Kalbach-87 formalism [50,51]. For photonuclear reactions, the slope value must be computed at the time the table was produced according to Chadwick's modification [11] to Kalbach's original formalism. Sampling of Law 44 emission energy is analogous to Law 4. Error checking should be done to ensure that the value for all R entries is in the range from zero to one and that the value for all a entries is a non-negative real number.

Currently, the emission angle in MCNP is sampled using Kalbach's original formalism. It is under discussion whether a new law should be added to address the fact that secondary particle emission from photonuclear multi-step compound reactions is more correctly represented as isotropic. Due to the relatively small a values typical of photonuclear emissions, this would be a small, less than five percent, correction.

The tabular distribution format for Energy Law 61 expands the Law 4 format to include a pointer to an angular distribution. Its data format is described in Table A-19. Positive angular distribution locators indicate a tabular angular distribution is available and it is sampled by the same algorithm as Angular Law 2 data. Table A-20 describes the data format for Law 61 tabular angular distribution data. A zero value for a locator indicates an isotropic distribution. Angular Law 1 data is not allowed. All stipulations for Law 4 still apply to the energy distribution and all stipulations for Angular Law 2 data apply to relevant angular information. Error checking should also be done to ensure that the value of all angular locators is either zero or a positive integer value.

Energy Law 5. Energy Law 5 is a temperature scaled equi-probable binned function. Its data format is described in Table A-21. At the current time, no Los Alamos National Laboratory supported library uses this law. The emission energy is computed

Table A-19. Tabular distribution format for Energy Law 61 (Correlated tabular energy/angle distribution).

Location in XSS	Parameter	Description
DLWP(J)+LTB(N)-1	N/A	First word of tabular distribution data for incident energy point N
DLWP(J)+LTB(N)-1	INTT'	Overloaded variable: Interpolation scheme for distribution mod(INTT',10) = 1 -> Histogram mod(INTT',10) = 2 -> Lin.-Lin. Number of discrete points in distribution ND = int(INTT' / 10)
DLWP(J)+LTB(N)	NP	Number of points in the distribution
DLWP(J)+LTB(N)+1 .. DLWP(J)+LTB(N)+NP	E_{out}(O) O=1..NP	Emission energy grid
DLWP(J)+LTB(N)+1+NP .. DLWP(J)+LTB(N)+2*NP	PDF(O) O=1..NP	Probability density function
DLWP(J)+LTB(N)+1+2*NP .. DLWP(J)+LTB(N)+3*NP	CDF(O) O=1..NP	Cumulative density function
DLWP(J)+LTB(N)+1+3*NP .. DLWP(J)+LTB(N)+4*NP	LAD(O) O=1..NP	Angular distribution locators

Table A-20. Tabular angular distribution format for Energy Law 61.

Location in XSS	Parameter	Description
DLWP(J)+LAD(O)-1	N/A	First word of tabular angular distribution data for emission energy O
DLWP(J)+LAD(O)-1	JJ	Interpolation parameter for cosine distribution (Only histogram or linear-linear allowed)
DLWP(J)+LAD(O)	NP	Number of points in the distribution
DLWP(J)+LAD(N)+1 .. DLWP(J)+LAD(N)+NP	CBB(P) P=1..NP	Cosine bin boundaries
DLWP(J)+LAD(N)+1+NP .. DLWP(J)+LAD(N)+2*NP	PDF(P) P=1..NP	Probability density function
DLWP(J)+LAD(N)+1+2*NP .. DLWP(J)+LAD(N)+3*NP	CDF(P) P=1..NP	Cumulative density function

Table A-21. Law dependent format for Energy Law 5 (General Spectrum).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions (If NR=0, NBT and INT are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	ENDF defined interpolation parameter in each region
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E_{in}(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE) .. LDAT(2+2*NR+2*NE)	T(N) N=1..NE	Temperature based on incident energy
LDAT(3+2*NR+2*NE)	NET	Number of X's tabulated
LDAT(4+2*NR+2*NE) .. LDAT(3+2*NR+2*NE+NET)	X(O) O=1..NET	Tabulated probabilistic function

by multiplying a nuclear temperature based on the incident energy times a randomly sampled equi-probable binned emission probability. This law has been superceded by the use of Law 4 distributions. It is not recommended for use for any purpose.

Energy Law 7. Energy Law 7 is a simple Maxwell-fission spectrum as defined in File 5 of ENDF-6. Its data format is described in Table A-22. It is appropriate for all fission reactions including photonuclear fission. The sampled emission energy is based on the function f(E->E_{out}) = C sqrt(E_{out}) exp(-E_{out} / T(E)). The emission energy is bounded by the range zero to the incident energy minus the restriction energy.

Energy Law 9. Energy Law 9 is an evaporation spectrum as defined in File 5 of ENDF-6. Its data format is described in Table A-23. It is appropriate for nucleon

Table A-22. Law dependent format for Energy Law 7 (Simple Maxwell Fission Spectrum).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions (If NR=0, NBT and INT are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	ENDF defined interpolation parameter in each region
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E$_{in}$(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE) .. LDAT(2+2*NR+2*NE)	T(N) N=1..NE	Temperature based on incident energy
LDAT(3+2*NR+2*NE)	U	Restriction energy

Table A-23. Law dependent format for Energy Law 9 (Evaporation Spectrum).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions (If NR=0, NBT and INT are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	ENDF defined interpolation parameter in each region
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E$_{in}$(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE) .. LDAT(2+2*NR+2*NE)	T(N) N=1..NE	Temperature based on incident energy
LDAT(3+2*NR+2*NE)	U	Restriction energy

emission from a compound nucleus decay. The sampled emission energy is based on the function f(E->E_{out}) = C E_{out} exp(-E_{out} / T(E)). The emission energy is bounded by the range zero to the incident energy minus the restriction energy.

Energy Law 11. Energy Law 11 is an energy-dependent Watt-spectrum as defined in File 5 of ENDF-6. Its data format is described in Table A-24. The sampled emission energy is based on the function f(E->E_{out}) = C exp(-E_{out} / a(E)) sinh(sqrt(-E_{out} b(E))). The emission energy is bounded by the range zero to the incident energy minus the restriction energy.

Energy Law 22. Energy Law 22 is a tabular linear function from UK Law 2 (no reference currently available). Its data format is described in Table A-25 and Table A-26. It is not recommended for use in photonuclear tables. It is similar to Law 1 and Law 4 in that an incident energy is used to sample a tabulated distribution. However, the table is always chosen as the next distribution under the incident energy and no interpolation is done. Emission energy is sampled by choosing a random number in the range zero to one, finding the cumulative bin just below the sample and using the corresponding constant and temperature in the formula E_{out} = CM * (E_{in} – T).

Energy Law 24. Energy Law 24 is a tabular energy multiplier distribution from UK Law 6 (no reference currently available). Its data format is described in Table A-27. It is not recommended for use by photonuclear tables. It is similar to Law 1 and Law 4 in that an incident energy is used to sample a tabulated distribution. However, the table is always chosen as the next distribution under the incident energy and no interpolation is done. Emission energy is sampled by choosing a random number in the range zero to

Table A-24. Law dependent format for Energy Law 11 (Energy Dependent Watt Spectrum).

Location in XSS	Parameter	Description

DLWP(J)+IDAT$_i$-1	LDAT(M)
	M=1..L
		Primary reference for law$_i$ dependent data (from law header)

LDAT(1)	NR$_a$	Number of interpolation regions (If NR$_a$=0, NBT$_a$ and INT$_a$ are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR$_a$)	NBT$_a$(N) N=1..NR$_a$	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR$_a$) .. LDAT(1+2*NR$_a$)	INT$_a$(N) N=1..NR$_a$	ENDF defined interpolation parameter in each region
LDAT(2+2*NR$_a$)	NE$_a$	Number of incident energies tabulated
LDAT(3+2*NR$_a$) .. LDAT(2+2*NR$_a$+NE$_a$)	E$_a$(N) N=1..NE$_a$	List of incident energies

LDAT(3+2*NR$_a$+NE$_a$) ..
LDAT(2+2*NR$_a$+2*NE$_a$)
	a(N)
	N=1..NE$_a$
		Energy dependent parameter a

LDAT(3+2*NR$_a$+2*NE$_a$) Let W = 3+2*NR$_a$+2*NE$_a$	NR$_b$	Number of interpolation regions (If NR$_b$=0, NBT$_b$ and INT$_b$ are omitted and linear-linear interpolation is assumed)
LDAT(W+1) .. LDAT(W+NR$_b$)	NBT$_b$(N) N=1..NR$_b$	Starting index to which the corresponding interpolation parameter applies
LDAT(W+1+NR$_b$) .. LDAT(W+2*NR$_b$)	INT$_b$(N) N=1..NR$_b$	ENDF defined interpolation parameter in each region
LDAT(W+1+2*NR$_b$)	NE$_b$	Number of incident energies tabulated
LDAT(W+2+2*NR$_b$) .. LDAT(W+1+2*NR$_b$+NE$_b$)	E$_b$(N) N=1..NE$_b$	List of incident energies

LDAT(W+2+2*NR$_b$+NE$_b$) ..
LDAT(W+1+2*NR$_b$+2*NE$_b$)
	b(N)
	N=1..NE$_b$
		Energy dependent parameter b

LDAT(W+2+2*NR$_b$+2*NE$_b$)	U	Restriction energy

Table A-25. Law dependent format for Energy Law 22 (Tabular Linear Functions).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(1)	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	Interpolation parameter in each region (ignored; histogram assumed)
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	LTB(N) N=1..NE	Locators for tabular distributions relative to DLWP(J)

Table A-26. Tabular distribution format for Energy Law 22.

Location in XSS	Parameter	Description
DLWP(J)+LTB(N)-1	N/A	First word of tabular distribution data for incident energy point N
DLWP(J)+LTB(N)	NP	Number of points in the distribution
DLWP(J)+LTB(N)+1 .. DLWP(J)+LTB(N)+NP	P(O) O=1..NP	Cumulative probability bin boundaries
DLWP(J)+LTB(N)+1+NP .. DLWP(J)+LTB(N)+2*NP	T(O) O=1..NP	Temperature for bin
DLWP(J)+LTB(N)+1+2*NP .. DLWP(J)+LTB(N)+3*NP	CM(O) O=1..NP	Constant multiplier for bin

Table A-27. Law dependent format for Energy Law 24 (Tabular Energy Multiplier).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	interpolation parameter in each region (ignored; assumed histogram)
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E_{in}(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE)	NET	Number of energies listed in each E_{out} table
LDAT(4+2*NR+NE) .. LDAT(3+2*NR+(NET+1)*NE)	T_1(N) N=1..NET … T_{NE}(N) N=1..NET	Multiplier table have NET listings comprising the boundaries of (NET-1) equi-probable bins. Sampling uses a linear-linear interpolation between bin boundaries.

one, using linear-linear interpolation within the equi-probable multiplier bin and computing the emission energy from the formula $E_{out} = T\ E_{in}$.

Energy Law 66. Energy Law 66 is an N-body phase-space distribution from File 6 Law 6 of ENDF-6. Its data format is described in Table A-28. It is not recommended for use with photonuclear reactions due to the non-Newtonian nature of photon interactions. Full details of this sampling scheme are found in Chapter 2 and Appendix F of the MCNP Users Guide [3].

Energy Law 67. Energy Law 67 is the laboratory system, correlated angle/energy law from File 6 Law 7 of ENDF-6. Its data format is described in Table A-29. The angular cosine data and subsequent energy distributions are described in Table

A-30 and Table A-31, respectively. This distribution first samples an appropriate cosine an then uses the tabular energy distribution corresponding to the sampled angle. This law is not recommended for photonuclear data.

Table A-28. Law dependent format for Energy Law 66 (N-body Phase Space Distribution).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..2	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NPSX	Number of bodies in the phase space
LDAT(2)	A_p	Total mass ratio for the NPSX particles

Table A-29. Law dependent format for Energy Law 67 (Tabulated Angle/Energy).

Location in XSS	Parameter	Description
DLWP(J)+IDAT$_i$-1	LDAT(M) M=1..L	Primary reference for law$_i$ dependent data (from law header)
LDAT(1)	NR	Number of interpolation regions (If NR=0, NBT and INT are omitted and linear-linear interpolation is assumed)
LDAT(2) .. LDAT(1+NR)	NBT(N) N=1..NR	Starting index to which the corresponding interpolation parameter applies
LDAT(2+NR) .. LDAT(1+2*NR)	INT(N) N=1..NR	ENDF defined interpolation parameter in each region Only histogram and linear-linear interpolation are allowed
LDAT(2+2*NR)	NE	Number of incident energies tabulated
LDAT(3+2*NR) .. LDAT(2+2*NR+NE)	E(N) N=1..NE	List of incident energies
LDAT(3+2*NR+NE) .. LDAT(2+2*NR+2*NE)	LTB(N) N=1..NE	Locators for tabular cosine distributions relative to DLWP(J)

Table A-30. Tabular distribution format for Energy Law 67.

Location in XSS	Parameter	Description
DLWP(J)+LTB(N)-1	N/A	First word of tabular distribution data for incident energy point N

DLWP(J)+LTB(N)-1	INTMU	Interpolation scheme for distribution
		INTMU = 1 -> Histogram
		INTMU = 2 -> Linear-linear

DLWP(J)+LTB(N)	NMU	Number of points in the distribution

DLWP(J)+LTB(N)+1 ..
DLWP(J)+LTB(N)+NMU
	XMU(O)
	O=1..NMU
		Secondary cosines

DLWP(J)+LTB(N)+1+NMU ..
DLWP(J)+LTB(N)+2*NMU
	LMU(O)
	O=1..NMU
		Locators for secondary cosine energy distribution relative to DLWP(J)

Table A-31. Tabular energy distribution format for Energy Law 67.

Location in XSS	Parameter	Description
DLWP(J)+LMU(O)-1	N/A	First word of tabular energy distribution for secondary cosine point O

DLWP(J)+LMU(O)-1	INTEP	Interpolation scheme for distribution
		INTEP = 1 -> Histogram
		INTEP = 2 -> Linear-linear

DLWP(J)+LMU(O)	NPEP	Number of points in the distribution

DLWP(J)+LMU(O)+1 ..
DLWP(J)+LMU(O)+NPEP
	EP(P)
	P=1..NPEP
		Secondary energy grid

DLWP(J)+LMU(O)+1+NPEP ..
DLWP(J)+LMU(O)+2*NPEP
	PDF(P)
	P=1..NPEP
		Probability density function

DLWP(J)+LMU(O)+1+2*NPEP ..
DLWP(J)+LMU(O)+3*NPEP
	CDF(P)
	P=1..NPEP
		Cumulative density function

APPENDIX B
MKPNT PROCESSING CODE

Introduction

This appendix contains the source code for the MKPNT data processing code. It’s functionality is discussed in depth in Chapter 4. In order to build the executable code, you must have an ANSI C compiler and the Unix make utility. The appropriate Makefile is included here such that the code is built with the command “make mkpnt”. It has only been tested on a Sun system using the standard sun compiler package. The files are listed with the filename as the heading followed directly by the source code for that file.

mkpnt.c

```
#include "endf6.h"
#include "acepnData.h"

void acepnFromENDF( endfMaterialInformation *mi, aceTable *table );
void acePrintNTable( char *filename, aceTable *table );


/*******************************************************************
** mkpnt
**
** Make Photonuclear Table
** Main Program
**
** Reads in an endf photonuclear file
**
** Creates an MCNP ace format photonuclear table (acepn)
**
*/
int main( int argc, char **argv )
{
  endfMaterialInformation *mi
    = (endfMaterialInformation*)malloc( sizeof(endfMaterialInformation) );

  aceTable *table
    = (aceTable*)malloc( sizeof(aceTable) );


  /*
  ** check command line arguments
  */
  if( argc != 3 ) {
    printf( "mkpnt: USAGE\n" );
    printf( "       @prompt> mkpnt endfFilenameIn aceFilenameOut\n" );
    printf( "       expects valid endf photonuclear file in filename in\n" );
    printf( "         will take the first material encountered in endf file\n" );
    printf( "       creates (or appends to) the ace library file filename out\n" );
    exit( -1 );
  }


  /*
  ** read the whole endf file into memory
  */
  endfReadMaterialFromFile( argv[1], mi );

  /*
  ** create a new photonuclear table
  **   fill the information from the endf information
  */
  afeMakeNTable( mi, table );


  /*
  ** print the new acepn table
  */
  acePrintNTable( argv[2], table );

  /*
  ** if no exit errors, print success
  */
  printf( "successfully processed \"%s\" into \"%s\"\n", argv[1], argv[2] );

}
```

acepnData.h

```
#ifndef acepnData_h
#define acepnData_h


#include <stdio.h>
#include <stdlib.h>

#define  NUMBER_IXS_ENTRIES  12

/*
** Structure: aceLaw4Distribution
*/
typedef struct acelaw4distribution {
  int      Offset;
  double   IncidentEnergy;
  int      InterpolationScheme;
  int      NumberOfDiscreteEmissions;
  int      NumberOfPoints;
  double  *EmissionEnergy;
  double  *Probability;
  double  *CumulativeProbability;
} aceLaw4Distribution;

/*
** Structure: aceLaw4
*/
typedef struct acelaw4 {
  int                   NumberOfRegions;
  int                  *NumberOfPointsInRegion;
  int                  *InterpolationSchemeInRegion;
  int                   NumberOfIncidentEnergies;
  aceLaw4Distribution  *Distribution;
} aceLaw4;

/*
** Structure: aceLaw7
*/
typedef struct acelaw7 {
  int      NumberOfRegions;
  int     *NumberOfPointsInRegion;
  int     *InterpolationSchemeInRegion;
  int      NumberOfIncidentEnergies;
  double  *IncidentEnergy;
  double  *Temperature;
  double   RestrictionEnergy;
} aceLaw7;

/*
** Structure: aceLaw9
*/
typedef struct acelaw9 {
  int      NumberOfRegions;
  int     *NumberOfPointsInRegion;
  int     *InterpolationSchemeInRegion;
  int      NumberOfIncidentEnergies;
  double  *IncidentEnergy;
  double  *Temperature;
  double   RestrictionEnergy;
} aceLaw9;

/*
** Structure: aceLaw44Distribution
*/
typedef struct acelaw44distribution {
  int      Offset;
  double   IncidentEnergy;
  int      InterpolationScheme;
  int      NumberOfDiscreteEmissions;
  int      NumberOfPoints;
  double  *EmissionEnergy;
  double  *Probability;
  double  *CumulativeProbability;
  double  *PrecompoundFraction;
  double  *AngularDistributionSlope;
} aceLaw44Distribution;

/*
** Structure: aceLaw44
*/
typedef struct acelaw44 {
  int                     NumberOfRegions;
  int                    *NumberOfPointsInRegion;
  int                    *InterpolationSchemeInRegion;
  int                     NumberOfIncidentEnergies;
  aceLaw44Distribution  *Distribution;
} aceLaw44;

/*
** Structure: aceLawInformation
*/
typedef struct acelawinformation {
  int      LocationOfNextLaw;
  int      OffsetToLawData;
  int      Number;
  char    *Name;
  int      NumberOfRegions;
  int     *NumberOfPointsInRegion;
  int     *InterpolationSchemeInRegion;
  int      NumberOfEnergies;
  double  *Energy;
  double  *Probability;
  void    *LawData;
} aceLawInformation;
```

```
/*
** Structure: aceEmissionData
*/
typedef struct aceemissiondata {
  int                 Offset;
  int                 CoordinateSystem;
  int                 AngularInformationType;
  void               *AngularInformation;
  int                 NumberOfEnergyLaws;
  aceLawInformation  *LawInformation;
} aceEmissionData;

/*
** Structure: aceMTInformation
*/
typedef struct acemtinformation {
  int      Number;
  char     Name[100];
  char     Reaction[100];
  int      NumberOfProducts;
  int     *ProductZA;
  int     *YieldOfProduct;
  int     *NumberOfEnergies;  /* the energy terms are pointers to the */
  double  *Energy;            /* data located at the acepnData level */
  int      StartingIndex;
  int      NumberOfEntries;
  double  *CrossSection;
  double   Q;
} aceMTInformation;

/*
** Structure: aceYieldInformation
*/
typedef struct aceyieldinformation {
  int      NumberOfRegions;
  int     *NumberOfPointsInRegion;
  int     *InterpolationSchemeInRegion;
  int      NumberOfYields;
  double  *Energy;
  double  *Yield;
} aceYieldInformation;

/*
** Structure: aceMTReference
*/
typedef struct acemtreference {
  int                   Type;
  int                   Offset;
  aceMTInformation     *MT;
  aceYieldInformation  *Yield;
  aceEmissionData      *Emit;
} aceMTReference;

/*
** Structure: aceProduct
*/
typedef struct aceproduct {
  int                ZA;
  int                IPT;
  char              *Name;
  char              *Symbol;
  int                IXS[NUMBER_IXS_ENTRIES];
  int                StartingIndex;
  int                NumberOfEntries;
  double            *ProductionCrossSection;
  double            *PartialHeatingNumber;
  int                NumberOfReactions;
  aceMTReference   **MTReference;
} aceProduct;
```

```
/*
** Structure: acepnData
*/
typedef struct acepndata {
  int                NumberOfEnergies;
  double            *Energy;
  aceMTInformation  *TotalCrossSection;
  aceMTInformation  *NonelasticCrossSection;
  aceMTInformation  *ElasticCrossSection;
  double            *TotalHeatingNumber;
  int                NumberOfMTs;
  int               *MTLocator;
  aceMTInformation **MT;
  int                NumberOfProducts;
  aceProduct       **Product;
} acepnData;

/*
** Structure: aceTable
*/
typedef struct acetable {
  char     TableIdentifier[11];           /* XSDIR Entry  1, XSLIB Entry 1 */
  int      ZA;
  int      Z;
  int      A;
  char     ThermalTableName[7];
  int      LibraryNumber;
  char     TableType;
  char    *IncidentParticleName;
  double   AtomicWeightRatio;             /* XSDIR Entry  2, XSLIB Entry 2 */
  char     LibraryFileName[9];            /* XSDIR Entry  3 */
  char     LibraryAccessRoute[71];        /* XSDIR Entry  4 */
  int      LibraryFileType;               /* XSDIR Entry  5 */
  int      AddressInLibrary;              /* XSDIR Entry  6 */
  int     *LengthXSS;                     /* XSDIR Entry  7, Pointer To NXS[0] */
  int      BinaryRecordLength;            /* XSDIR Entry  8 */
  int      EntriesPerBinaryRecord;        /* XSDIR Entry  9 */
  double   NeutronProcessingTemperature;  /* XSDIR Entry 10, XSLIB Entry 3 */
  char     ProcessDate[11];               /* XSLIB Entry  4 */
  char     Comment[71];                   /* XSLIB Entry  5 */
  char     MaterialIdentifier[11];        /* XSLIB Entry  6 */
  int      MaterialNumber;
  int      ZAs[16];                       /* XSLIB Entries 7 - 37, odd */
  double   AtomicWeightRatios[16];        /* XSLIB Entries 8 - 38, even */
  int      NXS[16];                       /* XSLIB Entries 39 - 54 */
  int      JXS[32];                       /* XSLIB Entries 55 - 86 */
  fpos_t   StartData;
  void    *Data;                          /* XSLIB Entries 87 - (87 + LengthXSS) */
} aceTable;


#endif
```

acepnIO.c

```
#include "acepnData.h"

void acePrintNTable( char *filename, aceTable *table );
void acePrintXSS( FILE *fACE, char ft, char dt,
                  int number, void *data, int *count );

/*****************************************************************
** acePrintNTable
*/
void acePrintNTable( char *filename, aceTable *table )
{
  int  i, j, k, l;
  int  count = 1;
  int  tempi;
  int  zeroi = 0;
```

```
double tempd;
double zerod = 0.0;

FILE  *fACE;

acepnData          *ndata;
aceProduct         *prod;
aceMTReference     *mtref;
aceEmissionData    *edata;
aceLawInformation  *lawinfo;
aceLaw4            *law4;
aceLaw7            *law7;
aceLaw9            *law9;
aceLaw44           *law44;


/*
** open the ace library for writing
*/
fACE = fopen( filename, "a" );
if( !fACE ) {
  printf( "ERROR: acePrintNTable:\n" );
  printf( "         cannot open file \"%s\" for writing\n",
          filename );
  exit( -1 );
}

/*
** print the header information
*/
fprintf( fACE, "%10s", table->TableIdentifier );
fprintf( fACE, "%12lf", table->AtomicWeightRatio );
fprintf( fACE, "%12lf", table->NeutronProcessingTemperature );
fprintf( fACE, "%10s\n", table->ProcessDate );

fprintf( fACE, "%-70s", table->Comment );
fprintf( fACE, "%10s\n", table->MaterialIdentifier );

/*
** print the atomic weight section as all zeros
** (this section is no longer used)
*/
for( i = 1; i <= 16; i++ ) {
  fprintf( fACE, "%7d%11lf", zeroi, zerod );
  if( i % 4 == 0 )
    fprintf( fACE, "\n" );
}

/*
** print the nxs array
*/
for( i = 1; i <= 16; i++ ) {
  fprintf( fACE, "%9d", table->NXS[i-1] );
  if( i % 8 == 0 )
    fprintf( fACE, "\n" );
}

/*
** print the jxs array
*/
for( i = 1; i <= 32; i++ ) {
  fprintf( fACE, "%9d", table->JXS[i-1] );
  if( i % 8 == 0 )
    fprintf( fACE, "\n" );
}

ndata = (acepnData*)table->Data;


/*
```

```
** *** ESZ JXS(1) ***
** print the main energy listing
*/
acePrintXSS( fACE, 'a', 'd', ndata->NumberOfEnergies,
            ndata->Energy, &count );

/*
** *** TOT JXS(2) ***
** print the total cross section table
*/
if( ndata->TotalCrossSection == NULL ) {
  printf( "ERROR: did not find MT 1 (total cross section)" );
  exit( -1 );
}
acePrintXSS( fACE, 'a', 'd', ndata->NumberOfEnergies,
            ndata->TotalCrossSection->CrossSection, &count );

/*
** *** NON JXS(3) ***
** if different from total, print the total non-elastic
**   cross section table
*/
if( table->JXS[2] != 0 && table->JXS[2] != table->JXS[1] ) {
  if( ndata->NonelasticCrossSection == NULL ) {
    printf("ERROR: did not find MT 3 (non-elastic cross section)");
    exit( -1 );
  }
  acePrintXSS( fACE, 'a', 'd', ndata->NumberOfEnergies,
              ndata->NonelasticCrossSection->CrossSection, &count );
}

/*
** *** ELS JXS(4) ***
** if exists, print the elastic cross section table
*/
if( table->JXS[3] != 0 ) {
  if( ndata->ElasticCrossSection == NULL ) {
    printf( "ERROR: did not find MT 2 (elastic cross section)" );
    exit( -1 );
  }
  acePrintXSS( fACE, 'a', 'd', ndata->NumberOfEnergies,
              ndata->ElasticCrossSection->CrossSection, &count );
}

/*
** *** HNT JXS(5) ***
** if exists, print the total heating number table
*/
if( table->JXS[4] != 0 ) {
  if( ndata->TotalHeatingNumber == NULL ) {
    printf( "ERROR: did not find total heating number" );
    exit( -1 );
  }
  acePrintXSS( fACE, 'a', 'd', ndata->NumberOfEnergies,
              ndata->TotalHeatingNumber, &count );
}

/*
** *** MTR JXS(6) ***
** print the MT numbers
*/
for( i = 0; i < ndata->NumberOfMTs; i++ )
  acePrintXSS( fACE, 'a', 'i', 1,
             &(ndata->MT[i]->Number), &count );

/*
** *** LQR JXS(7) ***
** print the Q value table
*/
for( i = 0; i < ndata->NumberOfMTs; i++ )
  acePrintXSS( fACE, 'a', 'd', 1, &(ndata->MT[i]->Q), &count );
```

```
/*
** *** LSIG JXS(8) ***
** *** SIG JXS(9) ***
** print the cross section offset entries and data
*/
acePrintXSS( fACE, 'a', 'i', ndata->NumberOfMTs,
            ndata->MTLocator, &count );

for( i = 0; i < ndata->NumberOfMTs; i++ ) {
  acePrintXSS( fACE, 'a', 'i', 1,
              &(ndata->MT[i]->StartingIndex), &count );
  acePrintXSS( fACE, 'a', 'i', 1,
              &(ndata->MT[i]->NumberOfEntries), &count );
  acePrintXSS( fACE, 'a', 'd', ndata->MT[i]->NumberOfEntries,
              &(ndata->MT[i]->CrossSection[ndata->MT[i]->StartingIndex-1]),
              &count );
}

/*
** *** IXSA JXS(10) ***
** print the IXS array
*/
for( i = 0; i < ndata->NumberOfProducts; i++ )
  acePrintXSS( fACE, 'a', 'i', NUMBER_IXS_ENTRIES,
              &(ndata->Product[i]->IXS[0]), &count );

/*
** *** IXS JXS(11) ***
** print the IXS data block
*/
for( i = 0; i < ndata->NumberOfProducts; i++ ) {

  prod = (aceProduct*)ndata->Product[i];

  /*
  ** Don't print any entries for IXS value entries
  **
  ** *** IPT IXS(1) ***
  ** particle ipt number
  **
  ** *** NTRP IXS(2) ***
  ** number of reactions producing this particle
  */

  /*
  ** *** PXS IXS(3) ***
  ** print the production cross section table
  */
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->StartingIndex), &count );
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->NumberOfEntries), &count );
  acePrintXSS( fACE, 'a', 'd', prod->NumberOfEntries,
              &(prod->ProductionCrossSection[prod->StartingIndex-1]),
              &count );

  /*
  ** *** PHN IXS(4) ***
  ** if exists, print the partial heating number table
  */
  if( prod->IXS[3] != 0 && table->JXS[4] != 0 ) {
    acePrintXSS( fACE, 'a', 'i', 1,
                &(prod->StartingIndex), &count );
    acePrintXSS( fACE, 'a', 'i', 1,
                &(prod->NumberOfEntries), &count );
    acePrintXSS( fACE, 'a', 'd', prod->NumberOfEntries,
                &(prod->PartialHeatingNumber[prod->StartingIndex-1]),
                &count );
  }
```

```
/*
** *** MTRP IXS(5) ***
** print the mt reference listing
*/
for( j = 0; j < prod->NumberOfReactions; j++ )
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->MTReference[j]->MT->Number), &count );

/*
** *** TYRP IXS(6) ***
** print the coordinate system listing
*/
for( j = 0; j < prod->NumberOfReactions; j++ )
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->MTReference[j]->Emit->CoordinateSystem),
              &count );

/*
** *** LSIGP IXS(7) ***
** *** SIGP IXS(8) ***
** print the cross section or yield data offsets
**   for each reaction and its data values
*/
for( j = 0; j < prod->NumberOfReactions; j++ )
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->MTReference[j]->Offset), &count );

for( j = 0; j < prod->NumberOfReactions; j++ ) {

  mtref = prod->MTReference[j];

  acePrintXSS( fACE, 'a', 'i', 1,
              &(mtref->Type), &count );

  switch( mtref->Type ) {

  case 5:
  case 12:
  case 16:
    acePrintXSS( fACE, 'a', 'i', 1,
                &(mtref->MT->Number), &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(mtref->Yield->NumberOfRegions), &count );

    acePrintXSS( fACE, 'a',  'i', mtref->Yield->NumberOfRegions,
                mtref->Yield->NumberOfPointsInRegion, &count );

    acePrintXSS( fACE, 'a',  'i', mtref->Yield->NumberOfRegions,
                mtref->Yield->InterpolationSchemeInRegion, &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(mtref->Yield->NumberOfYields), &count );

    acePrintXSS( fACE, 'a',  'd', mtref->Yield->NumberOfYields,
                mtref->Yield->Energy, &count );

    acePrintXSS( fACE, 'a',  'd', mtref->Yield->NumberOfYields,
                mtref->Yield->Yield, &count );

    break;

  case 13:
    acePrintXSS( fACE, 'a', 'i', 1,
                &(mtref->MT->StartingIndex), &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(mtref->MT->NumberOfEntries), &count );

    acePrintXSS( fACE, 'a', 'd', mtref->MT->NumberOfEntries,
                &(mtref->MT->CrossSection[mtref->MT->StartingIndex-1]),
                &count );

    break;

  default:
    printf( "WARNING: trying to print LSIG type %d\n", mtref->Type );

  } /* end switch MTReference type */

} /* end print cross section or yield info */

/*
** *** LANDP IXS(9) ***
** *** ANDP IXS(10) ***
** print the angular distribution locators and information
**   currently the LANDP block indicates isotropy by a zero (0)
**     entry or information contained in the energy data
**     by a negative one (-1) entry
**   currently there is no separate angular information,
**     therefore the ANDP block never exists
*/
for( j = 0; j < prod->NumberOfReactions; j++ )
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->MTReference[j]->Emit->AngularInformationType),
              &count );

/*
** *** LDLWP IXS(11) ***
** *** LDLW IXS(12) ***
** print the energy distribution offsets and data values
*/

for( j = 0; j < prod->NumberOfReactions; j++ )
  acePrintXSS( fACE, 'a', 'i', 1,
              &(prod->MTReference[j]->Emit->Offset), &count );

for( j = 0; j < prod->NumberOfReactions; j++ ) {

  edata = prod->MTReference[j]->Emit;

  for( k = 0; k < edata->NumberOfEnergyLaws; k++ ) {

    lawinfo = &(edata->LawInformation[k]);

    acePrintXSS( fACE, 'a', 'i', 1,
                &(lawinfo->LocationOfNextLaw), &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(lawinfo->Number), &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(lawinfo->OffsetToLawData), &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(lawinfo->NumberOfRegions), &count );

    acePrintXSS( fACE, 'a', 'i', lawinfo->NumberOfRegions,
                lawinfo->NumberOfPointsInRegion, &count );

    acePrintXSS( fACE, 'a', 'i', lawinfo->NumberOfRegions,
                lawinfo->InterpolationSchemeInRegion, &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                &(lawinfo->NumberOfEnergies), &count );

    acePrintXSS( fACE, 'a', 'd', lawinfo->NumberOfEnergies,
                lawinfo->Energy, &count );

    acePrintXSS( fACE, 'a', 'd', lawinfo->NumberOfEnergies,
                lawinfo->Probability, &count );
```

```
switch( lawinfo->Number ) {
case 4:

  law4 = (aceLaw4*)(lawinfo->LawData);

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law4->NumberOfRegions), &count );

  acePrintXSS( fACE, 'a', 'i', law4->NumberOfRegions,
               law4->NumberOfPointsInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', law4->NumberOfRegions,
               law4->InterpolationSchemeInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law4->NumberOfIncidentEnergies), &count );

  for( l = 0; l < law4->NumberOfIncidentEnergies; l++ )
    acePrintXSS( fACE, 'a', 'd', 1,
                 &(law4->Distribution[l].IncidentEnergy),
                 &count );

  for( l = 0; l < law4->NumberOfIncidentEnergies; l++ )
    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law4->Distribution[l].Offset),
                 &count );

  for( l = 0; l < law4->NumberOfIncidentEnergies; l++ ) {

    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law4->Distribution[l].InterpolationScheme),
                 &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law4->Distribution[l].NumberOfPoints),
                 &count );

    acePrintXSS( fACE, 'a', 'd', law4->Distribution[l].NumberOfPoints,
                 law4->Distribution[l].EmissionEnergy, &count );

    acePrintXSS( fACE, 'a', 'd', law4->Distribution[l].NumberOfPoints,
                 law4->Distribution[l].Probability, &count );

    acePrintXSS( fACE, 'a', 'd', law4->Distribution[l].NumberOfPoints,
                 law4->Distribution[l].CumulativeProbability, &count );

  }

  break;

case 7:

  law7 = (aceLaw7*)(lawinfo->LawData);

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law7->NumberOfRegions), &count );

  acePrintXSS( fACE, 'a', 'i', law7->NumberOfRegions,
               law7->NumberOfPointsInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', law7->NumberOfRegions,
               law7->InterpolationSchemeInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law7->NumberOfIncidentEnergies), &count );

  acePrintXSS( fACE, 'a', 'd', law7->NumberOfIncidentEnergies,
               law7->IncidentEnergy, &count );

  acePrintXSS( fACE, 'a', 'd', law7->NumberOfIncidentEnergies,
               law7->Temperature, &count );
  acePrintXSS( fACE, 'a', 'd', 1,
               &(law7->RestrictionEnergy), &count );

  break;

case 9:

  law9 = (aceLaw9*)(lawinfo->LawData);

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law9->NumberOfRegions), &count );

  acePrintXSS( fACE, 'a', 'i', law9->NumberOfRegions,
               law9->NumberOfPointsInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', law9->NumberOfRegions,
               law9->InterpolationSchemeInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law9->NumberOfIncidentEnergies), &count );

  acePrintXSS( fACE, 'a', 'd', law9->NumberOfIncidentEnergies,
               law9->IncidentEnergy, &count );

  acePrintXSS( fACE, 'a', 'd', law9->NumberOfIncidentEnergies,
               law9->Temperature, &count );

  acePrintXSS( fACE, 'a', 'd', 1,
               &(law9->RestrictionEnergy), &count );

  break;

case 44:

  law44 = (aceLaw44*)(lawinfo->LawData);

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law44->NumberOfRegions), &count );

  acePrintXSS( fACE, 'a', 'i', law44->NumberOfRegions,
               law44->NumberOfPointsInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', law44->NumberOfRegions,
               law44->InterpolationSchemeInRegion, &count );

  acePrintXSS( fACE, 'a', 'i', 1,
               &(law44->NumberOfIncidentEnergies), &count );

  for( l = 0; l < law44->NumberOfIncidentEnergies; l++ )
    acePrintXSS( fACE, 'a', 'd', 1,
                 &(law44->Distribution[l].IncidentEnergy),
                 &count );

  for( l = 0; l < law44->NumberOfIncidentEnergies; l++ )
    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law44->Distribution[l].Offset),
                 &count );

  for( l = 0; l < law44->NumberOfIncidentEnergies; l++ ) {

    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law44->Distribution[l].InterpolationScheme),
                 &count );

    acePrintXSS( fACE, 'a', 'i', 1,
                 &(law44->Distribution[l].NumberOfPoints),
                 &count );

    acePrintXSS( fACE, 'a', 'd', law44->Distribution[l].NumberOfPoints,
                 law44->Distribution[l].EmissionEnergy, &count );
```

```
            acePrintXSS( fACE, 'a', 'd', law44->Distribution[l].NumberOfPoints,
                         law44->Distribution[l].Probability, &count );

            acePrintXSS( fACE, 'a', 'd', law44->Distribution[l].NumberOfPoints,
                         law44->Distribution[l].CumulativeProbability, &count );

            acePrintXSS( fACE, 'a', 'd', law44->Distribution[l].NumberOfPoints,
                         law44->Distribution[l].PrecompoundFraction, &count );

            acePrintXSS( fACE, 'a', 'd', law44->Distribution[l].NumberOfPoints,
                         law44->Distribution[l].AngularDistributionSlope, &count );

          }

          break;

        default:
          printf( "ERROR: don't know energy law number %d\n", lawinfo->Number );
          break;
        }
      }
    }
  }

  fprintf( fACE, "\n" );
  fclose( fACE );

}


/*******************************************************************
** acePrintXSS
*/
void acePrintXSS( FILE *fACE, char ft, char dt,
                  int number, void *data, int *count )
{
  int      i;
  int     *idata;
  double  *ddata;


  if( number == 0 )
    return;

  switch( ft ) {

  case 'a':

    switch( dt ) {

    case 'i':
      idata = (int*)data;
      for( i = 0; i < number; i++ ) {
        fprintf( fACE, "%20d", idata[i] );
        if( *count % 4 == 0 )
          fprintf( fACE, "\n" );
        (*count)++;
      }
      break;

    case 'd':
      ddata = (double*)data;
      for( i = 0; i < number; i++ ) {
        fprintf( fACE, "%20.11E", ddata[i] );
        if( *count % 4 == 0 )
          fprintf( fACE, "\n" );
        (*count)++;
      }
      break;
    }

  }

}
```

afeCollectEnergies.c

```
#include "endf6.h"
#include "acepnData.h"


void afeCollectEnergies( endfMaterialInformation *mi,
                        int *count, double **energy );
void afeAddEnergyToGrid( double *add, int sizeadd,
                        double *grid, int *sizegrid, int *maxsize );


/*******************************************************************
** afeCollectEnergies
**
** Create a superset union of all relevant energy grids
**
** Collect all energy values from any MF3 section,
*/
void afeCollectEnergies( endfMaterialInformation *mi,
                        int *count, double **energy )
{
  int       i, j;
  int       maxsize;

  double    dvp[2];
  double   *e;
  double    mega = 1.0e6;

  endfMF3  *mf3data;
  endfMF6  *mf6data;


  (*count) = 0;
  for( i = 0; i < mi->NumberOfRecords; i++ ) {

    /*
    ** count cross section energy grids
    */
    if( mi->Records[i]->MF == 3 ) {

      mf3data = (endfMF3*)mi->Records[i]->MFMT;

      (*count) += mf3data->NumberOfPoints;

      /*
      ** add an extra point for possible double value point start
      */
      (*count) += 2;
    }

    /*
    ** count yield points and add to possible grid points
    */
    if( mi->Records[i]->MF == 6 ) {

      mf6data = (endfMF6*)mi->Records[i]->MFMT;

      for( j = 0; j < mf6data->NumberOfSubsections; j++ )
        (*count) += mf6data->Secondaries[j].NumberOfYieldPoints;

    }

  } /* end loop for max size of energy grid */
  /*
  ** allocate array at max possible size
  */
  maxsize = (*count);
  e = (double*)calloc( maxsize, sizeof( double ) );

  /*
  ** now count actual entries added to the union grid
  */
  (*count) = 0;
  for( i = 0; i < mi->NumberOfRecords; i++ ) {

    /*
    ** collect cross section energy grids
    */
    if( mi->Records[i]->MF == 3 ) {

      mf3data = (endfMF3*)mi->Records[i]->MFMT;

      afeAddEnergyToGrid( mf3data->Energy, mf3data->NumberOfPoints,
                          e, count, &maxsize );


      if( mf3data->CrossSection[0] != 0 ) {
        dvp[0] = dvp[1] = mf3data->Energy[0];
        afeAddEnergyToGrid( dvp, 2, e, count, &maxsize );
      }

      if( mf3data->Energy[mf3data->NumberOfPoints-1] < e[(*count)-1] ) {
        dvp[0] = dvp[1] = mf3data->Energy[mf3data->NumberOfPoints-1];
        afeAddEnergyToGrid( dvp, 2, e, count, &maxsize );
      }

    }

    /*
    ** collect mf6 yield energy grids
    */
    if( mi->Records[i]->MF == 6 ) {

      mf6data = (endfMF6*)mi->Records[i]->MFMT;

      for( j = 0; j < mf6data->NumberOfSubsections; j++ )
        afeAddEnergyToGrid( mf6data->Secondaries[j].YieldEnergy,
                            mf6data->Secondaries[j].NumberOfYieldPoints,
                            e, count, &maxsize );

    }

  } /* end for i records */

  e = realloc( e, (*count) * sizeof( double ) );

  for( i = 0; i < *count; i++ )
    e[i] = e[i] / mega;

  *energy = e;

}


/*********************************************************************
** afeAddEnergyToGrid
**
** mesh two energy grids together
**
** expects the grid to be in numerical order (least to greatest)
** if two consequtive points are coincident, add both
**
```

```
*/
void afeAddEnergyToGrid( double *add, int sizeadd,
                       double *grid, int *sizegrid, int *maxsize )
{
  int i;
  int threshold;
  int curpos;


  /*
  ** if no entries in union grid, add all remaining and exit function
  */
  if( *sizegrid == 0 ) {
    for( i = 0; i < sizeadd; i++ ) {
      if( *sizegrid > *maxsize) {
        *maxsize = *sizegrid;
        grid = (double*)realloc( grid, *maxsize );
      }
      grid[i] = add[i];
      (*sizegrid)++;
    }
    return;
  }

  /*
  ** else if previous entries exist, mesh new entries in order
  */
  for( i = 0, curpos = 0; i < sizeadd; i++ ) {

    /*
    ** skip matching entries
    */
    if( add[i] == grid[curpos] ) {

      /*
      ** unless add contains two equal consecutive entries,
      ** then add both  to grid unless they are already there
      */
      if( add[i] == add[i-1]  &&  i > 0
          && grid[curpos] != grid[curpos+1]
          &&  curpos < *sizegrid ) {
        if( (*sizegrid)+1 > (*maxsize) )
          grid = (double*)realloc( grid, ++(*maxsize) );
        memmove( &grid[curpos+1], &grid[curpos],
                 (*sizegrid - curpos) * sizeof( double ) );
        grid[curpos] = add[i];
        (*sizegrid)++;
      }

      /*
      ** else drop duplicate and move to next entry
      */
    }

    /*
    ** add new energy, shuffling union grid
    */
    else if( add[i] < grid[curpos] ) {
      if( (*sizegrid)+1 > (*maxsize) )
        grid = (double*)realloc( grid, ++(*maxsize) );
      memmove( &grid[curpos+1], &grid[curpos],
               (*sizegrid - curpos) * sizeof( double ) );
      grid[curpos] = add[i];
      (*sizegrid)++;
    }

    /*
    ** move to next grid position
    */
    else if( add[i] > grid[curpos] ) {
      curpos++;
      /*
      ** if at end of union grid, add all remaining and exit loop
      */
      if( curpos == *sizegrid ) {
        for( ; i < sizeadd; i++ ) {
          if( (*sizegrid)+1 > (*maxsize) )
            grid = (double*)realloc( grid, ++(*maxsize) );
          grid[curpos] = add[i];
          curpos++;
          (*sizegrid)++;
        }
      }
      /*
      ** else reset i to check against next grid entry
      */
      else
        i--;

    } /* end else if */

  } /* end for i add entries */

}
```

afeCreateNTableHeader.c

```
#include "endf6.h"
#include "acepnData.h"

#include <time.h>


void afeCreateNTableHeader( endfMaterialInformation *mi, aceTable *table );


/*******************************************************************
** afeCreateNTableHeader
*/
void afeCreateNTableHeader( endfMaterialInformation *mi, aceTable *table )
{
  double      eps = 0.000001;

  time_t      now;


  table->ZA = (int)(mi->TargetZA + eps);

  table->Z = table->ZA / 1000;
  table->A = table->ZA % 1000;

  table->LibraryNumber = 0;
  table->TableType = 'n';

  sprintf( table->TableIdentifier, "%6d.%02d%c",
           table->ZA, table->LibraryNumber, table->TableType );

  table->IncidentParticleName = (char*)calloc( 7, sizeof( char ) );
  strcpy( table->IncidentParticleName, "photon" );

  table->AtomicWeightRatio = mi->TargetAWR;

  time( &now );
  strftime( table->ProcessDate, 11, "  %m/%d/%y", localtime(&now) );

  table->MaterialNumber = mi->MaterialNumber;
  sprintf( table->MaterialIdentifier, "mat%d", table->MaterialNumber );

  sprintf( table->Comment, "%d-%s-%d from %s distributed on %s",
           table->Z, mi->TargetSymbol, table->A,
          mi->EvaluationLaboratory, mi->EvaluationDistributionDate );


}
```

afeGetMTInformation.c

```
#include "endf6.h"
#include "acepnData.h"

void afeGetMTInformation( endfMaterialInformation *mi, acepnData *data );
void afeFillCrossSection( endfMF3 *mf3data, aceMTInformation *mt );
int aceFindMT( int number, int numberofmts, aceMTInformation **mt );

void afeGetMTNames( aceMTInformation *mt );


/*******************************************************************
** afeGetMTInformation
**
*/
void afeGetMTInformation( endfMaterialInformation *mi, acepnData *data )
{
  int   i, j;
  int   mtn;

  double  *xs;

  endfMF3  *mf3data;

  aceMTInformation  *mt;


  /*
  ** determine the number of mf3 records and allocate an MTInformation
  **   entry for each mf3
  */
  data->TotalCrossSection == NULL;
  data->ElasticCrossSection == NULL;
  data->NonelasticCrossSection == NULL;
  for( i = 0; i < mi->NumberOfRecords; i++ )
    if( mi->Records[i]->MF == 3
        && mi->Records[i]->MT != 1
        && mi->Records[i]->MT != 2
        && mi->Records[i]->MT != 3 )
      data->NumberOfMTs++;

  data->MT = (aceMTInformation**)calloc( data->NumberOfMTs,
                                       sizeof( aceMTInformation* ) );
  for( i = 0; i < data->NumberOfMTs; i++ )
    data->MT[i] = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );


  /*
  ** fill in the MTInfomation
  */
  for( i = 0, j = 0; i < mi->NumberOfRecords; i++ ) {

    if( mi->Records[i]->MF == 3 ) {

      mf3data = (endfMF3*)mi->Records[i]->MFMT;

      if( mi->Records[i]->MT == 1 ) {
        data->TotalCrossSection =
          (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
        mt = data->TotalCrossSection;
      }
      else if( mi->Records[i]->MT == 2 ) {
        data->ElasticCrossSection =
          (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
        mt = data->ElasticCrossSection;
      }
      else if( mi->Records[i]->MT == 3 ) {
        data->NonelasticCrossSection =
          (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
        mt = data->NonelasticCrossSection;
      }
      else {
        mt = data->MT[j];
        j++;
      }

      mt->Number = mi->Records[i]->MT;

      afeGetMTNames( mt );

      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );

      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      afeFillCrossSection( mf3data, mt );

      mt->Q = mf3data->ReactionQM;
    }

  }

  /*
  ** check the single neutron exit channel mts
  **   summation in mt 4
  **   xs in mt 50 - 91
  */
  if( (mtn = aceFindMT( 4, data->NumberOfMTs, data->MT )) >= 0 ) {
    if( aceFindMT( 91, data->NumberOfMTs, data->MT ) < 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 91;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < *mt->NumberOfEnergies; i++ )
        mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
      mt->StartingIndex = data->MT[mtn]->StartingIndex;
      mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
      mt->Q = data->MT[mtn]->Q;
      data->MT[mtn]->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
    else {
      xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 50 && data->MT[i]->Number <= 91 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            xs[j] += data->MT[i]->CrossSection[j];
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( xs[i] != data->MT[mtn]->CrossSection[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( xs );
      else {
        printf( "WARNING: MT4 Summation set to sum of partials\n" );
        free( data->MT[mtn]->CrossSection );
        data->MT[mtn]->CrossSection = xs;
      }
    }
  }
  else {
    if( aceFindMT( 91, data->NumberOfMTs, data->MT ) >= 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 4;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 50 && data->MT[i]->Number <= 91 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            mt->CrossSection[j] += data->MT[i]->CrossSection[j];
      mt->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
  }

  /*
  ** check the single proton exit channel mts
  **   summation in mt 103
  **   xs in mt 600-649
  */
  if( (mtn = aceFindMT( 103, data->NumberOfMTs, data->MT )) >= 0 ) {
    if( aceFindMT( 649, data->NumberOfMTs, data->MT ) < 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 649;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < *mt->NumberOfEnergies; i++ )
        mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
      mt->StartingIndex = data->MT[mtn]->StartingIndex;
      mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
      mt->Q = data->MT[mtn]->Q;
      data->MT[mtn]->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
    else {
      xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 600 && data->MT[i]->Number <= 649 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            xs[j] += data->MT[i]->CrossSection[j];
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( xs[i] != data->MT[mtn]->CrossSection[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( xs );
      else {
        printf( "WARNING: MT103 Summation set to sum of partials\n" );
        free( data->MT[mtn]->CrossSection );
        data->MT[mtn]->CrossSection = xs;
      }
    }
  }
  else {
    if( aceFindMT( 649, data->NumberOfMTs, data->MT ) >= 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 103;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 600 && data->MT[i]->Number <= 649 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            mt->CrossSection[j] += data->MT[i]->CrossSection[j];
      mt->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
  }

  /*
  ** check the single deuteron exit channel mts
  **   summation in mt 104
  **   xs in mt 650 - 699
  */
  if( (mtn = aceFindMT( 104, data->NumberOfMTs, data->MT )) >= 0 ) {
    if( aceFindMT( 699, data->NumberOfMTs, data->MT ) < 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 699;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < *mt->NumberOfEnergies; i++ )
        mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
      mt->StartingIndex = data->MT[mtn]->StartingIndex;
      mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
      mt->Q = data->MT[mtn]->Q;
      data->MT[mtn]->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                        data->NumberOfMTs *
                                        sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
    else {
      xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 650 && data->MT[i]->Number <= 699 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            xs[j] += data->MT[i]->CrossSection[j];
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( xs[i] != data->MT[mtn]->CrossSection[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( xs );
      else {
        printf( "WARNING: MT104 Summation set to sum of partials\n" );
        free( data->MT[mtn]->CrossSection );
        data->MT[mtn]->CrossSection = xs;
      }
    }
  }
  else {
    if( aceFindMT( 699, data->NumberOfMTs, data->MT ) >= 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 104;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                    sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 650 && data->MT[i]->Number <= 699 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            mt->CrossSection[j] += data->MT[i]->CrossSection[j];
      mt->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                        data->NumberOfMTs *
                                        sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
  }

  /*
  ** check the single triton exit channel mts
  **   summation in mt 105
  **   xs in mt 700 - 749
  */
  if( (mtn = aceFindMT( 105, data->NumberOfMTs, data->MT )) >= 0 ) {
    if( aceFindMT( 749, data->NumberOfMTs, data->MT ) < 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 749;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < *mt->NumberOfEnergies; i++ )
        mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
      mt->StartingIndex = data->MT[mtn]->StartingIndex;
      mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
      mt->Q = data->MT[mtn]->Q;
      data->MT[mtn]->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
    else {
      xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 700 && data->MT[i]->Number <= 749 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            xs[j] += data->MT[i]->CrossSection[j];
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( xs[i] != data->MT[mtn]->CrossSection[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( xs );
      else {
        printf( "WARNING: MT105 Summation set to sum of partials\n" );
        free( data->MT[mtn]->CrossSection );
        data->MT[mtn]->CrossSection = xs;
      }
    }
  }
  else {
    if( aceFindMT( 749, data->NumberOfMTs, data->MT ) >= 0 ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 105;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( data->MT[i]->Number >= 700 && data->MT[i]->Number <= 749 )
          for( j = 0; j < data->NumberOfEnergies; j++ )
            mt->CrossSection[j] += data->MT[i]->CrossSection[j];
      mt->Q = 0;
      for( i = 0; i < data->NumberOfMTs; i++ )
        if( mt->Number < data->MT[i]->Number )
          break;
      data->NumberOfMTs++;
      data->MT = (aceMTInformation**)realloc( data->MT,
                                              data->NumberOfMTs *
                                              sizeof( aceMTInformation* ) );
      if( i == data->NumberOfMTs - 1 )
        data->MT[i] = mt;
      else {
        memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
        data->MT[i] = mt;
      }
    }
  }
```

```
/*
** check the single helium-3 exit channel mts
**   summation in mt 106
**   xs in mt 750 - 799
*/
if( (mtn = aceFindMT( 106, data->NumberOfMTs, data->MT )) >= 0 ) {
  if( aceFindMT( 799, data->NumberOfMTs, data->MT ) < 0 ) {
    mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
    mt->Number = 799;
    afeGetMTNames( mt );
    mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                     sizeof( double ) );
    mt->NumberOfEnergies = &(data->NumberOfEnergies);
    mt->Energy = data->Energy;
    for( i = 0; i < *mt->NumberOfEnergies; i++ )
      mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
    mt->StartingIndex = data->MT[mtn]->StartingIndex;
    mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
    mt->Q = data->MT[mtn]->Q;
    data->MT[mtn]->Q = 0;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( mt->Number < data->MT[i]->Number )
        break;
    data->NumberOfMTs++;
    data->MT = (aceMTInformation**)realloc( data->MT,
                                         data->NumberOfMTs *
                                         sizeof( aceMTInformation* ) );
    if( i == data->NumberOfMTs - 1 )
      data->MT[i] = mt;
    else {
      memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
      data->MT[i] = mt;
    }
  }
  else {
    xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( data->MT[i]->Number >= 750 && data->MT[i]->Number <= 799 )
        for( j = 0; j < data->NumberOfEnergies; j++ )
          xs[j] += data->MT[i]->CrossSection[j];
    for( i = 0; i < data->NumberOfEnergies; i++ )
      if( xs[i] != data->MT[mtn]->CrossSection[i] )
        break;
    if( i == data->NumberOfEnergies )
      free( xs );
    else {
      printf( "WARNING: MT106 Summation set to sum of partials\n" );
      free( data->MT[mtn]->CrossSection );
      data->MT[mtn]->CrossSection = xs;
    }
  }
}
else {
  if( aceFindMT( 799, data->NumberOfMTs, data->MT ) >= 0 ) {
    mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
    mt->Number = 106;
    afeGetMTNames( mt );
    mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                     sizeof( double ) );
    mt->NumberOfEnergies = &(data->NumberOfEnergies);
    mt->Energy = data->Energy;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( data->MT[i]->Number >= 750 && data->MT[i]->Number <= 799 )
        for( j = 0; j < data->NumberOfEnergies; j++ )
          mt->CrossSection[j] += data->MT[i]->CrossSection[j];
    mt->Q = 0;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( mt->Number < data->MT[i]->Number )
        break;
    data->NumberOfMTs++;
    data->MT = (aceMTInformation**)realloc( data->MT,
                                     data->NumberOfMTs *
                                     sizeof( aceMTInformation* ) );
    if( i == data->NumberOfMTs - 1 )
      data->MT[i] = mt;
    else {
      memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
      data->MT[i] = mt;
    }
  }
}

/*
** check the single alpha exit channel mts
**   summation in mt 107
**   xs in mt 800 - 849
*/
if( (mtn = aceFindMT( 107, data->NumberOfMTs, data->MT )) >= 0 ) {
  if( aceFindMT( 849, data->NumberOfMTs, data->MT ) < 0 ) {
    mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
    mt->Number = 849;
    afeGetMTNames( mt );
    mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                 sizeof( double ) );
    mt->NumberOfEnergies = &(data->NumberOfEnergies);
    mt->Energy = data->Energy;
    for( i = 0; i < *mt->NumberOfEnergies; i++ )
      mt->CrossSection[i] = data->MT[mtn]->CrossSection[i];
    mt->StartingIndex = data->MT[mtn]->StartingIndex;
    mt->NumberOfEntries = data->MT[mtn]->NumberOfEntries;
    mt->Q = data->MT[mtn]->Q;
    data->MT[mtn]->Q = 0;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( mt->Number < data->MT[i]->Number )
        break;
    data->NumberOfMTs++;
    data->MT = (aceMTInformation**)realloc( data->MT,
                                     data->NumberOfMTs *
                                     sizeof( aceMTInformation* ) );
    if( i == data->NumberOfMTs - 1 )
      data->MT[i] = mt;
    else {
      memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
      data->MT[i] = mt;
    }
  }
  else {
    xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( data->MT[i]->Number >= 800 && data->MT[i]->Number <= 849 )
        for( j = 0; j < data->NumberOfEnergies; j++ )
          xs[j] += data->MT[i]->CrossSection[j];
    for( i = 0; i < data->NumberOfEnergies; i++ )
      if( xs[i] != data->MT[mtn]->CrossSection[i] )
        break;
    if( i == data->NumberOfEnergies )
      free( xs );
    else {
      printf( "WARNING: MT107 Summation set to sum of partials\n" );
      free( data->MT[mtn]->CrossSection );
      data->MT[mtn]->CrossSection = xs;
    }
  }
}
else {
  if( aceFindMT( 849, data->NumberOfMTs, data->MT ) >= 0 ) {
    mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
    mt->Number = 107;
    afeGetMTNames( mt );
    mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                 sizeof( double ) );
    mt->NumberOfEnergies = &(data->NumberOfEnergies);
    mt->Energy = data->Energy;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( data->MT[i]->Number >= 800 && data->MT[i]->Number <= 849 )
        for( j = 0; j < data->NumberOfEnergies; j++ )
          mt->CrossSection[j] += data->MT[i]->CrossSection[j];
    mt->Q = 0;
    for( i = 0; i < data->NumberOfMTs; i++ )
      if( mt->Number < data->MT[i]->Number )
        break;
    data->NumberOfMTs++;
    data->MT = (aceMTInformation**)realloc( data->MT,
                                      data->NumberOfMTs *
                                      sizeof( aceMTInformation* ) );
    if( i == data->NumberOfMTs - 1 )
      data->MT[i] = mt;
    else {
      memmove( &(data->MT[i+1]), &(data->MT[i]), (data->NumberOfMTs - i) );
      data->MT[i] = mt;
    }
  }
}

/*
** check total and nonelastic cross sections
*/
xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
for( i = 0; i < data->NumberOfMTs; i++ ) {
  switch( data->MT[i]->Number ) {
    /* skip all summation cross sections and the elastic */
  case 1: case 2: case 3: /*these should never be here anyway */
  case 4: case 103: case 104: case 105: case 106: case 107:
  case 201: case 202: case 203: case 204: case 205: case 206: case 207:
    break;
  default:
    if( data->MT[i]->Number > 1000 )
      break;
    for( j = 0; j < data->NumberOfEnergies; j++ )
      xs[j] += data->MT[i]->CrossSection[j];
  }
}
if( data->NonelasticCrossSection == NULL ) {
  mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
  mt->Number = 2;
  afeGetMTNames( mt );
  mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                    sizeof( double ) );
  mt->NumberOfEnergies = &(data->NumberOfEnergies);
  mt->Energy = data->Energy;
  mt->CrossSection = xs;
  mt->StartingIndex = 1;
  mt->NumberOfEntries = data->NumberOfEnergies;
  mt->Q = 0;
  data->NonelasticCrossSection = mt;
}
else {
  for( i = 0; i < data->NumberOfEnergies; i++ )
    if( data->NonelasticCrossSection->CrossSection[i] != xs[i] )
      break;
  if( i == data->NumberOfEnergies )
    free( xs );
  else {
    printf( "WARNING: MT3 Summation set to sum of partials\n" );
    free( data->NonelasticCrossSection->CrossSection );
    data->NonelasticCrossSection->CrossSection = xs;
  }
}

if( data->ElasticCrossSection != NULL ) {
  if( data->TotalCrossSection == NULL ) {
    mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
```

```
      mt->Number = 1;
      afeGetMTNames( mt );
      mt->CrossSection = (double*)calloc( data->NumberOfEnergies,
                                          sizeof( double ) );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      for( i = 0; i < data->NumberOfEnergies; i++ )
        mt->CrossSection[i] = data->ElasticCrossSection->CrossSection[i]
          + data->NonelasticCrossSection->CrossSection[i];
      mt->StartingIndex = 1;
      mt->NumberOfEntries = data->NumberOfEnergies;
      mt->Q = 0;
      data->TotalCrossSection = mt;
    }
    else {
      xs = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );
      for( i = 0; i < data->NumberOfEnergies; i++ )
        xs[i] = data->NonelasticCrossSection->CrossSection[i]
               + data->ElasticCrossSection->CrossSection[i];
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( data->TotalCrossSection->CrossSection[i] != xs[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( xs );
      else {
        printf( "WARNING: MT1 Summation set to sum of partials\n" );
        free( data->TotalCrossSection->CrossSection );
        data->TotalCrossSection->CrossSection = xs;
      }
    }
  }
  else {
    if( data->TotalCrossSection == NULL ) {
      mt = (aceMTInformation*)calloc( 1, sizeof( aceMTInformation ) );
      mt->Number = 1;
      afeGetMTNames( mt );
      mt->NumberOfEnergies = &(data->NumberOfEnergies);
      mt->Energy = data->Energy;
      mt->CrossSection = data->NonelasticCrossSection->CrossSection;
      mt->StartingIndex = 1;
      mt->NumberOfEntries = data->NumberOfEnergies;
      mt->Q = 0;
      data->TotalCrossSection = mt;
      free( data->NonelasticCrossSection );
      data->NonelasticCrossSection = NULL;
    }
    else {
      for( i = 0; i < data->NumberOfEnergies; i++ )
        if( data->TotalCrossSection->CrossSection[i]
            != data->NonelasticCrossSection->CrossSection[i] )
          break;
      if( i == data->NumberOfEnergies )
        free( data->NonelasticCrossSection->CrossSection );
      else {
        printf( "WARNING: MT1 Summation set to sum of partials\n" );
        free( data->TotalCrossSection->CrossSection );
        data->TotalCrossSection->CrossSection
          = data->NonelasticCrossSection->CrossSection;
      }
      free( data->NonelasticCrossSection );
      data->NonelasticCrossSection = NULL;
    }
  }

}


/*******************************************************************
** aceFindMT
**
** Find the MT with number n in list of pointers to MTs
**   return the positive number (including zero) if found
**   return a negative number if not found
*/
int aceFindMT( int number, int numberofmts, aceMTInformation **mt )
{
  int  i;


  for( i = 0; i < numberofmts; i++ )
    if( mt[i]->Number == number )
      break;

  if( i < numberofmts )
    return i;
  else
    return -1;

}


/*******************************************************************
** afeFillCrossSection
**
** Fill the cross section values from existing ENDF data
*/
void afeFillCrossSection( endfMF3 *mf3data, aceMTInformation *mt )
{
  int   i, j;
  int   start, end;

  double  *xs = mt->CrossSection;
  double   mega = 1.0e6;

  /*
  ** initialize starting index
  */
  mt->StartingIndex = 1;

  /*
  ** find first point on grid
  */
  for( start = 0;
       mt->Energy[start] * mega < mf3data->Energy[0];
       start++ )
    mt->StartingIndex++;

  /*
  ** find last point on grid
  */
  for( end = *mt->NumberOfEnergies - 1;
       mt->Energy[end] * mega > mf3data->Energy[mf3data->NumberOfPoints - 1];
       end-- )
    ;
  mt->NumberOfEntries = end - start + 1;

  /*
  ** fill all point in between by interpolation
  */
  for( i = start, j = 0; i <= end; i++ ) {

    if( i == start
        && mt->Energy[i] == mt->Energy[i+1]
        && mf3data->Energy[j] != mf3data->Energy[j+1] ) {
      xs[i] = 0.0;
      i++;
    }

    if( mt->Energy[i] * mega == mf3data->Energy[j] )
      xs[i] = mf3data->CrossSection[j];

    else if( mt->Energy[i] * mega < mf3data->Energy[j] ) {
      /* interp value assuming all endf data has been linearlized*/

      xs[i] = ( mt->Energy[i] * mega - mf3data->Energy[j-1] ) *
              ( mf3data->CrossSection[j] - mf3data->CrossSection[j-1] ) /
              ( mf3data->Energy[j] - mf3data->Energy[j-1] )
              + mf3data->CrossSection[j-1];
    }

    else {   /* if( mt->Energy[i] * mega > mf3data->Energy[j] ) */
      j++;
      i--;
    }

  }


  /*
  ** remove interpolated zero values from start of xs
  */
  for( i = start; i < end; i++ ) {
    if( xs[i] == 0.0 && xs[i+1] == 0.0 ) {
      (mt->StartingIndex)++;
      (mt->NumberOfEntries)--;
    }
    else
      i = end;
  }

}
```

afeGetMTNames.c

```
#include "acepnData.h"

void afeGetMTNames( aceMTInformation *mt );


/*********************************************************************
** afeGetMTNames
**
*/
void afeGetMTNames( aceMTInformation *mt )
{
  /*
  ** Name is specified from ENDF 102 format manual appendix B
  ** along with reaction and products.
  **
  ** Products are ZA, e.g. proton is 1001, neutron 1, gamma 0,
  **   etc...
  **
  ** Number of products uses -1 for explicit file 6 information
  **   -2 indicates fission cross section, yield from nu data
  **   -9 indicates summation cross section
  */

  switch( mt->Number ) {

    /*
    ** Summation cross sections
    */
  case 1:
    strcpy( mt->Name, "total" );
    strcpy( mt->Reaction, "(sum of elastic & non-elastic)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 3:
    strcpy( mt->Name, "non-elastic" );
    strcpy( mt->Reaction, "(sum of all non-elastic)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

    /*
    ** Reaction cross sections
    */
  case 2:
    strcpy( mt->Name, "elastic" );
    strcpy( mt->Reaction, "(gamma, gamma)" );
    mt->NumberOfProducts = 1;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 0;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    break;

  case 5:
    strcpy( mt->Name, "catch all" );
    strcpy( mt->Reaction, "(gamma, any)" );
    mt->NumberOfProducts = -1;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 11:
    strcpy( mt->Name, "2 neutron + deuterium" );
    strcpy( mt->Reaction, "(gamma, 2n d)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 1002;
    mt->ProductZA[2] = -1004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 2;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 16:
    strcpy( mt->Name, "2 neutron" );
    strcpy( mt->Reaction, "(gamma, 2n)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -2;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 2;
    mt->YieldOfProduct[1] = 1;
    break;

  case 17:
    strcpy( mt->Name, "3 neutron" );
    strcpy( mt->Reaction, "(gamma, 3n)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -3;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 3;
    mt->YieldOfProduct[1] = 1;
    break;

  case 18:
    strcpy( mt->Name, "fission" );
    strcpy( mt->Reaction, "(gamma, fission)" );
    mt->NumberOfProducts = -1;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 22:
    strcpy( mt->Name, "neutron + alpha" );
    strcpy( mt->Reaction, "(gamma, n alpha)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2004;
    mt->ProductZA[2] = -2005;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 24:
    strcpy( mt->Name, "2 neutron + alpha" );
    strcpy( mt->Reaction, "(gamma, 2n alpha)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2004;
    mt->ProductZA[2] = -2006;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 2;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 25:
    strcpy( mt->Name, "3 neutron + alpha" );
    strcpy( mt->Reaction, "(gamma, 3n alpha)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2004;
    mt->ProductZA[2] = -2007;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 3;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 28:
    strcpy( mt->Name, "neutron + proton" );
    strcpy( mt->Reaction, "(gamma, n p)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 1001;
    mt->ProductZA[2] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 29:
    strcpy( mt->Name, "neutron + 2 alpha" );
    strcpy( mt->Reaction, "(gamma, n 2alpha)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2004;
    mt->ProductZA[2] = -4009;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 2;
    mt->YieldOfProduct[2] = 1;
    break;

  case 30:
    strcpy( mt->Name, "2 neutron + 2 alpha" );
    strcpy( mt->Reaction, "(gamma, 2n 2alpha)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2004;
    mt->ProductZA[2] = -4010;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 2;
    mt->YieldOfProduct[1] = 2;
    mt->YieldOfProduct[2] = 1;
    break;

  case 32:
    strcpy( mt->Name, "neutron + deuterium" );
    strcpy( mt->Reaction, "(gamma, n d)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 1002;
    mt->ProductZA[2] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 33:
    strcpy( mt->Name, "neutron + triton" );
    strcpy( mt->Reaction, "(gamma, n t)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 1003;
    mt->ProductZA[2] = -1004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 34:
    strcpy( mt->Name, "neutron + helium-3" );
    strcpy( mt->Reaction, "(gamma, n he-3)" );
    mt->NumberOfProducts = 3;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 2003;
    mt->ProductZA[2] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 1;
    break;

  case 35:
    strcpy( mt->Name, "neutron + deuteron + 2 alpha" );
    strcpy( mt->Reaction, "(gamma, n d 2alpha)" );
    mt->NumberOfProducts = 4;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = 1002;
    mt->ProductZA[2] = 2004;
    mt->ProductZA[3] = -5011;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    mt->YieldOfProduct[2] = 2;
```

```
      mt->YieldOfProduct[3] = 1;
      break;

    case 36:
      strcpy( mt->Name, "neutron + triton + 2 alpha" );
      strcpy( mt->Reaction, "(gamma, n t 2alpha)" );
      mt->NumberOfProducts = 4;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = 1003;
      mt->ProductZA[2] = 2004;
      mt->ProductZA[3] = -5012;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 2;
      mt->YieldOfProduct[3] = 1;
      break;

    case 37:
      strcpy( mt->Name, "4 neutron" );
      strcpy( mt->Reaction, "(gamma, 4n)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -4;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 4;
      mt->YieldOfProduct[1] = 1;
      break;

    case 41:
      strcpy( mt->Name, "2 neutron + proton" );
      strcpy( mt->Reaction, "(gamma, 2n p)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = 1001;
      mt->ProductZA[2] = -1003;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 2;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;

    case 42:
      strcpy( mt->Name, "3 neutron + proton" );
      strcpy( mt->Reaction, "(gamma, 3n p)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = 1001;
      mt->ProductZA[2] = -1003;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 3;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;

    case 44:
      strcpy( mt->Name, "neutron + 2 proton" );
      strcpy( mt->Reaction, "(gamma, n 2p)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = 1001;
      mt->ProductZA[2] = -2003;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 2;
      mt->YieldOfProduct[2] = 1;
      break;

    case 45:
      strcpy( mt->Name, "neutron + proton + alpha" );
      strcpy( mt->Reaction, "(gamma, n p alpha)" );
      mt->NumberOfProducts = 4;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = 1001;
      mt->ProductZA[2] = 2004;
      mt->ProductZA[3] = -3006;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      mt->YieldOfProduct[3] = 1;
      break;

    case 102:
      strcpy( mt->Name, "radiative capture" );
      strcpy( mt->Reaction, "(gamma, Xgamma)" );
      mt->NumberOfProducts = -2;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 108:
      strcpy( mt->Name, "2 alpha" );
      strcpy( mt->Reaction, "(gamma, 2alpha)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 2004;
      mt->ProductZA[1] = -4008;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 2;
      mt->YieldOfProduct[1] = 1;
      break;

    case 109:
      strcpy( mt->Name, "3 alpha" );
      strcpy( mt->Reaction, "(gamma, 3alpha)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 2004;
      mt->ProductZA[1] = -6012;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 3;
      mt->YieldOfProduct[1] = 1;
      break;

    case 111:
      strcpy( mt->Name, "2 proton" );
      strcpy( mt->Reaction, "(gamma, 2proton)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1001;
      mt->ProductZA[1] = -2002;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 2;
      mt->YieldOfProduct[1] = 1;
      break;

    case 112:
      strcpy( mt->Name, "proton + alpha" );
      strcpy( mt->Reaction, "(gamma, p alpha)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1001;
      mt->ProductZA[1] = 2004;
      mt->ProductZA[2] = -3005;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;

    case 113:
      strcpy( mt->Name, "triton + 2 alpha" );
      strcpy( mt->Reaction, "(gamma, t 2alpha)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1003;
      mt->ProductZA[1] = 2004;
      mt->ProductZA[2] = -5011;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 2;
      mt->YieldOfProduct[2] = 1;
      break;

    case 114:
      strcpy( mt->Name, "deuteron + 2 alpha" );
      strcpy( mt->Reaction, "(gamma, d 2alpha)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1002;
      mt->ProductZA[1] = 2004;
      mt->ProductZA[2] = -5010;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 2;
      mt->YieldOfProduct[2] = 1;
      break;

    case 115:
      strcpy( mt->Name, "proton + deuteron" );
      strcpy( mt->Reaction, "(gamma, p d)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1001;
      mt->ProductZA[1] = 1002;
      mt->ProductZA[2] = -2003;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;

    case 116:
      strcpy( mt->Name, "proton + triton" );
      strcpy( mt->Reaction, "(gamma, p t)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1001;
      mt->ProductZA[1] = 1003;
      mt->ProductZA[2] = -2004;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;

    case 117:
      strcpy( mt->Name, "deuteron + alpha" );
      strcpy( mt->Reaction, "(gamma, d alpha)" );
      mt->NumberOfProducts = 3;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1002;
      mt->ProductZA[1] = 2004;
      mt->ProductZA[2] = -3006;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      mt->YieldOfProduct[2] = 1;
      break;


      /*
      ** Production cross sections
      */
    case 201:
      strcpy( mt->Name, "neutron production" );
      strcpy( mt->Reaction, "(gamma, Xn)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 202:
      strcpy( mt->Name, "gamma production" );
      strcpy( mt->Reaction, "(gamma, Xgamma)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 203:
      strcpy( mt->Name, "proton production" );
      strcpy( mt->Reaction, "(gamma, Xp)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 204:
      strcpy( mt->Name, "deuteron production" );
      strcpy( mt->Reaction, "(gamma, Xd)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 205:
      strcpy( mt->Name, "triton production" );
      strcpy( mt->Reaction, "(gamma, Xt)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 206:
      strcpy( mt->Name, "helium-3 production" );
      strcpy( mt->Reaction, "(gamma, XHe-3)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 207:
      strcpy( mt->Name, "alpha production" );
      strcpy( mt->Reaction, "(gamma, Xalpha)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

      /*
      ** one neutron exit channel with excited residual
      */
    case 4:
      strcpy( mt->Name, "single neutron channel sum" );
      strcpy( mt->Reaction, "(gamma, n)" );
      mt->NumberOfProducts = -9;
      mt->ProductZA = NULL;
      mt->YieldOfProduct = NULL;
      break;

    case 50:
      strcpy( mt->Name, "neutron + ground state residual" );
      strcpy( mt->Reaction, "(gamma, n0)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 51:
      strcpy( mt->Name, "neutron + 1st excited state residual" );
      strcpy( mt->Reaction, "(gamma, n1)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 52:
      strcpy( mt->Name, "neutron + 2nd excited state residual" );
      strcpy( mt->Reaction, "(gamma, n2)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 53:
      strcpy( mt->Name, "neutron + 3rd excited state residual" );
      strcpy( mt->Reaction, "(gamma, n3)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 54:
      strcpy( mt->Name, "neutron + 4th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n4)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 55:
      strcpy( mt->Name, "neutron + 5th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n5)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 56:
      strcpy( mt->Name, "neutron + 6th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n6)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 57:
      strcpy( mt->Name, "neutron + 7th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n7)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 58:
      strcpy( mt->Name, "neutron + 8th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n8)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 59:
      strcpy( mt->Name, "neutron + 9th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n9)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 60:
      strcpy( mt->Name, "neutron + 10th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n10)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
      mt->ProductZA[1] = -1;
      mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->YieldOfProduct[0] = 1;
      mt->YieldOfProduct[1] = 1;
      break;

    case 61:
      strcpy( mt->Name, "neutron + 11th excited state residual" );
      strcpy( mt->Reaction, "(gamma, n11)" );
      mt->NumberOfProducts = 2;
      mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
      mt->ProductZA[0] = 1;
```

```
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 62:
    strcpy( mt->Name, "neutron + 12th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n12)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 63:
    strcpy( mt->Name, "neutron + 13th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n13)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 64:
    strcpy( mt->Name, "neutron + 14th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n14)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 65:
    strcpy( mt->Name, "neutron + 15th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n15)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 66:
    strcpy( mt->Name, "neutron + 16th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n16)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 67:
    strcpy( mt->Name, "neutron + 17th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n17)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 68:
    strcpy( mt->Name, "neutron + 18th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n18)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 69:
    strcpy( mt->Name, "neutron + 19th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n19)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 70:
    strcpy( mt->Name, "neutron + 20th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n20)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 71:
    strcpy( mt->Name, "neutron + 21st excited state residual" );
    strcpy( mt->Reaction, "(gamma, n21)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 72:
    strcpy( mt->Name, "neutron + 22nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, n22)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 73:
    strcpy( mt->Name, "neutron + 23rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, n23)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 74:
    strcpy( mt->Name, "neutron + 24th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n24)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 75:
    strcpy( mt->Name, "neutron + 25th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n25)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 76:
    strcpy( mt->Name, "neutron + 26th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n26)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 77:
    strcpy( mt->Name, "neutron + 27th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n27)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 78:
    strcpy( mt->Name, "neutron + 28th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n28)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 79:
    strcpy( mt->Name, "neutron + 29th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n29)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 80:
    strcpy( mt->Name, "neutron + 30th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n30)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 81:
    strcpy( mt->Name, "neutron + 31st excited state residual" );
    strcpy( mt->Reaction, "(gamma, n31)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 82:
    strcpy( mt->Name, "neutron + 32nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, n32)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 83:
    strcpy( mt->Name, "neutron + 33rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, n33)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 84:
    strcpy( mt->Name, "neutron + 34th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n34)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 85:
    strcpy( mt->Name, "neutron + 35th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n35)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 86:
    strcpy( mt->Name, "neutron + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 87:
    strcpy( mt->Name, "neutron + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 88:
    strcpy( mt->Name, "neutron + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 89:
    strcpy( mt->Name, "neutron + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 90:
    strcpy( mt->Name, "neutron + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, n40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 91:
    strcpy( mt->Name, "neutron + continuum state residual" );
    strcpy( mt->Reaction, "(gamma, nz)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1;
    mt->ProductZA[1] = -1;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;


    /*
    ** one proton exit channel with excited residual
    */
  case 103:
    strcpy( mt->Name, "single proton channel sum" );
    strcpy( mt->Reaction, "(gamma, proton)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 600:
    strcpy( mt->Name, "proton + ground state residual" );
    strcpy( mt->Reaction, "(gamma, p0)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 601:
    strcpy( mt->Name, "proton + 1st excited state residual" );
    strcpy( mt->Reaction, "(gamma, p1)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 602:
    strcpy( mt->Name, "proton + 2nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p2)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 603:
    strcpy( mt->Name, "proton + 3rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p3)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;
```

```
case 604:
  strcpy( mt->Name, "proton + 4th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p4)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 605:
  strcpy( mt->Name, "proton + 5th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p5)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 606:
  strcpy( mt->Name, "proton + 6th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p6)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 607:
  strcpy( mt->Name, "proton + 7th excited state residual" );
  strcpy( mt->Reaction, "(gamma, pth)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 608:
  strcpy( mt->Name, "proton + 8th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p8)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 609:
  strcpy( mt->Name, "proton + 9th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p9)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;
```

```
case 610:
  strcpy( mt->Name, "proton + 10th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p10)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 611:
  strcpy( mt->Name, "proton + 11th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p11)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 612:
  strcpy( mt->Name, "proton + 12th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p12)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 613:
  strcpy( mt->Name, "proton + 13th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p13)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 614:
  strcpy( mt->Name, "proton + 14th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p14)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 615:
  strcpy( mt->Name, "proton + 15th excited state residual" );
  strcpy( mt->Reaction, "(gamma, p15)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1001;
  mt->ProductZA[1] = -1001;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
```

```
    break;

  case 616:
    strcpy( mt->Name, "proton + 16th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p16)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 617:
    strcpy( mt->Name, "proton + 17th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p17)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 618:
    strcpy( mt->Name, "proton + 18th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p18)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 619:
    strcpy( mt->Name, "proton + 19th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p19)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 620:
    strcpy( mt->Name, "proton + 20th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p20)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 621:
    strcpy( mt->Name, "proton + 21st excited state residual" );
    strcpy( mt->Reaction, "(gamma, p21)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 622:
    strcpy( mt->Name, "proton + 22nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p22)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 623:
    strcpy( mt->Name, "proton + 23rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p23)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 624:
    strcpy( mt->Name, "proton + 24th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p24)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 625:
    strcpy( mt->Name, "proton + 25th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p25)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 626:
    strcpy( mt->Name, "proton + 26th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p26)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 627:
    strcpy( mt->Name, "proton + 27th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p27)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 628:
    strcpy( mt->Name, "proton + 28th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p28)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 629:
    strcpy( mt->Name, "proton + 29th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p29)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 630:
    strcpy( mt->Name, "proton + 30th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p30)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 631:
    strcpy( mt->Name, "proton + 31st excited state residual" );
    strcpy( mt->Reaction, "(gamma, p31)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 632:
    strcpy( mt->Name, "proton + 32nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p32)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 633:
    strcpy( mt->Name, "proton + 33rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p33)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 634:
    strcpy( mt->Name, "proton + 34th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p34)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 635:
    strcpy( mt->Name, "proton + 35th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p35)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 636:
    strcpy( mt->Name, "proton + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 637:
    strcpy( mt->Name, "proton + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 638:
    strcpy( mt->Name, "proton + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 639:
    strcpy( mt->Name, "proton + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 640:
    strcpy( mt->Name, "proton + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 641:
    strcpy( mt->Name, "proton + 41st excited state residual" );
    strcpy( mt->Reaction, "(gamma, p41)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 642:
    strcpy( mt->Name, "proton + 42nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p42)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 643:
    strcpy( mt->Name, "proton + 43rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, p43)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 644:
    strcpy( mt->Name, "proton + 44th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p44)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 645:
    strcpy( mt->Name, "proton + 45th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p45)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 646:
    strcpy( mt->Name, "proton + 46th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p46)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 647:
    strcpy( mt->Name, "proton + 47th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p47)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 648:
    strcpy( mt->Name, "proton + 48th excited state residual" );
    strcpy( mt->Reaction, "(gamma, p48)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 649:
    strcpy( mt->Name, "proton + continuum state residual" );
    strcpy( mt->Reaction, "(gamma, pz)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1001;
    mt->ProductZA[1] = -1001;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;


    /*
    ** one deuteron exit channel with excited residual
    */
  case 104:
    strcpy( mt->Name, "single deuteron channel sum" );
    strcpy( mt->Reaction, "(gamma, deuteron)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 650:
    strcpy( mt->Name, "deuteron + ground state residual" );
    strcpy( mt->Reaction, "(gamma, d0)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 651:
    strcpy( mt->Name, "deuteron + 1st excited state residual" );
    strcpy( mt->Reaction, "(gamma, d1)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 652:
    strcpy( mt->Name, "deuteron + 2nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d2)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 653:
    strcpy( mt->Name, "deuteron + 3rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d3)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 654:
    strcpy( mt->Name, "deuteron + 4th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d4)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 655:
    strcpy( mt->Name, "deuteron + 5th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d5)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 656:
    strcpy( mt->Name, "deuteron + 6th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d6)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 657:
    strcpy( mt->Name, "deuteron + 7th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d7)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 658:
    strcpy( mt->Name, "deuteron + 8th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d8)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 659:
    strcpy( mt->Name, "deuteron + 9th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d9)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 660:
    strcpy( mt->Name, "deuteron + 10th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d10)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 661:
    strcpy( mt->Name, "deuteron + 11th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d11)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 662:
    strcpy( mt->Name, "deuteron + 12th excited state residual" );
```

```
    strcpy( mt->Reaction, "(gamma, d12)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 663:
    strcpy( mt->Name, "deuteron + 13th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d13)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 664:
    strcpy( mt->Name, "deuteron + 14th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d14)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 665:
    strcpy( mt->Name, "deuteron + 15th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d15)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 666:
    strcpy( mt->Name, "deuteron + 16th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d16)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 667:
    strcpy( mt->Name, "deuteron + 17th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d17)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 668:
    strcpy( mt->Name, "deuteron + 18th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d18)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 669:
    strcpy( mt->Name, "deuteron + 19th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d19)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 670:
    strcpy( mt->Name, "deuteron + 20th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d20)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 671:
    strcpy( mt->Name, "deuteron + 21st excited state residual" );
    strcpy( mt->Reaction, "(gamma, d21)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 672:
    strcpy( mt->Name, "deuteron + 22nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d22)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 673:
    strcpy( mt->Name, "deuteron + 23rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d23)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;
```

```
case 674:
  strcpy( mt->Name, "deuteron + 24th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d24)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 675:
  strcpy( mt->Name, "deuteron + 25th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d25)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 676:
  strcpy( mt->Name, "deuteron + 26th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d26)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 677:
  strcpy( mt->Name, "deuteron + 27th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d27)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 678:
  strcpy( mt->Name, "deuteron + 28th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d28)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 679:
  strcpy( mt->Name, "deuteron + 29th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d29)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;
```

```
case 680:
  strcpy( mt->Name, "deuteron + 30th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d30)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 681:
  strcpy( mt->Name, "deuteron + 31st excited state residual" );
  strcpy( mt->Reaction, "(gamma, d31)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 682:
  strcpy( mt->Name, "deuteron + 32nd excited state residual" );
  strcpy( mt->Reaction, "(gamma, d32)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 683:
  strcpy( mt->Name, "deuteron + 33rd excited state residual" );
  strcpy( mt->Reaction, "(gamma, d33)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 684:
  strcpy( mt->Name, "deuteron + 34th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d34)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 685:
  strcpy( mt->Name, "deuteron + 35th excited state residual" );
  strcpy( mt->Reaction, "(gamma, d35)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1002;
  mt->ProductZA[1] = -1002;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
```

```
    break;

  case 686:
    strcpy( mt->Name, "deuteron + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 687:
    strcpy( mt->Name, "deuteron + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 688:
    strcpy( mt->Name, "deuteron + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 689:
    strcpy( mt->Name, "deuteron + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 690:
    strcpy( mt->Name, "deuteron + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 691:
    strcpy( mt->Name, "deuteron + 41st excited state residual" );
    strcpy( mt->Reaction, "(gamma, d41)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 692:
    strcpy( mt->Name, "deuteron + 42nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d42)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 693:
    strcpy( mt->Name, "deuteron + 43rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, d43)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 694:
    strcpy( mt->Name, "deuteron + 44th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d44)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 695:
    strcpy( mt->Name, "deuteron + 45th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d45)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 696:
    strcpy( mt->Name, "deuteron + 46th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d46)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 697:
    strcpy( mt->Name, "deuteron + 47th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d47)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 698:
    strcpy( mt->Name, "deuteron + 48th excited state residual" );
    strcpy( mt->Reaction, "(gamma, d48)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 699:
    strcpy( mt->Name, "deuteron + continuum state residual" );
    strcpy( mt->Reaction, "(gamma, dz)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1002;
    mt->ProductZA[1] = -1002;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;


    /*
    ** one triton exit channel with excited residual
    */
  case 105:
    strcpy( mt->Name, "single triton channel sum" );
    strcpy( mt->Reaction, "(gamma, triton)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 700:
    strcpy( mt->Name, "triton + ground state residual" );
    strcpy( mt->Reaction, "(gamma, t0)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 701:
    strcpy( mt->Name, "triton + 1st excited state residual" );
    strcpy( mt->Reaction, "(gamma, t1)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 702:
    strcpy( mt->Name, "triton + 2nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t2)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 703:
    strcpy( mt->Name, "triton + 3rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t3)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 704:
    strcpy( mt->Name, "triton + 4th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t4)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 705:
    strcpy( mt->Name, "triton + 5th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t5)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 706:
    strcpy( mt->Name, "triton + 6th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t6)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 707:
    strcpy( mt->Name, "triton + 7th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t7)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 708:
    strcpy( mt->Name, "triton + 8th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t8)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 709:
    strcpy( mt->Name, "triton + 9th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t9)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 710:
    strcpy( mt->Name, "triton + 10th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t10)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 711:
    strcpy( mt->Name, "triton + 11th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t11)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 712:
    strcpy( mt->Name, "triton + 12th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t12)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 713:
    strcpy( mt->Name, "triton + 13th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t13)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 714:
    strcpy( mt->Name, "triton + 14th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t14)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 715:
    strcpy( mt->Name, "triton + 15th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t15)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 716:
    strcpy( mt->Name, "triton + 16th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t16)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 717:
    strcpy( mt->Name, "triton + 17th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t17)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 718:
    strcpy( mt->Name, "triton + 18th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t18)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 719:
    strcpy( mt->Name, "triton + 19th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t19)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 720:
    strcpy( mt->Name, "triton + 20th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t20)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 721:
    strcpy( mt->Name, "triton + 21st excited state residual" );
    strcpy( mt->Reaction, "(gamma, t21)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 722:
    strcpy( mt->Name, "triton + 22nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t22)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 723:
    strcpy( mt->Name, "triton + 23rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t23)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 724:
    strcpy( mt->Name, "triton + 24th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t24)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 725:
    strcpy( mt->Name, "triton + 25th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t25)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 726:
    strcpy( mt->Name, "triton + 26th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t26)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 727:
    strcpy( mt->Name, "triton + 27th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t27)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 728:
    strcpy( mt->Name, "triton + 28th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t28)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 729:
    strcpy( mt->Name, "triton + 29th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t29)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 730:
    strcpy( mt->Name, "triton + 30th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t30)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 731:
    strcpy( mt->Name, "triton + 31st excited state residual" );
    strcpy( mt->Reaction, "(gamma, t31)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 732:
    strcpy( mt->Name, "triton + 32nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t32)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 733:
    strcpy( mt->Name, "triton + 33rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t33)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 734:
    strcpy( mt->Name, "triton + 34th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t34)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 735:
    strcpy( mt->Name, "triton + 35th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t35)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 736:
    strcpy( mt->Name, "triton + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 737:
    strcpy( mt->Name, "triton + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 738:
    strcpy( mt->Name, "triton + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 739:
    strcpy( mt->Name, "triton + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 740:
    strcpy( mt->Name, "triton + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, t40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 741:
    strcpy( mt->Name, "triton + 41st excited state residual" );
    strcpy( mt->Reaction, "(gamma, t41)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 742:
    strcpy( mt->Name, "triton + 42nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t42)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 743:
    strcpy( mt->Name, "triton + 43rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, t43)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 1003;
    mt->ProductZA[1] = -1003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;
```

```
case 744:
  strcpy( mt->Name, "triton + 44th excited state residual" );
  strcpy( mt->Reaction, "(gamma, t44)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 745:
  strcpy( mt->Name, "triton + 45th excited state residual" );
  strcpy( mt->Reaction, "(gamma, t45)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 746:
  strcpy( mt->Name, "triton + 46th excited state residual" );
  strcpy( mt->Reaction, "(gamma, t46)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 747:
  strcpy( mt->Name, "triton + 47th excited state residual" );
  strcpy( mt->Reaction, "(gamma, t47)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 748:
  strcpy( mt->Name, "triton + 48th excited state residual" );
  strcpy( mt->Reaction, "(gamma, t48)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 749:
  strcpy( mt->Name, "triton + continuum state residual" );
  strcpy( mt->Reaction, "(gamma, tz)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 1003;
  mt->ProductZA[1] = -1003;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;
```

```
    /*
    ** one helium-3 exit channel with excited residual
    */
  case 106:
    strcpy( mt->Name, "single helium-3 channel sum" );
    strcpy( mt->Reaction, "(gamma, he3)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 750:
    strcpy( mt->Name, "helium-3 + ground state residual" );
    strcpy( mt->Reaction, "(gamma, he3-0)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 751:
    strcpy( mt->Name, "helium-3 + 1st excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-1)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 752:
    strcpy( mt->Name, "helium-3 + 2nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-2)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 753:
    strcpy( mt->Name, "helium-3 + 3th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-3)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 754:
    strcpy( mt->Name, "helium-3 + 4th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-4)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 755:
    strcpy( mt->Name, "helium-3 + 5th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-5)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 756:
    strcpy( mt->Name, "helium-3 + 6th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-6)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 757:
    strcpy( mt->Name, "helium-3 + 7th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-7)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 758:
    strcpy( mt->Name, "helium-3 + 8th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-8)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 759:
    strcpy( mt->Name, "helium-3 + 9th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-9)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 760:
    strcpy( mt->Name, "helium-3 + 10th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-10)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
```

```
    mt->YieldOfProduct[1] = 1;
    break;

  case 761:
    strcpy( mt->Name, "helium-3 + 11th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-11)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 762:
    strcpy( mt->Name, "helium-3 + 12th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-12)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 763:
    strcpy( mt->Name, "helium-3 + 13th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-13)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 764:
    strcpy( mt->Name, "helium-3 + 14th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-14)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 765:
    strcpy( mt->Name, "helium-3 + 15th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-15)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 766:
    strcpy( mt->Name, "helium-3 + 16th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-16)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 767:
    strcpy( mt->Name, "helium-3 + 17th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-17)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 768:
    strcpy( mt->Name, "helium-3 + 18th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-18)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 769:
    strcpy( mt->Name, "helium-3 + 19th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-19)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 770:
    strcpy( mt->Name, "helium-3 + 20th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-20)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 771:
    strcpy( mt->Name, "helium-3 + 21st excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-21)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 772:
    strcpy( mt->Name, "helium-3 + 22nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-22)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 773:
    strcpy( mt->Name, "helium-3 + 23rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-23)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 774:
    strcpy( mt->Name, "helium-3 + 24th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-24)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 775:
    strcpy( mt->Name, "helium-3 + 25th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-25)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 776:
    strcpy( mt->Name, "helium-3 + 26th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-26)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 777:
    strcpy( mt->Name, "helium-3 + 27th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-27)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 778:
    strcpy( mt->Name, "helium-3 + 28th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-28)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
```

```
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 779:
    strcpy( mt->Name, "helium-3 + 29th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-29)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 780:
    strcpy( mt->Name, "helium-3 + 30th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-30)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 781:
    strcpy( mt->Name, "helium-3 + 31st excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-31)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 782:
    strcpy( mt->Name, "helium-3 + 32nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-32)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 783:
    strcpy( mt->Name, "helium-3 + 33rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-33)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 784:
    strcpy( mt->Name, "helium-3 + 34th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-34)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 785:
    strcpy( mt->Name, "helium-3 + 35th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-35)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 786:
    strcpy( mt->Name, "helium-3 + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 787:
    strcpy( mt->Name, "helium-3 + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 788:
    strcpy( mt->Name, "helium-3 + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 789:
    strcpy( mt->Name, "helium-3 + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 790:
    strcpy( mt->Name, "helium-3 + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 791:
    strcpy( mt->Name, "helium-3 + 41st excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-41)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 792:
    strcpy( mt->Name, "helium-3 + 42nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-42)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 793:
    strcpy( mt->Name, "helium-3 + 43rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-43)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 794:
    strcpy( mt->Name, "helium-3 + 44th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-44)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 795:
    strcpy( mt->Name, "helium-3 + 45th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-45)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 796:
    strcpy( mt->Name, "helium-3 + 46th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-46)" );
```

```
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 797:
    strcpy( mt->Name, "helium-3 + 47th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-47)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 798:
    strcpy( mt->Name, "helium-3 + 48th excited state residual" );
    strcpy( mt->Reaction, "(gamma, he3-48)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 799:
    strcpy( mt->Name, "helium-3 + continuum state residual" );
    strcpy( mt->Reaction, "(gamma, he3z)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2003;
    mt->ProductZA[1] = -2003;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;


    /*
    ** one alpha exit channel with excited residual
    */
  case 107:
    strcpy( mt->Name, "single alpha channel sum" );
    strcpy( mt->Reaction, "(gamma, alpha)" );
    mt->NumberOfProducts = -9;
    mt->ProductZA = NULL;
    mt->YieldOfProduct = NULL;
    break;

  case 800:
    strcpy( mt->Name, "alpha + ground state residual" );
    strcpy( mt->Reaction, "(gamma, alpha0)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 801:
    strcpy( mt->Name, "alpha + 1st excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha1)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 802:
    strcpy( mt->Name, "alpha + 2nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha2)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 803:
    strcpy( mt->Name, "alpha + 3rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha3)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 804:
    strcpy( mt->Name, "alpha + 4th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha4)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 805:
    strcpy( mt->Name, "alpha + 5th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha5)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 806:
    strcpy( mt->Name, "alpha + 6th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha6)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 807:
    strcpy( mt->Name, "alpha + 7th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha7)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 808:
    strcpy( mt->Name, "alpha + 8th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha8)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 809:
    strcpy( mt->Name, "alpha + 9th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha9)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 810:
    strcpy( mt->Name, "alpha + 10th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha10)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 811:
    strcpy( mt->Name, "alpha + 11th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha11)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 812:
    strcpy( mt->Name, "alpha + 12th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha12)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;
```

```
case 813:
  strcpy( mt->Name, "alpha + 13th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha13)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 814:
  strcpy( mt->Name, "alpha + 14th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha14)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 815:
  strcpy( mt->Name, "alpha + 15th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha15)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 816:
  strcpy( mt->Name, "alpha + 16th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha16)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 817:
  strcpy( mt->Name, "alpha + 17th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha17)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 818:
  strcpy( mt->Name, "alpha + 18th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha18)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;
```

```
case 819:
  strcpy( mt->Name, "alpha + 19th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha19)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 820:
  strcpy( mt->Name, "alpha + 20th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha20)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 821:
  strcpy( mt->Name, "alpha + 21st excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha21)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 822:
  strcpy( mt->Name, "alpha + 22nd excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha22)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 823:
  strcpy( mt->Name, "alpha + 23rd excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha23)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
  break;

case 824:
  strcpy( mt->Name, "alpha + 24th excited state residual" );
  strcpy( mt->Reaction, "(gamma, alpha24)" );
  mt->NumberOfProducts = 2;
  mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->ProductZA[0] = 2004;
  mt->ProductZA[1] = -2004;
  mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
  mt->YieldOfProduct[0] = 1;
  mt->YieldOfProduct[1] = 1;
```

```
    break;

  case 825:
    strcpy( mt->Name, "alpha + 25th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha25)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 826:
    strcpy( mt->Name, "alpha + 26th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha26)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 827:
    strcpy( mt->Name, "alpha + 27th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha27)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 828:
    strcpy( mt->Name, "alpha + 28th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha28)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 829:
    strcpy( mt->Name, "alpha + 29th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha29)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 830:
    strcpy( mt->Name, "alpha + 30th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha30)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 831:
    strcpy( mt->Name, "alpha + 31st excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha31)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 832:
    strcpy( mt->Name, "alpha + 32nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha32)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 833:
    strcpy( mt->Name, "alpha + 33rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha33)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 834:
    strcpy( mt->Name, "alpha + 34th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha34)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 835:
    strcpy( mt->Name, "alpha + 35th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha35)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 836:
    strcpy( mt->Name, "alpha + 36th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha36)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 837:
    strcpy( mt->Name, "alpha + 37th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha37)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 838:
    strcpy( mt->Name, "alpha + 38th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha38)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 839:
    strcpy( mt->Name, "alpha + 39th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha39)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 840:
    strcpy( mt->Name, "alpha + 40th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha40)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 841:
    strcpy( mt->Name, "alpha + 41st excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha41)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 842:
    strcpy( mt->Name, "alpha + 42nd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha42)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 843:
    strcpy( mt->Name, "alpha + 43rd excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha43)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 844:
    strcpy( mt->Name, "alpha + 44th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha44)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 845:
    strcpy( mt->Name, "alpha + 45th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha45)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 846:
    strcpy( mt->Name, "alpha + 46th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha46)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 847:
    strcpy( mt->Name, "alpha + 47th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha47)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 848:
    strcpy( mt->Name, "alpha + 48th excited state residual" );
    strcpy( mt->Reaction, "(gamma, alpha48)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
```

```
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  case 849:
    strcpy( mt->Name, "alpha + continuum state residual" );
    strcpy( mt->Reaction, "(gamma, alphaz)" );
    mt->NumberOfProducts = 2;
    mt->ProductZA = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->ProductZA[0] = 2004;
    mt->ProductZA[1] = -2004;
    mt->YieldOfProduct = calloc( mt->NumberOfProducts, sizeof( double ) );
    mt->YieldOfProduct[0] = 1;
    mt->YieldOfProduct[1] = 1;
    break;

  default:
    printf( "ERROR: unexpected case; got mt number %d\n", mt->Number );
    break;

  }

}
```

afeGetMTProducts.c

```
#include "endf6.h"
#include "acepnData.h"

void afeGetMTProducts( endfMaterialInformation *mi, aceTable *table );
void afeMF5toLawData( endfMF5Distribution *mf5, aceLawInformation *li );
void afeMF6toLawData( endfSecondary *sec, aceLawInformation *li );
void afeComputeCumulativeProbability( int law, int d, void *lawdata,
                                      int Interpolation );
void afeComputeKalbachSlopeParameter( endfMaterialInformation *mi,
                                    aceTable *table, aceLaw44 *law44,
                                    int pOutZA, double AWRout );
double afeComputeBindingEnergy( int cnZA, int pZA );
int afeIntCompare( const void *v1, const void *v2 );
int afeZAtoIPT( int ZA );
double aceComputeYield( double energy, aceYieldInformation *yieldinfo );
double aceInterpretPoint( int interp, double energy, double *egrid, double *vgrid );

int aceFindMT( int number, int numberofmts, aceMTInformation **mt );


/*********************************************************************
** afeGetMTProducts
**
*/
void afeGetMTProducts( endfMaterialInformation *mi, aceTable *table )
{
  int   i, j, k, l, m, n;
  int   end;
  int   count;
  int  *mtcount;
  int  *zalist;

  double  mega = 1.0e6;

  acepnData            *data = (acepnData*)table->Data;
  aceProduct           *tpProd;
  aceMTReference       *mtref;
  aceMTInformation     *mt;
  aceYieldInformation  *yielddata;
  aceEmissionData      *emitdata;

  endfMF1T *tnu;
```

```
endfMF1P *pnu;
endfMF1  *mf1data;
endfMF4  *mf4data;
endfMF5  *mf5data;
endfMF6  *mf6data;


/*
** Count the maximum number of secondaries if every one was different
*/
for( i = 0, count = 0; i < mi->NumberOfRecords; i++ ) {
  /* only one product, neutrons, can be specified by mf5 */
  if( mi->Records[i]->MF == 5 )
    count++;
  /* explicit number of products from mf6 description */
  else if( mi->Records[i]->MF == 6 ) {
    mf6data = (endfMF6*)(mi->Records[i]->MFMT);
    count += mf6data->NumberOfSubsections;
  }
}

/*
** create a list of all secondaries specified
*/
data->NumberOfProducts = count;
zalist = (int*)calloc( count, sizeof( int ) );

for( i = 0, count = 0; i < mi->NumberOfRecords; i++ ) {
  if( mi->Records[i]->MF == 5 )
    zalist[count++] = 1;
  else if( mi->Records[i]->MF == 6 ) {
    mf6data = (endfMF6*)(mi->Records[i]->MFMT);
    for( j = 0; j < mf6data->NumberOfSubsections; j++ )
      zalist[count++] = (int)(mf6data->Secondaries[j].ParticleZA + 0.000001);
  }
}

/*
** sort list least to greatest
*/
for( i = 0; i < count; i++ )
  for( j = i+1; j < count; j++ )
    if( zalist[j] < zalist[i] )
      qsort( (void*)zalist, (size_t)count, sizeof(int), &afeIntCompare );

/*
** remove duplicate entries from list
*/
for( i = 1; i < count; i++ )
  if( zalist[i] == zalist[i-1] ) {
    memmove( &zalist[i-1], &zalist[i], (count - i)*sizeof(int) );
    count--;
    i--;
  }

/*
** remove all za references greater than alpha (2004) for now
*/
while( zalist[count - 1] > 2004 && count > 0)
    count--;


/*
** create the product holders
*/
data->NumberOfProducts = count;
data->Product = (aceProduct**)calloc( count, sizeof( aceProduct* ) );
for( i = 0; i < count; i++ ) {
  data->Product[i] = (aceProduct*)calloc( 1, sizeof( aceProduct ) );
  data->Product[i]->ZA = zalist[i];
  data->Product[i]->IPT = afeZAtoIPT( zalist[i] );
```

```
}

/*
** free the temporary za listing
*/
free( zalist );

/*
** count the number of reactions involving each product
*/
for( i = 0; i < mi->NumberOfRecords; i++ ) {
  if( mi->Records[i]->MF == 5 ) {
    for( k = 0; k < data->NumberOfProducts; k++ )
      if( data->Product[k]->ZA == 1 )
        (data->Product[k]->NumberOfReactions)++;
  }
  else if( mi->Records[i]->MF == 6 ) {
    mf6data = (endfMF6*)(mi->Records[i]->MFMT);
    for( j = 0; j < mf6data->NumberOfSubsections; j++ ) {
      for( k = 0; k < data->NumberOfProducts; k++ )
        if( data->Product[k]->ZA
            == (int)(mf6data->Secondaries[j].ParticleZA + 0.000001) )
          (data->Product[k]->NumberOfReactions)++;
    }
  }
}

/*
** allocate space for references to said reactions
*/
for( i = 0; i < data->NumberOfProducts; i++ ) {
  data->Product[i]->MTReference = (aceMTReference**)
    calloc( data->Product[i]->NumberOfReactions, sizeof( aceMTReference* ) );
  for( j = 0; j < data->Product[i]->NumberOfReactions; j++ )
    data->Product[i]->MTReference[j]
      = (aceMTReference*)calloc( 1, sizeof( aceMTReference ) );
}

/*
** create a temp count for each product to ensure don't exceed count above
*/
mtcount = (int*)calloc( data->NumberOfProducts, sizeof( int ) );

/*
** connect the appropriate mt reference and fill in the yield and emission data
*/
for( i = 0; i < mi->NumberOfRecords; i++ ) {

  if( mi->Records[i]->MF == 5 ) {

    mf5data = (endfMF5*)(mi->Records[i]->MFMT);

    for( k = 0; k < data->NumberOfProducts; k++ )
      if( data->Product[k]->ZA == 1 )
        break;
    if( k >= data->NumberOfProducts )
      break;

    switch( mi->Records[i]->MT ) {
    case 4:
      l = aceFindMT( 91, data->NumberOfMTs, data->MT );
      break;
    case 103:
      l = aceFindMT( 649, data->NumberOfMTs, data->MT );
      break;
    case 104:
      l = aceFindMT( 699, data->NumberOfMTs, data->MT );
      break;
    case 105:
      l = aceFindMT( 749, data->NumberOfMTs, data->MT );
      break;
```

```
case 106:
 l = aceFindMT( 799, data->NumberOfMTs, data->MT );
 break;
case 107:
 l = aceFindMT( 849, data->NumberOfMTs, data->MT );
 break;
default:
 l = aceFindMT( mi->Records[i]->MT, data->NumberOfMTs, data->MT );
 break;
}
if( l < 0 ) {
 printf( "ERROR: mt reference by mf5 mt%d not found in ace table\n",
         mi->Records[i]->MT );
 exit( -1 );
}

if( mtcount[k] >= data->Product[k]->NumberOfReactions ) {
 printf( "ERROR: trying to fill more reactions that found\n" );
 exit( -1 );
}

data->Product[k]->MTReference[mtcount[k]]->Type = 5;
data->Product[k]->MTReference[mtcount[k]]->MT = data->MT[l];

data->Product[k]->MTReference[mtcount[k]]->Yield
 = (aceYieldInformation*)calloc( 1, sizeof( aceYieldInformation ) );
yielddata = data->Product[k]->MTReference[mtcount[k]]->Yield;

if( data->MT[l]->Number == 18 ) {
 for( m = 0; m < mi->NumberOfRecords; m++ )
   if( mi->Records[m]->MF == 1 ) {
     if( mi->Records[m]->MT == 452 ) {
       mf1data = (endfMF1*)(mi->Records[m]->MFMT);
       tnu = (endfMF1T*)mf1data->NuParameters;
       break;
     }
     else if( mi->Records[m]->MT == 456 ) {
       mf1data = (endfMF1*)(mi->Records[m]->MFMT);
       pnu = (endfMF1P*)mf1data->NuParameters;
       break;
     }
   }
 if( m == mi->NumberOfRecords ) {
   printf( "ERROR: fission cross section given without nubar\n" );
   exit( -1 );
 }

 if( mi->Records[m]->MT == 452 ) {

   if( tnu->NumberOfInterpolationRegions == 0 ) {
     printf( "ERROR: total nu with poly coeff. not currently supported\n" );
     exit( -1 );
   }

   printf( "NOTE: fission spectrum will use total nu\n" );

   yielddata->NumberOfRegions = tnu->NumberOfInterpolationRegions;
   yielddata->NumberOfPointsInRegion
     = (int*)calloc( tnu->NumberOfInterpolationRegions, sizeof( int ) );
   yielddata->InterpolationSchemeInRegion
     = (int*)calloc( tnu->NumberOfInterpolationRegions, sizeof( int ) );
   for( n = 0; n < tnu->NumberOfInterpolationRegions; n++ ) {
     yielddata->NumberOfPointsInRegion[n]
       = tnu->NumberOfPointsInRegion[n];
     yielddata->InterpolationSchemeInRegion[n]
       = tnu->InterpolationSchemeInRegion[n];
   }

   yielddata->NumberOfYields = tnu->NumberOfPoints;
   yielddata->Energy
     = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
```

```
      yielddata->Yield
        = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
      for( n = 0; n < yielddata->NumberOfYields; n++ ) {
        yielddata->Energy[n] = tnu->Energy[n] / mega;
        yielddata->Yield[n] = tnu->Nu[n];
      }
    }
    else if( mi->Records[m]->MT == 456 ) {

      printf( "NOTE: fission spectrum will use prompt nu\n" );

      yielddata->NumberOfRegions = pnu->NumberOfInterpolationRegions;
      yielddata->NumberOfPointsInRegion
        = (int*)calloc( pnu->NumberOfInterpolationRegions, sizeof( int ) );
      yielddata->InterpolationSchemeInRegion
        = (int*)calloc( pnu->NumberOfInterpolationRegions, sizeof( int ) );
      for( n = 0; n < pnu->NumberOfInterpolationRegions; n++ ) {
        yielddata->NumberOfPointsInRegion[n]
          = pnu->NumberOfPointsInRegion[n];
        yielddata->InterpolationSchemeInRegion[n]
          = pnu->InterpolationSchemeInRegion[n];
      }

      yielddata->NumberOfYields = pnu->NumberOfPoints;
      yielddata->Energy
        = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
      yielddata->Yield
        = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
      for( n = 0; n < yielddata->NumberOfYields; n++ ) {
        yielddata->Energy[n] = pnu->Energy[n] / mega;
        yielddata->Yield[n] = pnu->Nu[n];
      }
    }
  } /* end of if fission, set nu as yield */
  else {
    yielddata->NumberOfRegions = 1;
    yielddata->NumberOfPointsInRegion = (int*)calloc( 1, sizeof( int ) );
    yielddata->InterpolationSchemeInRegion = (int*)calloc( 1, sizeof( int ) );
    yielddata->NumberOfPointsInRegion[0] = 2;
    yielddata->InterpolationSchemeInRegion[0] = 2;

    yielddata->NumberOfYields = 2;
    yielddata->Energy = (double*)calloc( 2, sizeof( double ) );
    yielddata->Yield = (double*)calloc( 2, sizeof( double ) );
    yielddata->Energy[0] = data->Energy[0];
    yielddata->Energy[1] = data->Energy[data->NumberOfEnergies-1];
    for( m = 0; m < data->MT[l]->NumberOfProducts; m++ )
      if( data->MT[l]->ProductZA[m] == 1 )
        break;
    yielddata->Yield[0] = data->MT[l]->YieldOfProduct[m];
    yielddata->Yield[1] = data->MT[l]->YieldOfProduct[m];
  }

  data->Product[k]->MTReference[mtcount[k]]->Emit
    = (aceEmissionData*)calloc( 1, sizeof( aceEmissionData ) );
  emitdata = data->Product[k]->MTReference[mtcount[k]]->Emit;

  for( m = 0; m < mi->NumberOfRecords; m++ )
    if( mi->Records[m]->MF == 4
        && mi->Records[m]->MT == mi->Records[i]->MT )
      break;
  if( m >= mi->NumberOfRecords ) {
    printf( "ERROR: no corresponding mf4 data for mf5 mt%d\n",
            data->MT[l]->Number );
    exit( -1 );
  }

  /* set coordinate system -1 for CM or 1 for Lab */
  mf4data = (endfMF4*)mi->Records[m]->MFMT;
  if( mf4data->FrameOfReference == 3 ) {
    if( data->Product[k]->ZA > 2004 )
```

```
      emitdata->CoordinateSystem = 1;
    else
      emitdata->CoordinateSystem = -1;
  }
  else if( mf4data->FrameOfReference == 2 )
    emitdata->CoordinateSystem = -1;
  else
    emitdata->CoordinateSystem = 1;

  emitdata->AngularInformationType = 0;
  emitdata->AngularInformation = NULL;

  count = mf5data->NumberOfPartialEnergyDistributions;
  emitdata->NumberOfEnergyLaws = count;
  emitdata->LawInformation
    = (aceLawInformation*)calloc( count, sizeof( aceLawInformation ) );

  for( m = 0; m < count; m++ ) {

    emitdata->LawInformation[m].NumberOfRegions
      = mf5data->Distributions[m].NumberOfEnergyRegions;
    emitdata->LawInformation[m].NumberOfPointsInRegion
      = (int*)calloc( emitdata->LawInformation[m].NumberOfRegions,
                      sizeof( int ) );
    emitdata->LawInformation[m].InterpolationSchemeInRegion
      = (int*)calloc( emitdata->LawInformation[m].NumberOfRegions,
                      sizeof( int ) );
    for( n = 0; n < emitdata->LawInformation[m].NumberOfRegions; n++ ) {
      emitdata->LawInformation[m].NumberOfPointsInRegion[n]
        = mf5data->Distributions[m].NumberOfEnergyPointsInRegion[n];
       emitdata->LawInformation[m].InterpolationSchemeInRegion[n]
        = mf5data->Distributions[m].InterpolationSchemeInEnergyRegion[n];
    }

    emitdata->LawInformation[m].NumberOfEnergies
      = mf5data->Distributions[m].NumberOfEnergyPoints;
    emitdata->LawInformation[m].Energy
      = (double*)calloc( emitdata->LawInformation[m].NumberOfEnergies,
                      sizeof( double ) );
    emitdata->LawInformation[m].Probability
      = (double*)calloc( emitdata->LawInformation[m].NumberOfEnergies,
                      sizeof( double ) );
    for( n = 0; n < emitdata->LawInformation[m].NumberOfEnergies; n++ ) {
      emitdata->LawInformation[m].Energy[n]
        = mf5data->Distributions[m].Energy[n] / mega;
      emitdata->LawInformation[m].Probability[n]
        = mf5data->Distributions[m].EnergyProbability[n] * mega;
    }

    afeMF5toLawData( &(mf5data->Distributions[m]),
                     &(emitdata->LawInformation[m]) );
  }


  mtcount[k]++;

} /* end of if mf5 record */

else if( mi->Records[i]->MF == 6 ) {

  mf6data = (endfMF6*)(mi->Records[i]->MFMT);

  for( j = 0; j < mf6data->NumberOfSubsections; j++ ) {

    for( k = 0; k < data->NumberOfProducts; k++ )
      if( data->Product[k]->ZA
          == (int)(mf6data->Secondaries[j].ParticleZA + 0.000001) )
        break;
    if( k >= data->NumberOfProducts )
      continue;
```

```
switch( mi->Records[i]->MT ) {
case 4:
  l = aceFindMT( 91, data->NumberOfMTs, data->MT );
  break;
case 103:
  l = aceFindMT( 649, data->NumberOfMTs, data->MT );
  break;
case 104:
  l = aceFindMT( 699, data->NumberOfMTs, data->MT );
  break;
case 105:
  l = aceFindMT( 749, data->NumberOfMTs, data->MT );
  break;
case 106:
  l = aceFindMT( 799, data->NumberOfMTs, data->MT );
  break;
case 107:
  l = aceFindMT( 849, data->NumberOfMTs, data->MT );
  break;
default:
  l = aceFindMT( mi->Records[i]->MT, data->NumberOfMTs, data->MT );
  break;
}
if( l < 0 ) {
  printf( "ERROR: mt reference by mf6 mt%d not found in ace table\n",
          mi->Records[i]->MT );
  exit( -1 );
}

if( mtcount[k] >= data->Product[k]->NumberOfReactions ) {
  printf( "ERROR: trying to fill more reactions that found\n" );
  exit( -1 );
}

data->Product[k]->MTReference[mtcount[k]]->Type = 16;
data->Product[k]->MTReference[mtcount[k]]->MT = data->MT[l];

data->Product[k]->MTReference[mtcount[k]]->Yield
= (aceYieldInformation*)calloc( 1, sizeof( aceYieldInformation ) );
yielddata = data->Product[k]->MTReference[mtcount[k]]->Yield;

yielddata->NumberOfRegions = mf6data->Secondaries[j].NumberOfYieldRegions;
yielddata->NumberOfPointsInRegion
  = (int*)calloc( yielddata->NumberOfRegions, sizeof( int ) );
yielddata->InterpolationSchemeInRegion
  = (int*)calloc( yielddata->NumberOfRegions, sizeof( int ) );
for( m = 0; m < yielddata->NumberOfRegions; m++ ) {
  yielddata->NumberOfPointsInRegion[m]
    = mf6data->Secondaries[j].NumberOfYieldPointsInRegion[m];
  yielddata->InterpolationSchemeInRegion[m]
    = mf6data->Secondaries[j].InterpolationSchemeInYieldRegion[m];
}

yielddata->NumberOfYields = mf6data->Secondaries[j].NumberOfYieldPoints;
yielddata->Energy
  = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
yielddata->Yield
  = (double*)calloc( yielddata->NumberOfYields, sizeof( double ) );
for( m = 0; m < yielddata->NumberOfYields; m++ ) {
  yielddata->Energy[m] = mf6data->Secondaries[j].YieldEnergy[m] / mega;
  yielddata->Yield[m] = mf6data->Secondaries[j].Yield[m];
}

data->Product[k]->MTReference[mtcount[k]]->Emit
  = (aceEmissionData*)calloc( 1, sizeof( aceEmissionData ) );
emitdata = data->Product[k]->MTReference[mtcount[k]]->Emit;

/* set coordinate system -1 for CM or 1 for Lab */
if( mf6data->FrameOfReference == 3 ) {
  if( data->Product[k]->ZA > 2004 )
    emitdata->CoordinateSystem = 1;
```

```
        else
          emitdata->CoordinateSystem = -1;
      }
      else if( mf6data->FrameOfReference == 2 )
        emitdata->CoordinateSystem = -1;
      else
        emitdata->CoordinateSystem = 1;

      emitdata->AngularInformationType = 0;
      emitdata->AngularInformation = NULL;

      emitdata->NumberOfEnergyLaws = 1;
      emitdata->LawInformation
        = (aceLawInformation*)calloc( 1, sizeof( aceLawInformation ) );

      emitdata->LawInformation[0].NumberOfRegions = 1;
      emitdata->LawInformation[0].NumberOfPointsInRegion
        = (int*)calloc( 1, sizeof( int ) );
      emitdata->LawInformation[0].InterpolationSchemeInRegion
        = (int*)calloc( 1, sizeof( int ) );
      emitdata->LawInformation[0].NumberOfPointsInRegion[0] = 2;
      emitdata->LawInformation[0].InterpolationSchemeInRegion[0] = 2;

      emitdata->LawInformation[0].NumberOfEnergies = 2;
      emitdata->LawInformation[0].Energy
        = (double*)calloc( 2, sizeof( double ) );
      emitdata->LawInformation[0].Probability
        = (double*)calloc( 2, sizeof( double ) );
      emitdata->LawInformation[0].Energy[0]
        = data->Energy[0];
      emitdata->LawInformation[0].Energy[1]
        = data->Energy[data->NumberOfEnergies-1];
      emitdata->LawInformation[0].Probability[0] = 1;
      emitdata->LawInformation[0].Probability[1] = 1;

      afeMF6toLawData( &(mf6data->Secondaries[j]),
                       &(emitdata->LawInformation[0]) );

      if( emitdata->LawInformation[0].Number == 44 ) {
        afeComputeKalbachSlopeParameter( mi, table,
                    (aceLaw44*)emitdata->LawInformation[0].LawData,
                    data->Product[k]->ZA,
                    mf6data->Secondaries[j].ParticleAWR );
        emitdata->AngularInformationType = -1;
      }

      mtcount[k]++;

    } /* end of for j subsections of mf6 data */

  } /* end of else if mf6 record */

} /* end of for i records in endf material */

/*
** compute the production cross section for each product
*/
for( i = 0; i < data->NumberOfProducts; i++ ) {

  data->Product[i]->StartingIndex = 1;
  data->Product[i]->NumberOfEntries = data->NumberOfEnergies;

  data->Product[i]->ProductionCrossSection
    = (double*)calloc( data->NumberOfEnergies, sizeof( double ) );

  for( j = 0; j < data->Product[i]->NumberOfReactions; j++ ) {

    mtref = data->Product[i]->MTReference[j];

    for( k = mtref->MT->StartingIndex - 1;
         k < (mtref->MT->StartingIndex - 1 + mtref->MT->NumberOfEntries);
```

```
          k++ )
        data->Product[i]->ProductionCrossSection[k]
          += aceComputeYield( mtref->MT->Energy[k], mtref->Yield )
             * mtref->MT->CrossSection[k];
    }

    end = data->Product[i]->NumberOfEntries;
    for( j = 0; j < end; j++ ) {
      if( data->Product[i]->ProductionCrossSection[j] == 0
          && data->Product[i]->ProductionCrossSection[j+1] == 0 ) {
        (data->Product[i]->StartingIndex)++;
        (data->Product[i]->NumberOfEntries)--;
      }
      else
        j = end;
    }

  }

  for( i = 0; i < data->NumberOfProducts - 1; i++ )
    for( j = i+1; j < data->NumberOfProducts; j++ )
      if( data->Product[i]->IPT > data->Product[j]->IPT ) {
        tpProd = data->Product[j];
        data->Product[j] = data->Product[i];
        data->Product[i] = tpProd;
      }

}


/*********************************************************************
** aceComputeYield
**
*/
double aceComputeYield( double energy, aceYieldInformation *yieldinfo )
{
  int  i, j;
  int  points;


  for( i = 0; i < yieldinfo->NumberOfYields; i++ )
    if( energy == yieldinfo->Energy[i] )
      return yieldinfo->Yield[i];
    else if( energy < yieldinfo->Energy[i] )
      break;

  if( i == 0 ) {
    printf( "WARNING: received energy less than first yield energy\n" );
    return yieldinfo->Yield[0];
  }
  else if ( i == yieldinfo->NumberOfYields ) {
    printf( "WARNING: received energy greater than last yield energy\n" );
    return yieldinfo->Yield[yieldinfo->NumberOfYields-1];
  }
  else {
    points = yieldinfo->NumberOfPointsInRegion[0];
    for( j = 0; i > points && j < yieldinfo->NumberOfRegions; j++ )
         points += yieldinfo->NumberOfPointsInRegion[j];

    return aceInterpretPoint( yieldinfo->InterpolationSchemeInRegion[j],
                              energy, &(yieldinfo->Energy[i-1]),
                              &(yieldinfo->Yield[i-1]) );
  }

}


/*********************************************************************
** aceInterpretPoint
**
*/
```

```
double aceInterpretPoint( int interp, double energy, double *egrid, double *vgrid )
{
  switch( interp ) {

  case 1: /* histogram y in x */

    return vgrid[0];

  case 2: /* lin x lin y */

    return (  ( energy - egrid[0] )
            / ( egrid[1] - egrid[0] )
            * ( vgrid[1] - vgrid[0] )
            + vgrid[0]
           );

  case 3: /* log x log y */

    return ( exp(  log( energy / egrid[0] )
                 / log( egrid[1] / egrid[0] )
                 * log( vgrid[1] / vgrid[0] )
                 + log( vgrid[0])
                )
           );

  case 4: /* log x lin y */

    return (  log( energy / egrid[0] )
            / log( egrid[1] / egrid[0] )
            * ( vgrid[1] - vgrid[0] )
            + ( vgrid[0])
           );

  case 5: /* lin x log y */

    return ( exp(  ( energy - egrid[0] )
                 / ( egrid[1] - egrid[0] )
                 * log( vgrid[1] / vgrid[0] )
                 + log( vgrid[0])
                )
           );
  }
}


/*******************************************************************
** afeMF5toLawData
**
*/
void afeMF5toLawData( endfMF5Distribution *mf5d, aceLawInformation *li )
{
  int  i;

  double   mega = 1.0e6;

  aceLaw7  *law7;
  aceLaw9  *law9;

  endfMF5LF7  *mf5lf7;
  endfMF5LF9  *mf5lf9;


  switch( mf5d->EnergyDistributionLaw ) {

  case 1:
    break;

  case 5:
    break;

  case 7:
    li->Number = 7;
    li->Name = (char*)calloc( strlen("simple Maxwell fission spectrum") + 1,
                              sizeof( char ) );
    strcpy( li->Name, "simple Maxwell fission spectrum" );
    law7 = (aceLaw7*)calloc( 1, sizeof( aceLaw7 ) );
    li->LawData = (void*)law7;
    law7->RestrictionEnergy = mf5d->UpperEnergyDelta / mega;
    mf5lf7 = (endfMF5LF7*)mf5d->Parameters;
    law7->NumberOfRegions = mf5lf7->NumberOfThetaRegions;
    law7->NumberOfPointsInRegion
      = (int*)calloc( law7->NumberOfRegions, sizeof( int ) );
    law7->InterpolationSchemeInRegion
      = (int*)calloc( law7->NumberOfRegions, sizeof( int ) );
    for( i = 0; i < law7->NumberOfRegions; i++ ) {
      law7->NumberOfPointsInRegion[i]
        = mf5lf7->NumberOfThetaPointsInRegion[i];
      law7->InterpolationSchemeInRegion[i]
        = mf5lf7->InterpolationSchemeInThetaRegion[i];
    }
    law7->NumberOfIncidentEnergies = mf5lf7->NumberOfThetaPoints;
    law7->IncidentEnergy
      = (double*)calloc( law7->NumberOfIncidentEnergies, sizeof( double ) );
    law7->Temperature
      = (double*)calloc( law7->NumberOfIncidentEnergies, sizeof( double ) );
    for( i = 0; i < law7->NumberOfIncidentEnergies; i++ ) {
      law7->IncidentEnergy[i] = mf5lf7->ThetaEnergy[i] / mega;
      law7->Temperature[i] = mf5lf7->Theta[i];
    }
    break;

  case 9:
    li->Number = 9;
    li->Name = (char*)calloc( strlen("evaporation spectrum") + 1,
                              sizeof( char ) );
    strcpy( li->Name, "evaporation spectrum" );
    law9 = (aceLaw9*)calloc( 1, sizeof( aceLaw9 ) );
    li->LawData = (void*)law9;
    law9->RestrictionEnergy = mf5d->UpperEnergyDelta / mega;
    mf5lf9 = (endfMF5LF9*)mf5d->Parameters;
    law9->NumberOfRegions = mf5lf9->NumberOfThetaRegions;
    law9->NumberOfPointsInRegion
      = (int*)calloc( law9->NumberOfRegions, sizeof( int ) );
    law9->InterpolationSchemeInRegion
      = (int*)calloc( law9->NumberOfRegions, sizeof( int ) );
    for( i = 0; i < law9->NumberOfRegions; i++ ) {
      law9->NumberOfPointsInRegion[i]
        = mf5lf9->NumberOfThetaPointsInRegion[i];
      law9->InterpolationSchemeInRegion[i]
        = mf5lf9->InterpolationSchemeInThetaRegion[i];
    }
    law9->NumberOfIncidentEnergies = mf5lf9->NumberOfThetaPoints;
    law9->IncidentEnergy
      = (double*)calloc( law9->NumberOfIncidentEnergies, sizeof( double ) );
    law9->Temperature
      = (double*)calloc( law9->NumberOfIncidentEnergies, sizeof( double ) );
    for( i = 0; i < law9->NumberOfIncidentEnergies; i++ ) {
      law9->IncidentEnergy[i] = mf5lf9->ThetaEnergy[i] / mega;
      law9->Temperature[i] = mf5lf9->Theta[i];
    }
    break;

  case 11:
    break;

  case 12:
    break;

  default:
    break;

  }
```

```
}


/******************************************************************
** afeMF6toLawData
**
*/
void afeMF6toLawData( endfSecondary *sec, aceLawInformation *li )
{
  int  i, j;

  double   mega = 1.0e6;

  aceLaw4   *law4;
  aceLaw44  *law44;

  endfLaw1  *elaw1;


  switch( sec->ReactionLaw ) {

  case 1:

    elaw1 = (endfLaw1*)sec->LawData;

    switch( elaw1->AngularRepresentation ) {
    case 1:
      li->Number = 4;
      li->Name = (char*)calloc( strlen("continuous tabular spectrum")+1,
                                sizeof( char ) );
      strcpy( li->Name, "continuous tabular spectrum" );
      law4 = (aceLaw4*)calloc( 1, sizeof( aceLaw4 ) );
      li->LawData = (void*)law4;
      law4->NumberOfRegions = elaw1->NumberOfSecEnergyRegions;
      law4->NumberOfPointsInRegion
        = (int*)calloc( law4->NumberOfRegions, sizeof( int ) );
      law4->InterpolationSchemeInRegion
        = (int*)calloc( law4->NumberOfRegions, sizeof( int ) );
      for( i = 0; i < law4->NumberOfRegions; i++ ) {
        law4->NumberOfPointsInRegion[i]
          = elaw1->NumberOfSecEnergyPointsInRegion[i];
        law4->InterpolationSchemeInRegion[i]
          = elaw1->InterpolationSchemeInSecEnergyRegion[i];
      }
      law4->NumberOfIncidentEnergies = elaw1->NumberOfSecEnergyPoints;
      law4->Distribution = (aceLaw4Distribution*)
        calloc( law4->NumberOfIncidentEnergies, sizeof( aceLaw4Distribution ) );
      for( i = 0; i < law4->NumberOfIncidentEnergies; i++ ) {

        if( elaw1->ES[i].NumberOfAngularParameters != 0 ) {
          printf( "ERROR: MF6 Law 1, Lang 1 and NA not 0\n" );
          printf( "       Will process, but ignore angular information\n" );
        }

        law4->Distribution[i].IncidentEnergy = elaw1->ES[i].IncidentEnergy / mega;
        law4->Distribution[i].NumberOfDiscreteEmissions
          = elaw1->ES[i].NumberOfDiscreteEmissions;
        law4->Distribution[i].InterpolationScheme
          = elaw1->InterpolationSchemeForSecEnergy;

        law4->Distribution[i].NumberOfPoints
          = elaw1->ES[i].NumberOfEmissionEnergies;
        law4->Distribution[i].EmissionEnergy = (double*)
          calloc( law4->Distribution[i].NumberOfPoints, sizeof( double ) );
        law4->Distribution[i].Probability = (double*)
          calloc( law4->Distribution[i].NumberOfPoints, sizeof( double ) );
        law4->Distribution[i].CumulativeProbability = (double*)
          calloc( law4->Distribution[i].NumberOfPoints, sizeof( double ) );
        for( j = 0; j < law4->Distribution[i].NumberOfPoints; j++ ) {
          law4->Distribution[i].EmissionEnergy[j]
        = elaw1->ES[i].EmissionEnergy[j] / mega;
      law4->Distribution[i].Probability[j]
        = elaw1->ES[i].Parameters[j][0] * mega;
    }

    afeComputeCumulativeProbability( 4, i, (void*)law4,
                 elaw1->InterpolationSchemeForSecEnergy );
  }
  break;

case 2:
  li->Number = 44;
  li->Name = (char*)
    calloc( strlen("correlated Kalbach energy-angle spectrum")+1,
            sizeof( char ) );
  strcpy( li->Name, "correlated Kalbach energy-angle spectrum" );
  law44 = (aceLaw44*)calloc( 1, sizeof( aceLaw44 ) );
  li->LawData = (void*)law44;

  law44->NumberOfRegions = elaw1->NumberOfSecEnergyRegions;
  law44->NumberOfPointsInRegion
    = (int*)calloc( law44->NumberOfRegions, sizeof( int ) );
  law44->InterpolationSchemeInRegion
    = (int*)calloc( law44->NumberOfRegions, sizeof( int ) );
  for( i = 0; i < law44->NumberOfRegions; i++ ) {
    law44->NumberOfPointsInRegion[i]
      = elaw1->NumberOfSecEnergyPointsInRegion[i];
    law44->InterpolationSchemeInRegion[i]
      = elaw1->InterpolationSchemeInSecEnergyRegion[i];
  }

  law44->NumberOfIncidentEnergies = elaw1->NumberOfSecEnergyPoints;
  law44->Distribution = (aceLaw44Distribution*)
    calloc( law44->NumberOfIncidentEnergies, sizeof( aceLaw44Distribution ) );
  for( i = 0; i < law44->NumberOfIncidentEnergies; i++ ) {

    if( elaw1->ES[i].NumberOfAngularParameters != 1 ) {
      printf( "ERROR: MF6 Law 1, Lang 1 and NA not 1\n" );
      printf( "       Will process, but ignore angular information\n" );
    }

    law44->Distribution[i].IncidentEnergy = elaw1->ES[i].IncidentEnergy / mega;
    law44->Distribution[i].NumberOfDiscreteEmissions
      = elaw1->ES[i].NumberOfDiscreteEmissions;
    law44->Distribution[i].InterpolationScheme
      = elaw1->InterpolationSchemeForSecEnergy;

    law44->Distribution[i].NumberOfPoints
      = elaw1->ES[i].NumberOfEmissionEnergies;
    law44->Distribution[i].EmissionEnergy = (double*)
      calloc( law44->Distribution[i].NumberOfPoints, sizeof( double ) );
    law44->Distribution[i].Probability = (double*)
      calloc( law44->Distribution[i].NumberOfPoints, sizeof( double ) );
    law44->Distribution[i].CumulativeProbability = (double*)
      calloc( law44->Distribution[i].NumberOfPoints, sizeof( double ) );
    law44->Distribution[i].PrecompoundFraction = (double*)
      calloc( law44->Distribution[i].NumberOfPoints, sizeof( double ) );
    law44->Distribution[i].AngularDistributionSlope = (double*)
      calloc( law44->Distribution[i].NumberOfPoints, sizeof( double ) );

    for( j = 0; j < law44->Distribution[i].NumberOfPoints; j++ ) {
      law44->Distribution[i].EmissionEnergy[j]
        = elaw1->ES[i].EmissionEnergy[j] / mega;
      law44->Distribution[i].Probability[j]
        = elaw1->ES[i].Parameters[j][0] * mega;
      law44->Distribution[i].PrecompoundFraction[j]
        = elaw1->ES[i].Parameters[j][1];
    }

    afeComputeCumulativeProbability( 44, i, (void*)law44,
                  elaw1->InterpolationSchemeForSecEnergy );
      }
      break;
    case 11:
    case 12:
    case 13:
    case 14:
    case 15:
    default:
      printf( "ERROR: don't know case lang %d in law 1 yet\n",
              elaw1->AngularRepresentation );
    }
    break;

  default:
    printf( "ERROR: don't know case law %d yet\n", sec->ReactionLaw );
    break;
  }
}

/*******************************************************************
** afeComputeKalbachSlopeParameter
**
*/
void afeComputeKalbachSlopeParameter( endfMaterialInformation *mi,
                                      aceTable *table, aceLaw44 *law44,
                                      int pOutZA, double AWRout )
{
  int  i, j;
  int  pInZA, tnZA, cnZA, photonuclear;

  double  Sa, Sb, Ia, Ib, Ea, Eb, EMa, EMb;
  double  AWRtn, AWRrn, AWRin;
  double  E1, E3;


  /*
  ** if photon incident on nucleus, treat as neutron and modify
  **   in accordance with Chadwick et. al., J. Nuc. Sci. & Tech.,
  **   Vol. 32, No. 11, pp. 1154-1158.
  */
  if( mi->IncidentParticleZA == 0 ) {
    pInZA = 1;
    AWRin = 1.0;
    photonuclear = 1;
  }
  else {
    pInZA = mi->IncidentParticleZA;
    AWRin = mi->IncidentParticleAWR;
    photonuclear = 0;
  }

  /* the offset is to ensure that ZA is not rounded down */
  tnZA = (int)(mi->TargetZA + 0.000001);

  cnZA = ( tnZA / 1000 + pInZA / 1000 ) * 1000
           + (tnZA % 1000 + pInZA % 1000 );

  Sa = afeComputeBindingEnergy( cnZA, pInZA );
  Sb = afeComputeBindingEnergy( cnZA, pOutZA );

  switch( pInZA ) {
  case 1:
  case 1001:
  case 1002:
    EMa = 1.0;
    break;
  case 2004:
    EMa = 0.0;
    break;
  default:
    printf( "ERROR: can't find EMa for ZA %d\n", pInZA );
    exit( -1 );
  }

  switch( pOutZA ) {
  case 1:
    EMb = 0.5;
    break;
  case 1001:
  case 1002:
  case 1003:
  case 2003:
    EMb = 1.0;
    break;
  case 2004:
    EMb = 2.0;
    break;
  default:
    printf( "ERROR: can't find EMa for ZA %d\n", pInZA );
    exit( -1 );
  }

  AWRtn  = mi->TargetAWR;
  AWRrn  = AWRin + AWRtn - AWRout;

  for( i = 0; i < law44->NumberOfIncidentEnergies; i++ ) {

    Ea = law44->Distribution[i].IncidentEnergy * AWRtn
           / (AWRtn + AWRin) + Sa;

    if( Ea > 130.0 )
      E1 = 130.0;
    else
      E1 = Ea;

    if( Ea > 41.0 )
      E3 = 41.0;
    else
      E3 = Ea;

    for( j = 0; j < law44->Distribution[i].NumberOfPoints; j++ ) {

      Eb = law44->Distribution[i].EmissionEnergy[j] * (AWRrn + AWRout)
             / AWRrn + Sb;

      law44->Distribution[i].AngularDistributionSlope[j]
        =    0.04 * E1 * Eb / Ea
         + 1.8e-6 * pow( E1 * Eb / Ea, 3.0 )
         + 6.7e-7 * EMa * EMb * pow( E3 * Eb / Ea, 4.0 );
    }
  }

  if( photonuclear ) {
    for( i = 0; i < law44->NumberOfIncidentEnergies; i++ ) {
      for( j = 0; j < law44->Distribution[i].NumberOfPoints; j++ ) {
        E1 = sqrt( law44->Distribution[i].IncidentEnergy / 1878.0 );
        E3 = 9.3 / sqrt( law44->Distribution[i].EmissionEnergy[j] );
        if( E3 > 4.0 ) E3 = 4.0;
        if( E3 < 1.0 ) E3 = 1.0;
        law44->Distribution[i].AngularDistributionSlope[j] *= E1*E3;
      }
    }
  }

}


/*********************************************************************
** afeComputeBindingEnergy
**
*/
double afeComputeBindingEnergy( int cnZA, int pZA )
```

```
{

  double  AC, ZC, NC, AA, ZA, NA, I;
  double  S, S1, S2, S3, S4, S5, S6;


  ZC = cnZA / 1000;
  AC = cnZA % 1000;
  NC = AC - ZC;

  ZA = ZC - pZA / 1000;
  AA = AC - pZA % 1000;
  NA = AA - ZA;

  switch( pZA ) {
  case 1:
  case 1001:
    I = 0.0;
    break;
  case 1002:
    I = 2.22;
    break;
  case 1003:
    I = 8.48;
    break;
  case 2003:
    I = 7.72;
    break;
  case 2004:
    I = 28.3;
    break;
  default:
    printf( "ERROR: can't find nucleon binding energy I",
            " in separation energy for ZA %d\n", pZA );
    exit( -1 );
  }

  S1  = ( AC - AA );

  S2  = ( NC - ZC ) * ( NC - ZC ) / AC;
  S2 -= ( NA - ZA ) * ( NA - ZA ) / AA;

  S3  = pow( AC, (2./3.) ) - pow( AA, (2./3.) );

  S4  = ( NC - ZC ) * ( NC - ZC ) / pow( AC, (4./3.) );
  S4 -= ( NA - ZA ) * ( NA - ZA ) / pow( AA, (4./3.) );

  S5  = ZC * ZC / pow( AC, (1./3.) );
  S5 -= ZA * ZA / pow( AA, (1./3.) );

  S6  = ( ZC * ZC / AC ) - ( ZA * ZA / AA );

  S = 15.68*S1 - 28.07*S2 - 18.56*S3 + 33.22*S4 -0.717*S5 +1.211*S6 - I;

  return S;

}


/*******************************************************************
** afeComputeCumulativeProbability
**
*/
void afeComputeCumulativeProbability( int law, int d, void *lawdata,
                                      int Interpolation )
{
  int  i;
  int  count;

  double   temp1, temp2;
  double   runtot;
  double  *energy;
  double  *prob;
  double  *cumprob;

  aceLaw4   *law4;
  aceLaw44  *law44;

  switch( law ) {
  case 4:
    count = ((aceLaw4*)lawdata)->Distribution[d].NumberOfPoints;
    energy = ((aceLaw4*)lawdata)->Distribution[d].EmissionEnergy;
    prob = ((aceLaw4*)lawdata)->Distribution[d].Probability;
    cumprob = ((aceLaw4*)lawdata)->Distribution[d].CumulativeProbability;
    break;
  case 44:
    count = ((aceLaw44*)lawdata)->Distribution[d].NumberOfPoints;
    energy = ((aceLaw44*)lawdata)->Distribution[d].EmissionEnergy;
    prob = ((aceLaw44*)lawdata)->Distribution[d].Probability;
    cumprob = ((aceLaw44*)lawdata)->Distribution[d].CumulativeProbability;
    break;
  default:
    printf( "ERROR: can't compute cumulative probability for law %d\n", law );
    exit( -1 );
  }


  if( Interpolation != 1 ) {  /* Linear-linear interpolation */
    for( i = 1, runtot = 0; i < count; i++ )
      runtot += .5 * ( prob[i] + prob[i-1] ) * ( energy[i] - energy[i-1] );
  }
  else { /* Histogram interpolation */
    for( i = 1, runtot = 0; i < count; i++ )
      runtot += prob[i-1] * ( energy[i] - energy[i-1] );
  }

  if( abs( runtot - 1.0 ) > 0.01 )
    printf( "WARNING: distribution %d not normalized\n", d );

  /* Renormalize all distributions for most accurate machine computation */
  for( i = 0; i < count; i++ )
    prob[i] = prob[i] / runtot;


  if( Interpolation != 1 ) {
    for( i = 1, cumprob[0] = 0.0; i < count; i++ )
      cumprob[i] = cumprob[i-1]
        +  .5 * ( prob[i] + prob[i-1] ) * ( energy[i] - energy[i-1] );
  }
  else {
    for( i = 1, cumprob[0] = 0.0; i < count; i++ )
      cumprob[i] = cumprob[i-1] + prob[i-1] * ( energy[i] - energy[i-1] );
  }

}


/*********************************************************************
** afeZAtoIPT
**
*/
int afeZAtoIPT( int ZA )
{
  switch( ZA ) {
  case 0:
    return 2;
  case 1:
    return 1;
  case 1000:
    return 3;
  case 1001:
    return 9;
  case 1002:
    return 31;
  case 1003:
    return 32;
  case 2003:
    return 33;
  case 2004:
    return 34;
  default:
    printf( "ERROR: afeZAtoIPT:can't convert ZA '%d' to an IPT\n", ZA );
    return( -1 );
  }

}


/*****************************************************************
** afeIntCompare
**
*/
int afeIntCompare( const void *v1, const void *v2 )
{
  if( *((int*)v1) < *((int*)v2) )
    return -1;
  else if( *((int*)v1) > *((int*)v2) )
    return 1;
  else
    return 0;

}
```

afeMakeNTable.c

```
#include "endf6.h"
#include "acepnData.h"


void afeMakeNTable( endfMaterialInformation *mi, aceTable *table );

void afeCollectEnergies( endfMaterialInformation *mi,
                        int *count, double **energy );
void afeGetMTInformation( endfMaterialInformation *mi, acepnData *data );
void afeGetMTProducts( endfMaterialInformation *mi, aceTable *table );
void afeCreateNTableHeader( endfMaterialInformation *mi, aceTable *table );
void afeVerifyNTable( aceTable *table );



/*****************************************************************
** afeMakeNTable
**
** Take an existing endf photonuclear data set and create a compact
**   evaluation in the photonuclear 'n' format (ACE-PN)
**
*/
void afeMakeNTable( endfMaterialInformation *mi, aceTable *table )
{
  acepnData  *data = (acepnData*)calloc( 1, sizeof( acepnData ) );


  table->Data = (void*)data;

  /*
  ** Create the Table header information
  */
  afeCreateNTableHeader( mi, table );


  /*
  ** Create the superset energy grid
  */
  afeCollectEnergies( mi, &(data->NumberOfEnergies), &(data->Energy) );


  /*
  ** Transfer the mt information to the appropriate fields
  */
  afeGetMTInformation( mi, (acepnData*)(table->Data) );


  /*
  ** Create the Product fields
  */
  afeGetMTProducts( mi, table );


  /*
  ** verify any cross checks and fill in all the "pointer" variables
  */
  afeVerifyNTable( table );


}
```

afeVerifyNTable.c

```
#include "endf6.h"
#include "acepnData.h"

#include <time.h>


void afeVerifyNTable( aceTable *table );


/*******************************************************************
** afeVerifyNTable
*/
void afeVerifyNTable( aceTable *table )
{
  int  i, j, k, l;

  acepnData             *ndata = (acepnData*)(table->Data);
  aceProduct            *prod;
  aceMTReference        *mtref;
  aceLawInformation     *lawinfo;
  aceLaw4               *law4;
  aceLaw4Distribution   *law4d;
  aceLaw7               *law7;
  aceLaw9               *law9;
  aceLaw44              *law44;
  aceLaw44Distribution  *law44d;


  /*
  ** Initialize the NXS & JXS arrays
  */
  for( i = 0; i < 16; i++ )
    table->NXS[i] = 0;
  for( i = 0; i < 32; i++ )
    table->JXS[i] = 0;

  /*
  ** set the nxs values
  */
  table->NXS[0] = 0; /* no XSS entries yet; set as proceeds */
  table->NXS[1] = table->ZA;
  table->NXS[2] = ndata->NumberOfEnergies;
  table->NXS[3] = ndata->NumberOfMTs;
  table->NXS[4] = ndata->NumberOfProducts;
```

```
/*
** set the jxs values
*/

/*
** *** ESZ JXS(1) ***
** arrange energy grid locator
*/
table->JXS[0] = table->NXS[0] + 1;
table->NXS[0] += table->NXS[2];

/*
** *** TOT JXS(2) ***
** arrange total cross section locator
*/
table->JXS[1] = table->NXS[0] + 1;
table->NXS[0] += table->NXS[2];

/*
** *** NON JXS(3) ***
** if present and unique, arrange non-elastic cross section locator
*/
if( ndata->NonelasticCrossSection != NULL ) {
  table->JXS[2] = table->NXS[0] + 1;
  table->NXS[0] += table->NXS[2];
}
else
  table->JXS[2] = table->JXS[1];

/*
** *** ELS JXS(4) ***
** if present, arrange elastic cross section locator
*/
if( ndata->ElasticCrossSection != NULL ) {
  table->JXS[3] = table->NXS[0] + 1;
  table->NXS[0] += table->NXS[2];
}
else
  table->JXS[3] = 0;

/*
** *** HTN JXS(5) ***
** if present, arrange total heating number locator
*/
if( ndata->TotalHeatingNumber != NULL ) {
  table->JXS[4] = table->NXS[0] + 1;
  table->NXS[0] += table->NXS[2];
}
else
  table->JXS[4] = 0;

/*
** *** MTR JXS(6) ***
** arrange the MT number entries
*/
table->JXS[5] = table->NXS[0] + 1;
table->NXS[0] += table->NXS[3];

/*
** *** LQR JXS(7) ***
** arrange the Q value entries
*/
table->JXS[6] = table->NXS[0] + 1;
table->NXS[0] += table->NXS[3];

/*
** *** LSIG JXS(8) ***
** arrange the cross section offset entries
*/
table->JXS[7] = table->NXS[0] + 1;
```

```
table->NXS[0] += table->NXS[3];

/*
** *** SIG JXS(9) ***
** calculate the first word of the cross section (SIG) table and
** for each MT reaction cross setion, calculate offset
*/
table->JXS[8] = table->NXS[0] + 1;
ndata->MTLocator = (int*)calloc( ndata->NumberOfMTs, sizeof( int ) );
for( i = 0; i < ndata->NumberOfMTs; i++ ) {
  ndata->MTLocator[i] = table->NXS[0] + 1 - table->JXS[8] + 1;
  table->NXS[0] += ( 2 + ndata->MT[i]->NumberOfEntries );
}

/*
** *** IXSA JXS(10) ***
** calculate the first word of the IXS array entries
*/
table->JXS[9] = table->NXS[0] + 1;
table->NXS[0] += ( table->NXS[4] * NUMBER_IXS_ENTRIES );

/*
** *** IXS JXS(11) ***
** calculate the first word of the IXS block entries
*/
table->JXS[10] = table->NXS[0] + 1;


/*
** set the ixs values for each product particle
*/
for( i = 0; i < ndata->NumberOfProducts; i++ ) {

  prod = ndata->Product[i];

  /*
  ** *** IPT IXS(1) ***
  ** store the ipt number of this particle
  */
  prod->IXS[0] = prod->IPT;

  /*
  ** *** NTRP IXS(2) ***
  ** store the number of reactions creating this particle
  */
  prod->IXS[1] = prod->NumberOfReactions;

  /*
  ** *** PXS IXS(3) ***
  ** arrange the production cross section entries
  */
  prod->IXS[2] = table->NXS[0] + 1;
  table->NXS[0] += ( 2 + prod->NumberOfEntries );

  /*
  ** *** PHN IXS(4) ***
  ** if present, arrange the partial heating number entries
  */
  if( table->JXS[4] != 0 && prod->PartialHeatingNumber != NULL ) {
    prod->IXS[3] = table->NXS[0] + 1;
    table->NXS[0] += ( 2 + prod->NumberOfEntries );
  }
  else
    prod->IXS[3] = 0;

  /*
  ** *** MTRP IXS(5) ***
  ** arrange the MT reference entries
  */
  prod->IXS[4] = table->NXS[0] + 1;
  table->NXS[0] += prod->NumberOfReactions;
```

```
/*
** *** TYRP IXS(6) ***
** arrange the Coordinate System entries
*/
prod->IXS[5] = table->NXS[0] + 1;
table->NXS[0] += prod->NumberOfReactions;

/*
** *** LSIGP IXS(7) ***
** arrange the cross section or yield information offset entries
*/
prod->IXS[6] = table->NXS[0] + 1;
table->NXS[0] += prod->NumberOfReactions;

/*
** *** SIGP IXS(8) ***
** arrange the cross section or yield information locator
*/
prod->IXS[7] = table->NXS[0] + 1;

/*
** arrange the cross section offsets for each MT reference
*/
for( j = 0; j < prod->NumberOfReactions; j++ ) {

  mtref = prod->MTReference[j];

  switch( mtref->Type ) {

  case 5:
  case 12:
  case 16:
    mtref->Offset = table->NXS[0] + 1 - prod->IXS[7] + 1;
    table->NXS[0] += ( 4 + 2 * mtref->Yield->NumberOfRegions
                        + 2 * mtref->Yield->NumberOfYields );
    break;

  case 13:
    mtref->Offset = table->NXS[0] + 1 - prod->IXS[7] + 1;
    table->NXS[0] += ( 3 + mtref->MT->NumberOfEntries );
    break;
  }
}

/*
** *** LANDP IXS(9) ***
** arrange the angular offset information
*/
prod->IXS[8] = table->NXS[0] + 1;
table->NXS[0] += prod->NumberOfReactions;

/*
** *** ANDP IXS(10) ***
** currently no angular data is allowed
*/
prod->IXS[9] = 0;

/*
** *** LDLWP IXS(11) ***
** arrange the emission offset information
*/
prod->IXS[10] = table->NXS[0] + 1;
table->NXS[0] += prod->NumberOfReactions;

/*
** *** LDLW IXS(12) ***
** arrange the emission data locator
*/
prod->IXS[11] = table->NXS[0] + 1;
```

```
/*
** arrange the emission offsets for each MT reference
*/
for( j = 0; j < prod->NumberOfReactions; j++ ) {

  mtref = prod->MTReference[j];

  mtref->Emit->Offset = table->NXS[0] + 1 - prod->IXS[11] + 1;

  /*
  ** for each law, arrange appropriate offsets
  */
  for( k = 0; k < mtref->Emit->NumberOfEnergyLaws; k++ ) {

    lawinfo = &(mtref->Emit->LawInformation[k]);

    table->NXS[0] += ( 5 + 2 * lawinfo->NumberOfRegions
                         + 2 * lawinfo->NumberOfEnergies );
    lawinfo->OffsetToLawData = table->NXS[0] + 1 - prod->IXS[11] + 1;

    switch( lawinfo->Number ) {

    case 4:
      law4 = (aceLaw4*)(lawinfo->LawData);

      table->NXS[0] += ( 2 + 2 * law4->NumberOfRegions
                           + 2 * law4->NumberOfIncidentEnergies );

      for( l = 0; l < law4->NumberOfIncidentEnergies; l++ ) {
        law4d = &(law4->Distribution[l]);

        law4d->Offset = table->NXS[0] + 1 - prod->IXS[11] + 1;
        table->NXS[0] += ( 2 + 3 * law4d->NumberOfPoints );
      }
      break;

    case 7:
      law7 = (aceLaw7*)(lawinfo->LawData);

      table->NXS[0] += ( 3 + 2 * law7->NumberOfRegions
                           + 2 * law7->NumberOfIncidentEnergies );
      break;

    case 9:
      law9 = (aceLaw9*)(lawinfo->LawData);

      table->NXS[0] += ( 3 + 2 * law9->NumberOfRegions
                           + 2 * law9->NumberOfIncidentEnergies );
      break;

    case 44:
      law44 = (aceLaw44*)(lawinfo->LawData);

      table->NXS[0] += ( 2 + 2 * law44->NumberOfRegions
                           + 2 * law44->NumberOfIncidentEnergies );

      for( l = 0; l < law44->NumberOfIncidentEnergies; l++ ) {
        law44d = &(law44->Distribution[l]);

        law44d->Offset = table->NXS[0] + 1 - prod->IXS[11] + 1;
        table->NXS[0] += ( 2 + 5 * law44d->NumberOfPoints );
      }
      break;
    }

    if( k == mtref->Emit->NumberOfEnergyLaws )
      lawinfo->LocationOfNextLaw = table->NXS[0] + 1;
    else
      lawinfo->LocationOfNextLaw = 0;
      } /* end of loop on energy laws */

    } /* end of loop on number of reactions for product */

  } /* end of loop on number of products */

}
```

endf6.h

```
#ifndef ENDF6_h
#define ENDF6_h

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "endfLine.h"
#include "endfNumber.h"
#include "endfConvert.h"

#include "endfMF1.h"
#include "endfMF2.h"
#include "endfMF3.h"
#include "endfMF4.h"
#include "endfMF5.h"
#include "endfMF6.h"

#include "endfMF1MT451.h"


/*
** Function: endfReadMaterialFromFile
*/
int endfReadMaterialFromFile( char *filename, endfMaterialInformation *mi );

/*
** Function: endfPrintMaterialToFile
*/
int endfPrintMaterialToFile( char *filename, endfMaterialInformation *mi );


#endif
```

endf6.c

```
#include "endf6.h"

/*********************************************************************
** endfReadMaterialFromFile
**
** read in an endf6 style file for a single material
**
*/
int  endfReadMaterialFromFile( char *filename, endfMaterialInformation *mi  )
{
  FILE         *fENDF;
  endfLine      line;
  endfLine      oldline;
  int           i, file;


  /*
  ** open the endf file for reading
  */
  fENDF = fopen( filename, "r" );
  if( !fENDF ) {
    printf( "ERROR: endfReadMaterialFromFile:\n" );
    printf( "         cannot open file \"%s\" for reading\n", filename );
    return( -1 );
  }

  /*
  ** initialize the line
  */
  line.body[0] = '\0';
  line.material = 0;
  line.mf = 0;
  line.mt = 0;
  line.number = 0;
  line.file = fENDF;
  oldline = line;


  /*
  ** find the start of the material information
  */
  while( line.mf != 1 && line.mt != 451 ) {
    oldline = line;
    endfReadLine( &line );
  }


  /*
  ** copy the line previous to new material into evaluation title
  ** if no line before, file with null string
  */
  strcpy( mi->EvaluationTitle, oldline.body );

  /*
  ** read the data from the file
  */
  endfReadMF1MT451( &line, mi );
  endfReadRecords( &line, mi );

  /*
  ** close the file
  */
  close( fENDF );

  return( 0 );

}


/*******************************************************************
** endfPrintMaterialToFile
**
** print out an entire endf6 style file
**
** expects output file pointer is in 'line->body'
**
*/
int endfPrintMaterialToFile( char *filename, endfMaterialInformation *mi )
{
  int       intzero = 0;
  double    doublezero = 0.0;
  FILE     *fENDF;
  endfLine  line;

  /*
  ** open the endf file for writing
  */
  fENDF = fopen( filename, "w" );
  if( !fENDF ) {
    printf( "ERROR: endfPrintMaterialToFile:\n" );
    printf( "         cannot open file \"%s\" for writing\n", filename );
    return( -1 );
  }
```

```
/*
** initialize the line
*/
line.body[0] = '\0';
line.material = 0;
line.mf = 0;
line.mt = 0;
line.number = 0;
line.file = fENDF;

/*
** make sure the material is in endf 6 format
*/
if( mi->LibraryFormat != 6 ) {
  printf( "ERROR: endfPrintMaterialToFile:\n" );
  printf( "         ENDF-6 file format not set: mi->LibraryFormat is '%d'\n",
          mi->LibraryFormat );
  return( -2 );
}

/*
** if exists print the evaluation title line
*/
if( mi->EvaluationTitle[0] != '\0' ) {
  strcpy( line.body, mi->EvaluationTitle );
  line.material = 7777;
  line.mf = 0;
  line.mt = 0;
  line.number = 0;
  endfPrintLineToFile( line );
}

endfPrintMF1MT451( &line, mi );
endfPrintRecords( &line, mi );

/*
** create zeroes line for end markers
*/
endfNextLine( &line );
endfPutNumber( &line, 1, 2, (void*)&doublezero );
endfPutNumber( &line, 2, 2, (void*)&doublezero );
endfPutNumber( &line, 3, 1, (void*)&intzero );
endfPutNumber( &line, 4, 1, (void*)&intzero );
endfPutNumber( &line, 5, 1, (void*)&intzero );
endfPutNumber( &line, 6, 1, (void*)&intzero );

/*
** print end of last section marker
*/
line.mf = 0;
line.mt = 0;
endfPrintLineToFile( line );

/*
** print end of material marker
*/
line.number++;
line.material = 0;
endfPrintLineToFile( line );

/*
** print end of file marker
*/
line.number = 0;
line.material = -1;
endfPrintLineToFile( line );

/*
** close file
*/
```

```
  close( fENDF );


}
```

endfConvert.h

```
#ifndef endfConvert_h
#define endfConvert_h

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*
** Function: endfAtomicName
*/
void endfAtomicName( int z, char *name );

/*
** Function: endfAtomicSymbol
*/
void endfAtomicSymbol( int z, char *symbol );

/*
** Function: endfParticleType
*/
void endfParticleType( int IncidentParticleZA, char *IncidentParticleType );


#endif
```

endfConvert.c

```
#include "endfConvert.h"

/*******************************************************************
** endfAtomicName
**
** given an atomic number, return the element name
**
** double check that no names are longer than 16
** limit set in mf1mt451 structure definition
**
*/
void endfAtomicName( int z, char *name )
{
  char  *names[110] = { "", "Hydrogen", "Helium", "Lithium", "Beryllium", "Boron",
                  "Carbon", "Nitrogen", "Oxygen", "Fluorine", "Neon",
                  "Sodium", "Magnesium", "Aluminum", "Silicon", "Phosphorus",
                  "Sulfur", "Chlorine", "Argon", "Potassium", "Calcium",
                  "Scandium", "Titanium", "Vanadium", "Chromium", "Manganese",
                  "Iron", "Cobalt", "Nickel", "Copper", "Zinc",
                  "Gallium", "Germanium", "Arsenic", "Selenium", "Bromine",
                  "Krypton", "Rubidium", "Strontium", "Yttrium", "Zirconium",
                  "Niobium", "Molybdenum", "Technetium", "Ruthenium", "Rhodium",
                  "Palladium", "Silver", "Cadmium", "Indium", "Tin",
                  "Antimony", "Tellurium", "Iodine", "Xenon", "Cesium",
                  "Barium", "Lanthanum", "Cerium", "Praseodymium", "Neodymium",
                  "Promethium", "Samarium", "Europium", "Gadolinium", "Terbium",
                  "Dysprosium", "Holmium", "Erbium", "Thulium", "Ytterbium",
                  "Lutetium", "Hafnium", "Tantalum", "Tungsten", "Rhenium",
                  "Osmium", "Iridium", "Platinum", "Gold", "Mercury",
                  "Thallium", "Lead", "Bismuth", "Polonium", "Astatine",
                  "Radon", "Francium", "Radium", "Actinium", "Thorium",
                  "Protactinium", "Uranium", "Neptunium", "Plutonium", "Americium",
                  "Curium", "Berkelium", "Californium", "Einsteinium", "Fermium",
                  "Mendelevium", "Nobelium", "Lawrencium", "Rutherfordium", "Dubnium",
                 "Seaborgium", "Bohrium", "Hassium", "Meitnerium" };

  strncpy( name, names[z], strlen(names[z]) );
  name[strlen(names[z])] = '\0';

}


/******************************************************************
** endfAtomicSymbol
**
** given an atomic number, return the elemental symbol
**
*/
void endfAtomicSymbol( int z, char *symbol )
{
  char  *symbols[110] = { "", "H", "He", "Li", "Be", "B", "C", "N", "O", "F", "Ne",
                         "Na", "Mg", "Al", "Si", "P", "S", "Cl", "Ar", "K", "Ca",
                         "Sc", "Ti", "V", "Cr", "Mn", "Fe", "Co", "Ni", "Cu", "Zn",
                         "Ga", "Ge", "As", "Se", "Br", "Kr", "Rb", "Sr", "Y", "Zr",
                         "Nb", "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "In", "Sn",
                         "Sb", "Te", "I", "Xe", "Cs", "Ba", "La", "Ce", "Pr", "Nd",
                         "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm", "Yb",
                         "Lu", "Hf", "Ta", "W", "Re", "Os", "Ir", "Pt", "Au", "Hg",
                         "Tl", "Pb", "Bi", "Po", "At", "Rn", "Fr", "Ra", "Ac", "Th",
                         "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf", "Es", "Fm",
                         "Md", "No", "Lr", "Rf", "Db", "Sg", "Bh", "Hs", "Mt" };

  strncpy( symbol, symbols[z], strlen(symbols[z]) );
  symbol[strlen(symbols[z])] = '\0';

}

/******************************************************************
** endfParticleType
**
** given an incident particle awr, return the particle type
**
*/
void endfParticleType( int IncidentParticleZA, char *IncidentParticleType )
{
  int  i;
  const int  NumberOfTypes = 8;

  char  *type[8] = {
            "photon",
           "neutron",
           "electron",
           "proton",
           "deuteron",
           "triton",
           "helium-3",
           "alpha"
  };

  double za[8] = {
            0.0,
           1.0,
           1000.0,
           1001.0,
           1002.0,
           1003.0,
           2003.0,
           2004.0
  };


  for( i = 0; i < NumberOfTypes; i++ )
    if( za[i] == IncidentParticleZA )
      break;
  if( i == NumberOfTypes ) {
    printf( "ERROR: What did you shoot at me?!? a '%f'\n", IncidentParticleZA );
    exit( -1 );
  }

  strcpy( IncidentParticleType, type[i] );

}
```

endfLine.h

```
#ifndef endfLine_h
#define endfLine_h

#include <stdio.h>
#include <string.h>
#include <stdlib.h>


/*
** Structure: endfLine
*/
typedef struct endfline {
  char    body[67];
  int     material;
  int     mf;
  int     mt;
  int     number;
  FILE   *file;
} endfLine;


/*
** Function: endfReadLine
*/
int endfReadLine( endfLine *line );

/*
** Function: endfPrintLine
*/
void endfPrintLine( endfLine line );

/*
** Function: endfPrintLineToFile
*/
void endfPrintLineToFile( endfLine line );

/*
** Function: endfNextLine
*/
void endfNextLine( endfLine *line );


#endif
```

endfLine.c

```
#include "endfLine.h"

/*******************************************************************
** endfReadLine
**
** reads a line from an endf style file given in 'line.file'
**
** loads:
**    body <- record
**    mat <- materal number
**    mf <- file number
**    mt <- reaction number
**    number <- line number
**
** These errors are not yet implemented.
** thinking about having a strict/non-std option
** Fatal Error: if line length is not 80 characters
** Fatal Error: if any variable is unreadable
** Fatal Error: if mat, mf, mt, or number doesn't match expected
*/
int endfReadLine( endfLine *line )
{
  char  temp[133], t[10];
  int   i;
  static int   LineNumber = 0;


  LineNumber++;

  fgets( temp, 132, line->file );
  if( temp[80] != '\n' ) {
    printf( "ERROR: did not find proper 80 character line: line %d\n%s\n",
            LineNumber, temp );
    exit( -1 );
  }

  strncpy( line->body, &temp[0], 66 );
  line->body[66]='\0';

  strncpy( t, &temp[66], 4 );
  t[4]='\0';
  if( !sscanf( t, "%d", &line->material ) ) {
    printf( "ERROR: could not read mat number '%s' in line %d\n%s\n",
            t, LineNumber, temp );
    exit( -1 );
  }

  strncpy( t, &temp[70], 2 );
  t[2]='\0';
  if( !sscanf( t, "%d", &line->mf ) ) {
    printf( "ERROR: could not read mf number '%s' in line %d\n%s\n",
            t, LineNumber, temp );
    exit( -1 );
  }

  strncpy( t, &temp[72], 3 );
  t[3]='\0';
  if( !sscanf( t, "%d", &line->mt ) ) {
    printf( "ERROR: could not read mt number '%s' in line %d\n%s\n",
            t, LineNumber, temp );
    exit( -1 );
  }

  strncpy( t, &temp[75], 5 );
  t[5]='\0';
  if( !sscanf( t, "%d", &line->number ) ) {
    printf( "ERROR: could not read line number '%s' in line %d\n%s\n",
            t, LineNumber, temp );
    exit( -1 );
  }

  if( line->number == 149 ) {
    i = 0;
  }

}


/*********************************************************************
** endfPrintLine
**
** print the line in the endf format for use in id with original file
**
```

```
*/
void endfPrintLine( endfLine line )
{
  printf( "%66s%4d%2d%3d%5d\n", line.body, line.material,
          line.mf, line.mt, line.number );

}


/*****************************************************************
** endfPrintLineToFile
**
** print the line in the endf format to the file pointer 'line.file'
**
*/
void endfPrintLineToFile( endfLine line )
{
  fprintf( line.file, "%66s%4d%2d%3d%5d\n", line.body, line.material,
           line.mf, line.mt, line.number );

}


/*****************************************************************
** endfNextLine
**
** blank the 'line->body' and increment the 'line->number' by 1
**
*/
void endfNextLine( endfLine *line )
{

  static char l[] = "                                                                  ";

  strcpy( line->body, l );
  line->number++;

}
```

endfMF1MT451.h

```
#ifndef endfMF1MT451_h
#define endfMF1MT451_h

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "endfLine.h"
#include "endfNumber.h"
#include "endfConvert.h"

#include "endfMF1.h"
#include "endfMF2.h"
#include "endfMF3.h"
#include "endfMF4.h"
#include "endfMF5.h"
#include "endfMF6.h"


/*
** Structure: endfRecord
*/
typedef struct endfrecord {                   /* ENDF Parameter Name */
  int     MF;                             /* MFi */
  int     MT;                             /* MTi */
  int     NumberOfLines;                  /* NCi */
  int     ModificationNumber;             /* MODi */
  void   *MFMT;
} endfRecord;
```

```
/*
** Structure: endfMaterialInformation
*/
typedef struct endfmaterialinformation {  /* ENDF Parameter Name */
  char     EvaluationTitle[67];           /* Title line given on MF0 MT0 */
  double   TargetZA;                      /* ZA */
  char     TargetName[16];
  char     TargetSymbol[3];
  int      TargetIsotope;
  int      TargetAtomicNumber;
  double   TargetAWR;                     /* AWR */
  int      TargetResonancePointer;        /* LRP */
  int      TargetFissions;                /* LFI */
  int      LibraryIdentifier;             /* NLIB */
  int      MaterialModificationNumber;    /* NMOD */
  int      MaterialNumber;                /* MAT */
  double   TargetExcitationEnergy;        /* ELIS */
  double   TargetStability;               /* STA */
  int      TargetState;                   /* LIS */
  int      TargetIsomer;                  /* LISO */
  int      LibraryFormat;                 /* NFOR */
  double   IncidentParticleAWR;           /* AWI */
  char     IncidentParticleType[16];
  int      SublibraryNumber;              /* NSUB */
  int      IncidentParticleZA;
  int      EvaluationType;
  int      LibraryVersion;                /* NVER */
  double   Temperature;                   /* TEMP */
  int      DerivedLibrary;                /* LDRV */
  int      NumberOfCommentLines;          /* NWD */
  int      NumberOfRecords;               /* NXC */
  char     EvaluationLaboratory[12];
  char     EvaluationDate[6];
  char     EvaluationAuthors[34];
  char     EvaluationReference[23];
  char     EvaluationDistributionDate[6];
  char     EvaluationRevisionDate[6];
  char     EvaluationSource[19];
  char     EvaluationRevisionNumber[22];
  char     EvaluationMasterDate[6];
  char        **Comments;
  endfRecord  **Records;
} endfMaterialInformation;


/*
** Function: endfReadMF1MT451
*/
void endfReadMF1MT451( endfLine *line, endfMaterialInformation *mi );

/*
** Function: endfReadRecords
*/
void endfReadRecords( endfLine *line, endfMaterialInformation *mi );

/*
** Function: endfPrintMF1MT451
*/
void endfPrintMF1MT451( endfLine *line, endfMaterialInformation *mi );

/*
** Function: endfPrintRecords
*/
void endfPrintRecords( endfLine *line, endfMaterialInformation *mi );


#endif
```

endfMF1MT451.c

```
#include "endfMF1MT451.h"

/*******************************************************************
** endfReadMF1MT451
**
** loads the material information from the mf1 mt451 required section
** allocates records for holding the specified record entries
**
** Fatal Error: if 'LibraryFormat' is not 6 (ENDF-6 style)
**
*/
void endfReadMF1MT451( endfLine *line, endfMaterialInformation *mi )
{
  int i;

  mi->MaterialNumber = line->material;

  /* [ZA, AWR, LRP, LFI, NLIB, NMOD] */
  endfGetNumber( *line, 1, 2, &mi->TargetZA );
  endfGetNumber( *line, 2, 2, &mi->TargetAWR );
  endfGetNumber( *line, 3, 1, &mi->TargetResonancePointer );
  endfGetNumber( *line, 4, 1, &mi->TargetFissions );
  endfGetNumber( *line, 5, 1, &mi->LibraryIdentifier );
  endfGetNumber( *line, 6, 1, &mi->MaterialModificationNumber );

  /*
  ** Set some other values based on the ZA number
  ** When converting ZA to an int, add eps to avoid rounding error
  */
  mi->TargetIsotope = (int)(mi->TargetZA + 0.00001) % 1000;
  mi->TargetAtomicNumber = ( (int)(mi->TargetZA + 0.00001) - mi->TargetIsotope ) / 1000;
  endfAtomicName( mi->TargetAtomicNumber, mi->TargetName );
  endfAtomicSymbol( mi->TargetAtomicNumber, mi->TargetSymbol );


  endfReadLine( line );
  /* [ ELIS, STA, LIS, LISO, 0, NFOR ] */
  endfGetNumber( *line, 1, 2, &mi->TargetExcitationEnergy );
  endfGetNumber( *line, 2, 2, &mi->TargetStability );
  endfGetNumber( *line, 3, 1, &mi->TargetState );
  endfGetNumber( *line, 4, 1, &mi->TargetIsomer );
  endfGetNumber( *line, 6, 1, &mi->LibraryFormat );

  if( mi->LibraryFormat != 6 ) {
    printf( "ERROR: endfReadMF1MT451:\n" );
    printf( "         this is not an ENDF-6 file: read library format '%d'\n",
            mi->LibraryFormat );
    endfPrintLine( *line );
    exit( -1 );
  }


  endfReadLine( line );
  /* [ AWI, 0.0, 0, 0, NSUB, NVER ] */
  endfGetNumber( *line, 1, 2, &mi->IncidentParticleAWR );
  endfGetNumber( *line, 5, 1, &mi->SublibraryNumber );
  endfGetNumber( *line, 6, 1, &mi->LibraryVersion );

  mi->IncidentParticleZA = mi->SublibraryNumber / 10;
  mi->EvaluationType = mi->SublibraryNumber % 10;
  endfParticleType( mi->IncidentParticleZA, mi->IncidentParticleType );


  endfReadLine( line );
  /* [ TEMP, 0.0, LDRV, 0, NWD, NXC ] */
  endfGetNumber( *line, 1, 2, &mi->Temperature );
  endfGetNumber( *line, 3, 1, &mi->DerivedLibrary );
  endfGetNumber( *line, 5, 1, &mi->NumberOfCommentLines );
  endfGetNumber( *line, 6, 1, &mi->NumberOfRecords );


  /*
  ** Allocate space and read comment lines
  */
  mi->Comments = (char**)calloc( mi->NumberOfCommentLines, sizeof( char* ) );
  for( i = 0; i < mi->NumberOfCommentLines; i++ ) {
    mi->Comments[i] = (char*)calloc( 67, sizeof( char ) );
    endfReadLine( line );
    strncpy( mi->Comments[i], line->body, 66 );
    mi->Comments[i][66] = '\0';
  }

  strncpy( mi->EvaluationLaboratory, &mi->Comments[0][11], 11 );
  mi->EvaluationLaboratory[11] = '\0';

  strncpy( mi->EvaluationDate, &mi->Comments[0][27], 5 );
  mi->EvaluationDate[5] = '\0';

  strncpy( mi->EvaluationAuthors, &mi->Comments[0][33], 33 );
  mi->EvaluationAuthors[33] = '\0';

  strncpy( mi->EvaluationReference, &mi->Comments[1][0], 22 );
  mi->EvaluationReference[22] = '\0';

  strncpy( mi->EvaluationDistributionDate, &mi->Comments[1][27], 5 );
  mi->EvaluationDistributionDate[5] = '\0';

  strncpy( mi->EvaluationRevisionDate, &mi->Comments[1][38], 5 );
  mi->EvaluationRevisionDate[5] = '\0';

  strncpy( mi->EvaluationMasterDate, &mi->Comments[1][55], 5 );
  mi->EvaluationMasterDate[5] = '\0';

  strncpy( mi->EvaluationRevisionNumber, &mi->Comments[2][44], 22 );
  mi->EvaluationRevisionNumber[22] = '\0';


  /*
  ** Allocate space and read record cards
  */
  mi->Records = (endfRecord**)calloc( mi->NumberOfRecords, sizeof( endfRecord* ) );
  for( i = 0; i < mi->NumberOfRecords; i++ ) {
    mi->Records[i] = (endfRecord*)calloc( 1, sizeof( endfRecord ) );
    endfReadLine( line );
    endfGetNumber( *line, 3, 1, &mi->Records[i]->MF );
    endfGetNumber( *line, 4, 1, &mi->Records[i]->MT );
    endfGetNumber( *line, 5, 1, &mi->Records[i]->NumberOfLines );
    endfGetNumber( *line, 6, 1, &mi->Records[i]->ModificationNumber );

    switch( mi->Records[i]->MF ) {
    case 1:
      mi->Records[i]->MFMT = (endfMF1*)calloc( 1, sizeof( endfMF1 ) );
      break;
    case 2:
      mi->Records[i]->MFMT = (endfMF2*)calloc( 1, sizeof( endfMF2 ) );
      break;
    case 3:
      mi->Records[i]->MFMT = (endfMF3*)calloc( 1, sizeof( endfMF3 ) );
      break;
    case 4:
      mi->Records[i]->MFMT = (endfMF4*)calloc( 1, sizeof( endfMF4 ) );
      break;
    case 5:
      mi->Records[i]->MFMT = (endfMF5*)calloc( 1, sizeof( endfMF5 ) );
      break;
    case 6:
      mi->Records[i]->MFMT = (endfMF6*)calloc( 1, sizeof( endfMF6 ) );
      break;
    default:
      printf( "ERROR: Record description '%d' on line '%d' says mf '%d'\n",
              line->number, i+1, mi->Records[i]->MF );
      exit( -1 );
    }
  } /* end of record information and allocation */


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 1 && line->mt != 0 ) {
    printf( "ERROR: expected end of mf1 mt451 section: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*********************************************************************
** endfReadRecords
**
** loads all material record information into the
** appropriate record section
**
** expects to be handed 'line' with endf file open
** to first line past the end of the mf1 mt451 data
**
*/
void endfReadRecords( endfLine *line, endfMaterialInformation *mi )
{
  int  i;
  int  file = 1;  /* current mf = 1 */


  for( i = 1; i < mi->NumberOfRecords; i++ ) {
    /*
    ** skip over the end of endf file section marker
    */
    if( mi->Records[i]->MF != file ) {
      endfReadLine( line );
      if( line->mf != 0 && line->mt != 0 ) {
        printf( "ERROR: end of file %d not properly terminated\n", file );
        endfPrintLine( *line );
        exit( -1 );
      }
      file = mi->Records[i]->MF;
    }

    /*
    ** check for beginning of proper section
    */
    endfReadLine( line );
    if( line->mf != mi->Records[i]->MF && line->mt != mi->Records[i]->MT ) {
      printf( "ERROR: didn't find next expected record\n" );
      printf( "       got    mf '%2d' and mt '%3d'\n",
              line->mf, line->mt );
      printf( "       wanted mf '%2d' and mt '%3d'\n",
              mi->Records[i]->MF, mi->Records[i]->MT );
      endfPrintLine( *line );
      exit( -1 );
    }

    /*
    ** hand off section to proper reader
    */
    switch( line->mf ) {
    case 1:
      endfReadMF1( line, (endfMF1*)mi->Records[i]->MFMT );
      break;
    case 2:
      endfReadMF2( line, (endfMF2*)mi->Records[i]->MFMT );
      break;
    case 3:
      endfReadMF3( line, (endfMF3*)mi->Records[i]->MFMT );
      break;
    case 4:
      endfReadMF4( line, (endfMF4*)mi->Records[i]->MFMT );
      break;
    case 5:
      endfReadMF5( line, (endfMF5*)mi->Records[i]->MFMT );
      break;
    case 6:
      endfReadMF6( line, (endfMF6*)mi->Records[i]->MFMT );
      break;
    default:
      printf( "ERROR: how do I load an mf%d file\n", line->mf );
      endfPrintLine( *line );
      exit( -1 );
    }

  }

}

/*******************************************************************
** endfPrintMF1MT451
**
** prints the material information in the mf1 mt451 required section
**
** 'line->file' should be a passed writable file pointer
**
*/
void endfPrintMF1MT451( endfLine *line, endfMaterialInformation *mi )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;

  line->mf = 1;
  line->mt = 451;
  line->material = mi->MaterialNumber;
  line->number = 0;

  endfNextLine( line );
  /* [ZA, AWR, LRP, LFI, NLIB, NMOD] */
  endfPutNumber( line, 1, 2, (void*)&mi->TargetZA );
  endfPutNumber( line, 2, 2, (void*)&mi->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&mi->TargetResonancePointer );
  endfPutNumber( line, 4, 1, (void*)&mi->TargetFissions );
  endfPutNumber( line, 5, 1, (void*)&mi->LibraryIdentifier );
  endfPutNumber( line, 6, 1, (void*)&mi->MaterialModificationNumber );
  endfPrintLineToFile( *line );


  endfNextLine( line );
  /* [ ELIS, STA, LIS, LISO, 0, NFOR ] */
  endfPutNumber( line, 1, 2, (void*)&mi->TargetExcitationEnergy );
  endfPutNumber( line, 2, 2, (void*)&mi->TargetStability );
  endfPutNumber( line, 3, 1, (void*)&mi->TargetState );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&mi->TargetIsomer );
  endfPutNumber( line, 6, 1, (void*)&mi->LibraryFormat );
  endfPrintLineToFile( *line );


  endfNextLine( line );
  /* [ AWI, 0.0, 0, 0, NSUB, NVER ] */
  endfPutNumber( line, 1, 2, (void*)&mi->IncidentParticleAWR );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&mi->SublibraryNumber );
  endfPutNumber( line, 6, 1, (void*)&mi->LibraryVersion );
  endfPrintLineToFile( *line );


  endfNextLine( line );
  /* [ TEMP, 0.0, LDRV, 0, NWD, NXC ] */
  endfPutNumber( line, 1, 2, (void*)&mi->Temperature );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&mi->DerivedLibrary );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&mi->NumberOfCommentLines );
  endfPutNumber( line, 6, 1, (void*)&mi->NumberOfRecords );
  endfPrintLineToFile( *line );


  /*
  ** print comment lines
  */
  for( i = 0; i < mi->NumberOfCommentLines; i++ ) {
    endfNextLine( line );
    strcpy( line->body, mi->Comments[i] );
    endfPrintLineToFile( *line );
  }

  /*
  ** print record cards
  */
  for( i = 0; i < mi->NumberOfRecords; i++ ) {
    endfNextLine( line );
    endfPutNumber( line, 3, 1, &mi->Records[i]->MF );
    endfPutNumber( line, 4, 1, &mi->Records[i]->MT );
    endfPutNumber( line, 5, 1, &mi->Records[i]->NumberOfLines );
    endfPutNumber( line, 6, 1, &mi->Records[i]->ModificationNumber );
    endfPrintLineToFile( *line );
  }


  /*
  ** print end of mf1 mt451 marker
  */
  endfNextLine( line );
  /* [ 0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintRecords
**
** prints each of the material record cards
**
** 'line->file' should be a passed writable file pointer
** 'line->mf' should contain the starting endf file section number
**
*/
void endfPrintRecords( endfLine *line, endfMaterialInformation *mi )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;
  for( i = 1; i < mi->NumberOfRecords; i++ ) {
    /*
    ** print an end of endf file section marker on mf change
    */
    if( mi->Records[i]->MF != line->mf ) {
      endfNextLine( line );
      /* [ 0, 0.0, 0, 0, 0, 0 ] */
      line->mf = 0;
      line->mt = 0;
      endfPutNumber( line, 1, 2, (void*)&doublezero );
      endfPutNumber( line, 2, 2, (void*)&doublezero );
      endfPutNumber( line, 3, 1, (void*)&intzero );
      endfPutNumber( line, 4, 1, (void*)&intzero );
      endfPutNumber( line, 5, 1, (void*)&intzero );
      endfPutNumber( line, 6, 1, (void*)&intzero );
      endfPrintLineToFile( *line );
      line->mf = mi->Records[i]->MF;
    }

    /*
    ** hand off section to proper printer
    */
    line->mt = mi->Records[i]->MT;
    switch( line->mf ) {
    case 1:
      endfPrintMF1( line, mi->Records[i]->MFMT );
      break;
    case 2:
      line->number += mi->Records[i]->NumberOfLines;  /* REMOVE WHEN WRITE READER/WRITER
*/
      endfPrintMF2( line, mi->Records[i]->MFMT );
      break;
    case 3:
      endfPrintMF3( line, mi->Records[i]->MFMT );
      break;
    case 4:
      endfPrintMF4( line, mi->Records[i]->MFMT );
      break;
    case 5:
      endfPrintMF5( line, mi->Records[i]->MFMT );
      break;
    case 6:
      endfPrintMF6( line, mi->Records[i]->MFMT );
      break;
    default:
      printf( "ERROR: don't know how to print an mf%d file\n", line->mf );
      exit( -1 );
    }

  }

}
```

endfMF1.h

```
#ifndef endfMF1_h
#define endfMF1_h

#include "endfLine.h"


/*
** Structure: endfMF1T
*/
typedef struct endfmf1t {                   /* ENDF Parameter Name */
  int     NumberOfCoefficients;         /* NC */
  double *PolyCoefficients;             /* Ci */
  int     NumberOfInterpolationRegions; /* NR */
  int    *NumberOfPointsInRegion;
  int    *InterpolationSchemeInRegion;
  int     NumberOfPoints;               /* NP */
  double *Energy;                       /* Ei */
  double *Nu;                           /* Nu(Ei) */
} endfMF1T;

/*
** Structure: endfMF1P
*/
typedef struct endfmf1p {                    /* ENDF Parameter Name */
  int     NumberOfInterpolationRegions; /* NR */
  int    *NumberOfPointsInRegion;
  int    *InterpolationSchemeInRegion;
  int     NumberOfPoints;               /* NP */
  double *Energy;                       /* Ei */
  double *Nu;                           /* Nu(Ei) */
  double  SpontaneousNu;                /* Nu */
} endfMF1P;

/*
** Structure: endfMF1D
*/
typedef struct endfmf1d {                    /* ENDF Parameter Name */
  int     NumberOfPrecursors;           /* NNF */
  double *DecayConstants;               /* Lambda_i */
  int     NumberOfInterpolationRegions; /* NR */
  int    *NumberOfPointsInRegion;
  int    *InterpolationSchemeInRegion;
  int     NumberOfPoints;               /* NP */
  double *Energy;                       /* Ei */
  double *Nu;                           /* Nu(Ei) */
  double  SpontaneousNu;                /* Nu */
} endfMF1D;

/*
** Structure: endfMF1E
*/
typedef struct endfmf1e {         /* ENDF Parameter Name */
  double  TotalEnergy;            /* ET */
  double  erTotalEnergy;
  double  FragmentEnergy;         /* EFR */
  double  erFragmentEnergy;
  double  PromptNeutronEnergy;    /* ENP */
  double  erPromptNeutronEnergy;
  double  DelayedNeutronEnergy;   /* END */
  double  erDelayedNeutronEnergy;
  double  PromptGammaEnergy;      /* EGP */
  double  erPromptGammaEnergy;
  double  DelayedGammaEnergy;     /* EGD */
  double  erDelayedGammaEnergy;
  double  BetaEnergy;             /* EB */
  double  erBetaEnergy;
  double  NeutrinoEnergy;         /* ENU */
  double  erNeutrinoEnergy;
  double  SeenEnergy;             /* ER: total - neutrino */
  double  erSeenEnergy;
} endfMF1E;

/*
** Structure: endfMF1
*/
typedef struct endfmf1 {                    /* ENDF Parameter Name */
  double  TargetZA;                    /* ZA */
  double  TargetAWR;                   /* AWR */
  int     NuRepresentation;            /* LNU */
  int     NuType;
  void   *NuParameters;
} endfMF1;

/*
```

```
** Function: endfReadMF1
*/
void endfReadMF1( endfLine *line, endfMF1 *data );

/*
** Function: endfReadMF1E
*/
void endfReadMF1E( endfLine *line, endfMF1 *data );

/*
** Function: endfReadMF1T
*/
void endfReadMF1T( endfLine *line, endfMF1 *data );

/*
** Function: endfReadMF1P
*/
void endfReadMF1P( endfLine *line, endfMF1 *data );

/*
** Function: endfReadMF1D
*/
void endfReadMF1D( endfLine *line, endfMF1 *data );

/*
** Function: endfPrintMF1
*/
void endfPrintMF1( endfLine *line, endfMF1 *data );

/*
** Function: endfPrintMF1E
*/
void endfPrintMF1E( endfLine *line, endfMF1 *data );

/*
** Function: endfPrintMF1T
*/
void endfPrintMF1T( endfLine *line, endfMF1 *data );

/*
** Function: endfPrintMF1P
*/
void endfPrintMF1P( endfLine *line, endfMF1 *data );

/*
** Function: endfPrintMF1D
*/
void endfPrintMF1D( endfLine *line, endfMF1 *data );


#endif
```

endfMF1.c

```
#include "endfMF1.h"

/*******************************************************************
** endfReadMF1
**
** read the nubar data
**
** 'NuType' is duplication of mt for ease of use
**   -1 is the energy distribution amoung the particles
**    1 is Total
**    2 is Prompt
**    3 is Delayed
**
*/
void endfReadMF1( endfLine *line, endfMF1 *data )
{
  if( line->mt == 452 )
    data->NuType = 1;
  else if( line->mt == 456 )
    data->NuType = 2;
  else if( line->mt == 455 )
    data->NuType = 3;
  else if( line->mt == 458 )
    data->NuType = -1;
  else {
    printf( "ERROR: What the heck is mt '%d'\n", line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

  /* [ ZA, AWR, 0, LNU, 0, 0 ] */
  endfGetNumber( *line, 1, 2, &data->TargetZA );
  endfGetNumber( *line, 2, 2, &data->TargetAWR );
  endfGetNumber( *line, 4, 1, &data->NuRepresentation );

  switch( data->NuType ) {
  case -1:
    data->NuParameters = (endfMF1E*)calloc( 1, sizeof( endfMF1E ) );
    endfReadMF1E( line, data );
    break;
  case 1:
    data->NuParameters = (endfMF1T*)calloc( 1, sizeof( endfMF1T ) );
    endfReadMF1T( line, data );
    break;
  case 2:
    data->NuParameters = (endfMF1P*)calloc( 1, sizeof( endfMF1P ) );
    endfReadMF1P( line, data );
    break;
  case 3:
    data->NuParameters = (endfMF1D*)calloc( 1, sizeof( endfMF1D ) );
    endfReadMF1D( line, data );
    break;
  }

}


/*******************************************************************
** endfReadMF1E
**
** read the components of energy release due to fission
**
*/
void endfReadMF1E( endfLine *line, endfMF1 *data )
{
  int    i;
  endfMF1E  *par = (endfMF1E*)data->NuParameters;


  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, 18, 9 ] */
  endfGetNumber( *line, 5, 1, &i );
  if( i != 18 ) {
    printf( "ERROR: Energy release number of entries expects 18; found '%d'\n", i );
    endfPrintLine( *line );
    exit( -1 );
  }
  endfGetNumber( *line, 6, 1, &i );
  if( i != 9 ) {
    printf( "ERROR: Energy release number of subparts expects 9; found '%d'\n", i );
    endfPrintLine( *line );
    exit( -1 );
  }


  endfReadLine( line );
  /* [ EFR, eEFR, ENP, eENP, END, eEND ] */
  endfGetNumber( *line, 1, 2, &par->FragmentEnergy );
  endfGetNumber( *line, 2, 2, &par->erFragmentEnergy );
  endfGetNumber( *line, 3, 2, &par->PromptNeutronEnergy );
  endfGetNumber( *line, 4, 2, &par->erPromptNeutronEnergy );
  endfGetNumber( *line, 5, 2, &par->DelayedNeutronEnergy );
  endfGetNumber( *line, 6, 2, &par->erDelayedNeutronEnergy );

  endfReadLine( line );
  /* [ EGP, eEGP, EGD, eEGD, EB, eEB ] */
  endfGetNumber( *line, 1, 2, &par->PromptGammaEnergy );
  endfGetNumber( *line, 2, 2, &par->erPromptGammaEnergy );
  endfGetNumber( *line, 3, 2, &par->DelayedGammaEnergy );
  endfGetNumber( *line, 4, 2, &par->erDelayedGammaEnergy );
  endfGetNumber( *line, 5, 2, &par->BetaEnergy );
  endfGetNumber( *line, 6, 2, &par->erBetaEnergy );

  endfReadLine( line );
  /* [ ENU, eENU, ER, eER, ET, eET ] */
  endfGetNumber( *line, 1, 2, &par->NeutrinoEnergy );
  endfGetNumber( *line, 2, 2, &par->erNeutrinoEnergy );
  endfGetNumber( *line, 3, 2, &par->SeenEnergy );
  endfGetNumber( *line, 4, 2, &par->erSeenEnergy );
  endfGetNumber( *line, 5, 2, &par->TotalEnergy );
  endfGetNumber( *line, 6, 2, &par->erTotalEnergy );


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 1 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 1 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfReadMF1T
**
** read the total nubar parameters
**
*/
void endfReadMF1T( endfLine *line, endfMF1 *data )
{
  int    i;
  endfMF1T  *par = (endfMF1T*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {
    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, NC, 0 ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfCoefficients );

    par->PolyCoefficients = (double*)calloc( par->NumberOfCoefficients,
                                             sizeof( double ) );

    for( i = 0; i < par->NumberOfCoefficients; i++ ) {
      if( i % 6 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (i%6)+1, 2, &par->PolyCoefficients[i] );
    }
  }
  else {
    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfInterpolationRegions );
    endfGetNumber( *line, 6, 1, &par->NumberOfPoints );
    par->NumberOfPointsInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );
    par->InterpolationSchemeInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );

    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   1, &par->NumberOfPointsInRegion[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 1, &par->InterpolationSchemeInRegion[i] );
    }

    par->Energy
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );
    par->Nu
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );

    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   2, &par->Energy[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 2, &par->Nu[i] );
    }
  }


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 1 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 1 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfReadMF1P
**
** read the prompt nubar parameters
**
*/
void endfReadMF1P( endfLine *line, endfMF1 *data )
{
  int    i;
  endfMF1P  *par = (endfMF1P*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {

    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, 1, 0 ] */
    endfGetNumber( *line, 5, 1, &i );

    if( i != 1 ) {
      printf( "ERROR: Prompt nubar represetation (LNU=1) expects 1; found '%d'\n", i );
      endfPrintLine( *line );
      exit( -1 );
    }

    endfReadLine( line );
    endfGetNumber( *line, 1, 2, &par->SpontaneousNu );
  }
  else {
    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfInterpolationRegions );
    endfGetNumber( *line, 6, 1, &par->NumberOfPoints );

    par->NumberOfPointsInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );
    par->InterpolationSchemeInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );

    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   1, &par->NumberOfPointsInRegion[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 1, &par->InterpolationSchemeInRegion[i] );
    }

    par->Energy
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );
    par->Nu
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );

    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   2, &par->Energy[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 2, &par->Nu[i] );
    }
  }


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 1 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 1 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfReadMF1D
**
** read the delayed nubar parameters
**
*/
void endfReadMF1D( endfLine *line, endfMF1 *data )
{
  int    i;
  endfMF1D  *par = (endfMF1D*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {

    endfReadLine( line );
    /* [0.0, 0.0, 0, 0, NNF, 0 ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfPrecursors );

    par->DecayConstants = (double*)calloc( par->NumberOfPrecursors, sizeof( double ) );
    for( i = 0; i < par->NumberOfPrecursors; i++ ) {
      if( i % 6 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (i%6)+1, 2, &par->DecayConstants[i] );
    }

    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, 1, 0 ] */
    endfGetNumber( *line, 5, 1, &i );
    if( i != 1 ) {
      printf( "ERROR: Prompt nubar represetation (LNU=1) expects 1; found '%d'\n", i );
      endfPrintLine( *line );
      exit( -1 );
    }

    endfReadLine( line );
    endfGetNumber( *line, 1, 2, &par->SpontaneousNu );
  }
  else {
    endfReadLine( line );
    /* [0.0, 0.0, 0, 0, NNF, 0 ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfPrecursors );

    par->DecayConstants = (double*)calloc( par->NumberOfPrecursors, sizeof( double ) );
    for( i = 0; i < par->NumberOfPrecursors; i++ ) {
      if( i % 6 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (i%6)+1, 2, &par->DecayConstants[i] );
    }

    endfReadLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfGetNumber( *line, 5, 1, &par->NumberOfInterpolationRegions );
    endfGetNumber( *line, 6, 1, &par->NumberOfPoints );

    par->NumberOfPointsInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );
    par->InterpolationSchemeInRegion
      = (int*)calloc( par->NumberOfInterpolationRegions, sizeof( int ) );

    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   1, &par->NumberOfPointsInRegion[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 1, &par->InterpolationSchemeInRegion[i] );
    }

    par->Energy
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );
    par->Nu
      = (double*)calloc( par->NumberOfPoints, sizeof( double ) );

    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6,   2, &par->Energy[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 2, &par->Nu[i] );
    }
  }


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 1 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 1 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfPrintMF1
**
** print the nubar data
```

```
**
*/
void endfPrintMF1( endfLine *line, endfMF1 *data )
{
  int     intzero = 0;
  double  doublezero = 0.0;


    endfNextLine( line );
    /* [ ZA, AWR, 0, LNU, 0, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&data->TargetZA );
    endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&data->NuRepresentation );
    endfPutNumber( line, 5, 1, (void*)&intzero );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );

    switch( data->NuType ) {
    case -1:
      endfPrintMF1E( line, data );
      break;
    case 1:
      endfPrintMF1T( line, data );
      break;
    case 2:
      endfPrintMF1P( line, data );
      break;
    case 3:
      endfPrintMF1D( line, data );
      break;
    default:
      printf( "ERROR: how did I get here? in PrintMF1 with weird NuType '%d'\n",
              data->NuType );
      exit( -1 );
    }


  /*
  ** print end of record marker
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}


/*****************************************************************
** endfPrintMF1E
**
** print the components of energy release due to fission
**
*/
void endfPrintMF1E( endfLine *line, endfMF1 *data )
{
  int    i;
  int     intzero = 0;
  double  doublezero = 0.0;
  endfMF1E  *par = (endfMF1E*)data->NuParameters;


  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 18, 9 ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  i = 18;
  endfPutNumber( line, 5, 1, (void*)&i );
  i = 9;
  endfPutNumber( line, 6, 1, (void*)&i );
  endfPrintLineToFile( *line );

  endfNextLine( line );
  /* [ EFR, eEFR, ENP, eENP, END, eEND ] */
  endfPutNumber( line, 1, 2, (void*)&par->FragmentEnergy );
  endfPutNumber( line, 2, 2, (void*)&par->erFragmentEnergy );
  endfPutNumber( line, 3, 2, (void*)&par->PromptNeutronEnergy );
  endfPutNumber( line, 4, 2, (void*)&par->erPromptNeutronEnergy );
  endfPutNumber( line, 5, 2, (void*)&par->DelayedNeutronEnergy );
  endfPutNumber( line, 6, 2, (void*)&par->erDelayedNeutronEnergy );
  endfPrintLineToFile( *line );

  endfNextLine( line );
  /* [ EGP, eEGP, EGD, eEGD, EB, eEB ] */
  endfPutNumber( line, 1, 2, (void*)&par->PromptGammaEnergy );
  endfPutNumber( line, 2, 2, (void*)&par->erPromptGammaEnergy );
  endfPutNumber( line, 3, 2, (void*)&par->DelayedGammaEnergy );
  endfPutNumber( line, 4, 2, (void*)&par->erDelayedGammaEnergy );
  endfPutNumber( line, 5, 2, (void*)&par->BetaEnergy );
  endfPutNumber( line, 6, 2, (void*)&par->erBetaEnergy );
  endfPrintLineToFile( *line );

  endfNextLine( line );
  /* [ ENU, eENU, ER, eER, ET, eET ] */
  endfPutNumber( line, 1, 2, (void*)&par->NeutrinoEnergy );
  endfPutNumber( line, 2, 2, (void*)&par->erNeutrinoEnergy );
  endfPutNumber( line, 3, 2, (void*)&par->SeenEnergy );
  endfPutNumber( line, 4, 2, (void*)&par->erSeenEnergy );
  endfPutNumber( line, 5, 2, (void*)&par->TotalEnergy );
  endfPutNumber( line, 6, 2, (void*)&par->erTotalEnergy );
  endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintMF1T
**
** print the total nubar parameters
**
*/
void endfPrintMF1T( endfLine *line, endfMF1 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;
  endfMF1T   *par = (endfMF1T*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NC, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfCoefficients );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfCoefficients; i++ ) {
      if( i % 6 == 0 )
```

```
        endfNextLine( line );
      endfPutNumber( line, (i%6)+1, 2, (void*)&par->PolyCoefficients[i] );
      if( i % 6 == 5 )
        endfPrintLineToFile( *line );
    }
    if( i % 6 != 0 )
      endfPrintLineToFile( *line );
  }
  else if( data->NuRepresentation == 2 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfInterpolationRegions );
    endfPutNumber( line, 6, 1, (void*)&par->NumberOfPoints );
    endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   1, (void*)&par->NumberOfPointsInRegion[i] );
      endfPutNumber( line, (1+2*i)%6+1, 1, (void*)&par->InterpolationSchemeInRegion[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   2, (void*)&par->Energy[i] );
      endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&par->Nu[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );
  }
  else {
    printf( "ERROR: Nu representation '%d' was not recognized\n",
            data->NuRepresentation );
    exit( -1 );
  }

}


/*********************************************************************
** endfPrintMF1P
**
** print the prompt nubar parameters
**
*/
void endfPrintMF1P( endfLine *line, endfMF1 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;
  endfMF1P   *par = (endfMF1P*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, 1, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    i = 1;
    endfPutNumber( line, 5, 1, (void*)&i );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );


    endfNextLine( line );
    endfPutNumber( line, 1, 2, (void*)&par->SpontaneousNu );
    endfPrintLineToFile( *line );
  }
  else if( data->NuRepresentation == 2 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfInterpolationRegions );
    endfPutNumber( line, 6, 1, (void*)&par->NumberOfPoints );
    endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   1, (void*)&par->NumberOfPointsInRegion[i] );
      endfPutNumber( line, (1+2*i)%6+1, 1, (void*)&par->InterpolationSchemeInRegion[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   2, (void*)&par->Energy[i] );
      endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&par->Nu[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );
  }
  else {
    printf( "ERROR: Nu representation '%d' was not recognized\n",
            data->NuRepresentation );
    exit( -1 );
  }

}


/*********************************************************************
** endfPrintMF1D
**
** print the delayed nubar parameters
**
*/
void endfPrintMF1D( endfLine *line, endfMF1 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;
  endfMF1D   *par = (endfMF1D*)data->NuParameters;


  if( data->NuRepresentation == 1 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NNF, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfPrecursors );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfPrecursors; i++ ) {
      if( i % 6 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (i%6)+1, 2, (void*)&par->DecayConstants[i] );
      if( i % 6 == 5 )
        endfPrintLineToFile( *line );
    }
    if( i % 6 != 0 )
      endfPrintLineToFile( *line );


    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, 1, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    i = 1;
    endfPutNumber( line, 5, 1, (void*)&i );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );


    endfNextLine( line );
    endfPutNumber( line, 1, 2, (void*)&par->SpontaneousNu );
    endfPrintLineToFile( *line );
  }
  else if( data->NuRepresentation == 2 ) {
    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NNF, 0 ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfPrecursors );
    endfPutNumber( line, 6, 1, (void*)&intzero );
    endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfPrecursors; i++ ) {
      if( i % 6 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (i%6)+1, 2, (void*)&par->DecayConstants[i] );
      if( i % 6 == 5 )
        endfPrintLineToFile( *line );
    }
    if( i % 6 != 0 )
      endfPrintLineToFile( *line );


    endfNextLine( line );
    /* [ 0.0, 0.0, 0, 0, NR, NP ] */
    endfPutNumber( line, 1, 2, (void*)&doublezero );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&intzero );
    endfPutNumber( line, 5, 1, (void*)&par->NumberOfInterpolationRegions );
    endfPutNumber( line, 6, 1, (void*)&par->NumberOfPoints );
    endfPrintLineToFile( *line );
    for( i = 0; i < par->NumberOfInterpolationRegions; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   1, (void*)&par->NumberOfPointsInRegion[i] );
      endfPutNumber( line, (1+2*i)%6+1, 1, (void*)&par->InterpolationSchemeInRegion[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );


    for( i = 0; i < par->NumberOfPoints; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   2, (void*)&par->Energy[i] );
      endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&par->Nu[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );
  }
  else {
    printf( "ERROR: Nu representation '%d' was not recognized\n",
            data->NuRepresentation );
    exit( -1 );
  }

}
```

endfMF2.h

```
#ifndef endfMF2_h
#define endfMF2_h

#include "endfLine.h"

/*
** Structure: endfMF2
*/
typedef struct endfmf2 {
  int     hello;
} endfMF2;

/*
** Function: endfReadMF2
*/
void endfReadMF2( endfLine *line, endfMF2 *data );

/*
** Function: endfPrintMF2
*/
void endfPrintMF2( endfLine *line, endfMF2 *data );


#endif
```

endfMF2.c

```
#include "endfMF2.h"

/*******************************************************************
** endfReadMF2
**
** What to do with the file 2???? blahh
**
*/
```

```
void endfReadMF2( endfLine *line, endfMF2 *data )
{
  int  i=1;

  while( i ) {
    endfReadLine( line );
    if( line->mt == 0 )
      i = 0;
  }

  printf( "write the mf2 reader sections\n" );
}

/*******************************************************************
** endfPrintMF2
**
** Still need to write the mf2 printer too
**
*/
void endfPrintMF2( endfLine *line, endfMF2 *data )
{
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}
```

endfMF3.h

```
#ifndef endfMF3_h
#define endfMF3_h

#include <stdio.h>
#include <stdlib.h>

#include "endfLine.h"
#include "endfNumber.h"


/*
** Structure: endfMF3
*/
typedef struct endfmf3 {
  double  TargetZA;
  double  TargetAWR;
  double  ReactionQM;
  double  ReactionQI;
  int     BreakupFlag;
  int     NumberOfInterpolationRegions;
  int     NumberOfPoints;
  int    *NumberOfPointsInRegion;
  int    *InterpolationSchemeInRegion;
  double *Energy;
  double *CrossSection;
} endfMF3;


/*
** Function: endfReadMF3
```

```
*/
void endfReadMF3( endfLine *line, endfMF3 *data );

/*
** Function: endfPrintMF3
*/
void endfPrintMF3( endfLine *line, endfMF3 *data );


#endif
```

endfMF3.c

```
#include "endfMF3.h"

/*****************************************************************
** endfReadMF3
**
** passed first line of mf3 section, load the remaining data
**
*/
void endfReadMF3( endfLine *line, endfMF3 *data )
{
  int i;

  /* [ ZA, AWR, 0, 0, 0, 0 ] */
  endfGetNumber( *line, 1, 2, &data->TargetZA );
  endfGetNumber( *line, 2, 2, &data->TargetAWR );

  endfReadLine( line );
  /* [ QM, QI, 0, LR, NR, NP ] */
  endfGetNumber( *line, 1, 2, &data->ReactionQM );
  endfGetNumber( *line, 2, 2, &data->ReactionQI );
  endfGetNumber( *line, 4, 1, &data->BreakupFlag );
  endfGetNumber( *line, 5, 1, &data->NumberOfInterpolationRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfPoints );


  data->NumberOfPointsInRegion
    = (int*)calloc( data->NumberOfInterpolationRegions, sizeof( int ) );
  data->InterpolationSchemeInRegion
    = (int*)calloc( data->NumberOfInterpolationRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfInterpolationRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1, &data->NumberOfPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1, &data->InterpolationSchemeInRegion[i] );
  }

  data->Energy
    = (double*)calloc( data->NumberOfPoints, sizeof( double ) );
  data->CrossSection
    = (double*)calloc( data->NumberOfPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->Energy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->CrossSection[i] );
  }

  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 3 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 3 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfPrintMF3
**
** print a mf3 section
**
** expects 'line->body' to point to a writable file pointer
** expects 'line->mf' to be set to mf3
** expects 'line->mt' to be set to current mt
**
*/
void endfPrintMF3( endfLine *line, endfMF3 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ ZA, AWR, 0, 0, 0, 0 ] */
  endfPutNumber( line, 1, 2, (void*)&data->TargetZA );
  endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );


  endfNextLine( line );
  /* [ QM, QI, 0, LR, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&data->ReactionQM );
  endfPutNumber( line, 2, 2, (void*)&data->ReactionQI );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&data->BreakupFlag );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfInterpolationRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfInterpolationRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1, (void*)&data->NumberOfPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1, (void*)&data->InterpolationSchemeInRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->Energy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->CrossSection[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  /*
  ** print end of record marker
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}
```

endfMF4.h

```
#ifndef endfMF4_h
#define endfMF4_h

#include <stdio.h>
#include <stdlib.h>

#include "endfLine.h"
#include "endfNumber.h"


/*
** Structure: endfMF4
*/
typedef struct endfmf4 {
  double  TargetZA;
  double  TargetAWR;
  int     TransformationMatrix;
  int     AngularRepresentation;
  int     AllIsotropic;
  int     FrameOfReference;
} endfMF4;


/*
** Function: endfReadMF4
*/
void endfReadMF4( endfLine *line, endfMF4 *data );

/*
** Function: endfPrintMF4
*/
void endfPrintMF4( endfLine *line, endfMF4 *data );


#endif
```

endfMF4.c

```
#include "endfMF4.h"

/*******************************************************************
** endfReadMF4
**
** passed first line of mf4 section, load the remaining data
**
*/
void endfReadMF4( endfLine *line, endfMF4 *data )
{
  double temp;

  /* [ ZA, AWR, LVT, LTT, 0, 0 ] */
  endfGetNumber( *line, 1, 2, &data->TargetZA );
  endfGetNumber( *line, 2, 2, &data->TargetAWR );
  endfGetNumber( *line, 3, 1, &data->TransformationMatrix );
  endfGetNumber( *line, 4, 1, &data->AngularRepresentation );

  endfReadLine( line );
  /* [ 0.0, AWR, LI, LCT, 0, 0 ] */
  endfGetNumber( *line, 2, 2, &temp );
  endfGetNumber( *line, 3, 1, &data->AllIsotropic );
  endfGetNumber( *line, 4, 1, &data->FrameOfReference );

  if( data->AllIsotropic != 1 ) {
    printf( "ERROR: File 4 is not all isotropic: HELP!! (%d)\n", data->AllIsotropic );
    endfPrintLine( *line );
    exit( -1 );
  }

  if( temp != data->TargetAWR ) {
    printf( "ERROR: File 4 AWR's don't match; 1'%d' 2'%d'\n", temp, data->TargetAWR );
    endfPrintLine( *line );
    exit( -1 );
  }

  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 4 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 4 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/**********************************************************************
** endfPrintMF4
**
** print an mf4 section record
**
** expects 'line->body' to point to a writable file pointer
** expects 'line->mf' to be set to mf4
** expects 'line->mt' to be set to current mt
**
*/
void endfPrintMF4( endfLine *line, endfMF4 *data )
{
  int     intzero = 0;
  double  doublezero = 0.0;

  endfNextLine( line );
  /* [ ZA, AWR, LVT, LTT, 0, 0 ] */
  endfPutNumber( line, 1, 2, (void*)&data->TargetZA );
  endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&data->TransformationMatrix );
  endfPutNumber( line, 4, 1, (void*)&data->AngularRepresentation );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );


  endfNextLine( line );
  /* [ 0.0, AWR, LI, LCT, 0, 0 ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&data->AllIsotropic );
  endfPutNumber( line, 4, 1, (void*)&data->FrameOfReference );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );
  /*
  ** print end of record marker
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}
```

endfMF5.h

```
#ifndef endfMF5_h
#define endfMF5_h

#include <stdio.h>
#include <stdlib.h>

#include "endfLine.h"
#include "endfNumber.h"


/*
** Structure: endfMF5LF1
*/
typedef struct endfmf5lf1 {
  int     NumberOfThetaRegions;
  int     NumberOfThetaPoints;
  int    *NumberOfThetaPointsInRegion;
  int    *InterpolationSchemeInThetaRegion;
  double *ThetaEnergy;
  double *Theta;
} endfMF5LF1;

/*
** Structure: endfMF5LF5
*/
typedef struct endfmf5lf5 {
  int     NumberOfThetaRegions;
  int     NumberOfThetaPoints;
  int    *NumberOfThetaPointsInRegion;
  int    *InterpolationSchemeInThetaRegion;
  double *ThetaEnergy;
  double *Theta;
  int     NumberOfGRegions;
  int     NumberOfGPoints;
  int    *NumberOfGPointsInRegion;
  int    *InterpolationSchemeInGRegion;
  double *GEnergy;
  double *G;
} endfMF5LF5;

/*
** Structure: endfMF5LF7
*/
typedef struct endfmf5lf7 {
  int     NumberOfThetaRegions;
  int     NumberOfThetaPoints;
  int    *NumberOfThetaPointsInRegion;
  int    *InterpolationSchemeInThetaRegion;
  double *ThetaEnergy;
  double *Theta;
} endfMF5LF7;
```

```
/*
** Structure: endfMF5LF9
*/
typedef struct endfmf5lf9 {
  int     NumberOfThetaRegions;
  int     NumberOfThetaPoints;
  int    *NumberOfThetaPointsInRegion;
  int    *InterpolationSchemeInThetaRegion;
  double *ThetaEnergy;
  double *Theta;
} endfMF5LF9;

/*
** Structure: endfMF5LF11
*/
typedef struct endfmf5lf11 {
  int     NumberOfARegions;
  int     NumberOfAPoints;
  int    *NumberOfAPointsInRegion;
  int    *InterpolationSchemeInARegion;
  double *AEnergy;
  double *A;
  int     NumberOfBRegions;
  int     NumberOfBPoints;
  int    *NumberOfBPointsInRegion;
  int    *InterpolationSchemeInBRegion;
  double *BEnergy;
  double *B;
} endfMF5LF11;

/*
** Structure: endfMF5LF12
*/
typedef struct endfmf5lf12 {
  int     NumberOfMTRegions;
  int     NumberOfMTPoints;
  int    *NumberOfMTPointsInRegion;
  int    *InterpolationSchemeInMTRegion;
  double *MTEnergy;
  double *MaximumTemperature;
} endfMF5LF12;

/*
** Structure: endfMF5Distribution
*/
typedef struct endfmf5distribution {
  double  UpperEnergyDelta;
  int     EnergyDistributionLaw;
  int     NumberOfEnergyRegions;
  int     NumberOfEnergyPoints;
  int    *NumberOfEnergyPointsInRegion;
  int    *InterpolationSchemeInEnergyRegion;
  double *Energy;
  double *EnergyProbability;
  void   *Parameters;
} endfMF5Distribution;

/*
** Structure: endfMF5
*/
typedef struct endfmf5 {
  double  TargetZA;
  double  TargetAWR;
  int     NumberOfPartialEnergyDistributions;
  endfMF5Distribution  *Distributions;
} endfMF5;


/*
** Function: endfReadMF5
```

```
*/
void endfReadMF5( endfLine *line, endfMF5 *data );

/*
** Function: endfReadMF5LF1
*/
void endfReadMF5LF1( endfLine *line, endfMF5LF1 *data );

/*
** Function: endfReadMF5LF5
*/
void endfReadMF5LF5( endfLine *line, endfMF5LF5 *data );

/*
** Function: endfReadMF5LF7
*/
void endfReadMF5LF7( endfLine *line, endfMF5LF7 *data );

/*
** Function: endfReadMF5LF9
*/
void endfReadMF5LF9( endfLine *line, endfMF5LF9 *data );

/*
** Function: endfReadMF5LF11
*/
void endfReadMF5LF11( endfLine *line, endfMF5LF11 *data );

/*
** Function: endfReadMF5LF12
*/
void endfReadMF5LF12( endfLine *line, endfMF5LF12 *data );


/*
** Function: endfPrintMF5
*/
void endfPrintMF5( endfLine *line, endfMF5 *data );

/*
** Function: endfPrintMF5LF1
*/
void endfPrintMF5LF1( endfLine *line, endfMF5LF1 *data );

/*
** Function: endfPrintMF5LF5
*/
void endfPrintMF5LF5( endfLine *line, endfMF5LF5 *data );

/*
** Function: endfPrintMF5LF7
*/
void endfPrintMF5LF7( endfLine *line, endfMF5LF7 *data );

/*
** Function: endfPrintMF5LF9
*/
void endfPrintMF5LF9( endfLine *line, endfMF5LF9 *data );

/*
** Function: endfPrintMF5LF11
*/
void endfPrintMF5LF11( endfLine *line, endfMF5LF11 *data );

/*
** Function: endfPrintMF5LF12
*/
void endfPrintMF5LF12( endfLine *line, endfMF5LF12 *data );


#endif
```

endfMF5.c

```
#include "endfMF5.h"

/*****************************************************************
** endfReadMF5
**
** passed first line of mf5 section, load the remaining data
**
*/
void endfReadMF5( endfLine *line, endfMF5 *data )
{
  int i,j;

  /* [ ZA, AWR, 0, 0, NK, 0 ] */
  endfGetNumber( *line, 1, 2, &data->TargetZA );
  endfGetNumber( *line, 2, 2, &data->TargetAWR );
  endfGetNumber( *line, 5, 1, &data->NumberOfPartialEnergyDistributions );

  data->Distributions = (endfMF5Distribution*)
    calloc( data->NumberOfPartialEnergyDistributions, sizeof( endfMF5Distribution ) );

  for( j = 0; j < data->NumberOfPartialEnergyDistributions; j++ ) {
    endfReadLine( line );
    /* [ U, 0,0, 0, LF, NR, NP ] */
    endfGetNumber( *line, 1, 2, &data->Distributions[j].UpperEnergyDelta );
    endfGetNumber( *line, 4, 1, &data->Distributions[j].EnergyDistributionLaw );
    endfGetNumber( *line, 5, 1, &data->Distributions[j].NumberOfEnergyRegions );
    endfGetNumber( *line, 6, 1, &data->Distributions[j].NumberOfEnergyPoints );

    data->Distributions[j].NumberOfEnergyPointsInRegion
      = (int*)calloc( data->Distributions[j].NumberOfEnergyRegions,
                      sizeof( int ) );
    data->Distributions[j].InterpolationSchemeInEnergyRegion
      = (int*)calloc( data->Distributions[j].NumberOfEnergyRegions,
                      sizeof( int ) );

    for( i = 0; i < data->Distributions[j].NumberOfEnergyRegions; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6, 1,
                &data->Distributions[j].NumberOfEnergyPointsInRegion[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 1,
                &data->Distributions[j].InterpolationSchemeInEnergyRegion[i] );
    }

    data->Distributions[j].Energy
      = (double*)calloc( data->Distributions[j].NumberOfEnergyPoints,
                         sizeof( double ) );
    data->Distributions[j].EnergyProbability
      = (double*)calloc( data->Distributions[j].NumberOfEnergyPoints,
                         sizeof( double ) );

    for( i = 0; i < data->Distributions[j].NumberOfEnergyPoints; i++ ) {
      if( i % 3 == 0 )
        endfReadLine( line );
      endfGetNumber( *line, (1+2*i)%6, 2,
                &data->Distributions[j].Energy[i] );
      endfGetNumber( *line, (1+2*i)%6+1, 2,
                &data->Distributions[j].EnergyProbability[i] );
    }

    switch( data->Distributions[j].EnergyDistributionLaw ) {
    case 1:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF1 ) );
      endfReadMF5LF1( line, (endfMF5LF1*)data->Distributions[j].Parameters );
      break;
    case 5:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF5 ) );
      endfReadMF5LF1( line, (endfMF5LF1*)data->Distributions[j].Parameters );
      break;
    case 7:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF7 ) );
      endfReadMF5LF7( line, (endfMF5LF7*)data->Distributions[j].Parameters );
      break;
    case 9:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF9 ) );
      endfReadMF5LF9( line, (endfMF5LF9*)data->Distributions[j].Parameters );
      break;
    case 11:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF11 ) );
      endfReadMF5LF11( line, (endfMF5LF11*)data->Distributions[j].Parameters );
      break;
    case 12:
      data->Distributions[j].Parameters = calloc( 1, sizeof( endfMF5LF12 ) );
      endfReadMF5LF12( line, (endfMF5LF12*)data->Distributions[j].Parameters );
      break;
    default:
      printf( "ERROR: Loading MF5 data and got case '%d'; HELP!!\n",
              data->Distributions[j].EnergyDistributionLaw );
      exit( -1 );
    }

  } /* end of for j number of partial energy distributions */

  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 5 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 5 mt 0: got mf '%d' mt '%d'\n",
            line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}

/*******************************************************************
** endfReadMF5LF1
**
** load the LF 1 theta spectral information
**
*/
void endfReadMF5LF1( endfLine *line, endfMF5LF1 *data )
{
  printf( "ERROR: haven't written the MF5 LF1 reader yet\n" );
  exit( -1 );
}


/*******************************************************************
** endfReadMF5LF5
**
** load the LF 5 theta spectral information
**
*/
void endfReadMF5LF5( endfLine *line, endfMF5LF5 *data )
{
  int i;


  /*
  ** Read the Energy / Theta pairs
  */
  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfThetaRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfThetaPoints );

  data->NumberOfThetaPointsInRegion
```

```
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );
  data->InterpolationSchemeInThetaRegion
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
               &data->NumberOfThetaPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
               &data->InterpolationSchemeInThetaRegion[i] );
    }

  data->ThetaEnergy
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );
  data->Theta
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->ThetaEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->Theta[i] );
  }


  /*
  ** Read the Energy / G pairs
  */
  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfGRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfGPoints );

  data->NumberOfGPointsInRegion
    = (int*)calloc( data->NumberOfGRegions, sizeof( int ) );
  data->InterpolationSchemeInGRegion
    = (int*)calloc( data->NumberOfGRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfGRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
               &data->NumberOfGPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
               &data->InterpolationSchemeInGRegion[i] );
    }

  data->GEnergy
    = (double*)calloc( data->NumberOfGPoints, sizeof( double ) );
  data->G
    = (double*)calloc( data->NumberOfGPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfGPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->GEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->G[i] );
  }

}


/*******************************************************************
** endfReadMF5LF7
**
** load the LF 7 theta spectral information
**
*/
void endfReadMF5LF7( endfLine *line, endfMF5LF7 *data )
{
  int i;

  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfThetaRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfThetaPoints );

  data->NumberOfThetaPointsInRegion
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );
  data->InterpolationSchemeInThetaRegion
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfThetaPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInThetaRegion[i] );
    }

  data->ThetaEnergy
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );
  data->Theta
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->ThetaEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->Theta[i] );
  }

}


/*******************************************************************
** endfReadMF5LF9
**
** load the LF 9 theta spectral information
**
*/
void endfReadMF5LF9( endfLine *line, endfMF5LF9 *data )
{
  int i;

  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfThetaRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfThetaPoints );

  data->NumberOfThetaPointsInRegion
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );
  data->InterpolationSchemeInThetaRegion
    = (int*)calloc( data->NumberOfThetaRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfThetaPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInThetaRegion[i] );
    }

  data->ThetaEnergy
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );
  data->Theta
    = (double*)calloc( data->NumberOfThetaPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->ThetaEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->Theta[i] );
  }

}


/*****************************************************************
** endfReadMF5LF11
**
** load the LF 11 theta spectral information
**
*/
void endfReadMF5LF11( endfLine *line, endfMF5LF11 *data )
{
  int i;

  /*
  ** Read the Energy / A pairs
  */
  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfARegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfAPoints );

  data->NumberOfAPointsInRegion
    = (int*)calloc( data->NumberOfARegions, sizeof( int ) );
  data->InterpolationSchemeInARegion
    = (int*)calloc( data->NumberOfARegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfARegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfAPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInARegion[i] );
    }

  data->AEnergy
    = (double*)calloc( data->NumberOfAPoints, sizeof( double ) );
  data->A
    = (double*)calloc( data->NumberOfAPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfAPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->AEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->A[i] );
  }


  /*
  ** Read the Energy / B pairs
  */
  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfBRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfBPoints );

  data->NumberOfBPointsInRegion
    = (int*)calloc( data->NumberOfBRegions, sizeof( int ) );
  data->InterpolationSchemeInBRegion
    = (int*)calloc( data->NumberOfBRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfBRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfBPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInBRegion[i] );
    }

  data->BEnergy
    = (double*)calloc( data->NumberOfBPoints, sizeof( double ) );
  data->B
    = (double*)calloc( data->NumberOfBPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfBPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->BEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->B[i] );
  }

}


/*******************************************************************
** endfReadMF5LF12
**
** load the LF 12 theta spectral information
**
*/
void endfReadMF5LF12( endfLine *line, endfMF5LF12 *data )
{
  int i;


  /*
  ** Read the Energy / Maximum Temperature pairs
  */
  endfReadLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfGetNumber( *line, 5, 1, &data->NumberOfMTRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfMTPoints );

  data->NumberOfMTPointsInRegion
    = (int*)calloc( data->NumberOfMTRegions, sizeof( int ) );
  data->InterpolationSchemeInMTRegion
    = (int*)calloc( data->NumberOfMTRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfMTRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfMTPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInMTRegion[i] );
    }

  data->MTEnergy
    = (double*)calloc( data->NumberOfMTPoints, sizeof( double ) );
  data->MaximumTemperature
    = (double*)calloc( data->NumberOfMTPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfMTPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->MTEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->MaximumTemperature[i] );
  }

}


/*******************************************************************
** endfPrintMF5
**
```

```
** print an mf5 section record
**
** expects 'line->body' to point to a writable file pointer
** expects 'line->mf' to be set to mf5
** expects 'line->mt' to be set to current mt
**
*/
void endfPrintMF5( endfLine *line, endfMF5 *data )
{
  int     i, j;
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ ZA, AWR, 0, 0, NK, 0 ] */
  endfPutNumber( line, 1, 2, (void*)&data->TargetZA );
  endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfPartialEnergyDistributions );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );


  for( j = 0; j < data->NumberOfPartialEnergyDistributions; j++ ) {

    endfNextLine( line );
    /* [ U, 0,0, 0, LF, NR, NP ] */
    endfPutNumber( line, 1, 2, (void*)&data->Distributions[j].UpperEnergyDelta );
    endfPutNumber( line, 2, 2, (void*)&doublezero );
    endfPutNumber( line, 3, 1, (void*)&intzero );
    endfPutNumber( line, 4, 1, (void*)&data->Distributions[j].EnergyDistributionLaw );
    endfPutNumber( line, 5, 1, (void*)&data->Distributions[j].NumberOfEnergyRegions );
    endfPutNumber( line, 6, 1, (void*)&data->Distributions[j].NumberOfEnergyPoints );
    endfPrintLineToFile( *line );


    for( i = 0; i < data->Distributions[j].NumberOfEnergyRegions; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   1,
                (void*)&data->Distributions[j].NumberOfEnergyPointsInRegion[i] );
      endfPutNumber( line, (1+2*i)%6+1, 1,
                (void*)&data->Distributions[j].InterpolationSchemeInEnergyRegion[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );


    for( i = 0; i < data->Distributions[j].NumberOfEnergyPoints; i++ ) {
      if( i % 3 == 0 )
        endfNextLine( line );
      endfPutNumber( line, (1+2*i)%6,   2,
                (void*)&data->Distributions[j].Energy[i] );
      endfPutNumber( line, (1+2*i)%6+1, 2,
                (void*)&data->Distributions[j].EnergyProbability[i] );
      if( i % 3 == 2 )
        endfPrintLineToFile( *line );
    }
    if( i % 3 != 0 )
      endfPrintLineToFile( *line );


    switch( data->Distributions[j].EnergyDistributionLaw ) {
    case 1:
      endfPrintMF5LF1( line, (endfMF5LF1*)data->Distributions[j].Parameters );
      break;
    case 5:
      endfPrintMF5LF5( line, (endfMF5LF5*)data->Distributions[j].Parameters );
      break;
    case 7:
      endfPrintMF5LF7( line, (endfMF5LF7*)data->Distributions[j].Parameters );
      break;
    case 9:
      endfPrintMF5LF9( line, (endfMF5LF9*)data->Distributions[j].Parameters );
      break;
    case 11:
      endfPrintMF5LF11( line, (endfMF5LF11*)data->Distributions[j].Parameters );
      break;
    case 12:
      endfPrintMF5LF12( line, (endfMF5LF12*)data->Distributions[j].Parameters );
      break;
    default:
      printf( "ERROR: Printing MF5 data and got case '%d'; HELP!!\n",
              data->Distributions[j].EnergyDistributionLaw );
    }

  } /* end of for j number of partial energy distributions */

  /*
  ** print end of record marker
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintMF5LF1
**
** print the LF 1 theta spectral information
**
*/
void endfPrintMF5LF1( endfLine *line, endfMF5LF1 *data )
{
  printf( "ERROR: haven't written the MF5 LF1 writer yet\n" );
  exit( -1 );
}


/*******************************************************************
** endfPrintMF5LF5
**
** print the LF 5 theta spectral information
**
*/
void endfPrintMF5LF5( endfLine *line, endfMF5LF5 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  /*
  ** Print the Energy / Theta pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfThetaRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfThetaPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
               (void*)&data->NumberOfThetaPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
               (void*)&data->InterpolationSchemeInThetaRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->ThetaEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->Theta[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  /*
  ** Print the Energy / G pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfGRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfGPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfGRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
               (void*)&data->NumberOfGPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
               (void*)&data->InterpolationSchemeInGRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfGPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->GEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->G[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );
```

```
}


/*****************************************************************
** endfPrintMF5LF7
**
** print the LF 7 theta spectral information
**
*/
void endfPrintMF5LF7( endfLine *line, endfMF5LF7 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  /*
  ** Print the Energy / Theta pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfThetaRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfThetaPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
               (void*)&data->NumberOfThetaPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
               (void*)&data->InterpolationSchemeInThetaRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->ThetaEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->Theta[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


}


/*****************************************************************
** endfPrintMF5LF9
**
** print the LF 9 theta spectral information
**
*/
void endfPrintMF5LF9( endfLine *line, endfMF5LF9 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;
  /*
  ** Print the Energy / Theta pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfThetaRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfThetaPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
              (void*)&data->NumberOfThetaPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
              (void*)&data->InterpolationSchemeInThetaRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfThetaPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->ThetaEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->Theta[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintMF5LF11
**
** print the LF 11 theta spectral information
**
*/
void endfPrintMF5LF11( endfLine *line, endfMF5LF11 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  /*
  ** Print the Energy / A pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfARegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfAPoints );
  endfPrintLineToFile( *line );
  for( i = 0; i < data->NumberOfARegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
               (void*)&data->NumberOfAPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
               (void*)&data->InterpolationSchemeInARegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfAPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->AEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->A[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  /*
  ** Print the Energy / B pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfBRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfBPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfBRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
               (void*)&data->NumberOfBPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
               (void*)&data->InterpolationSchemeInBRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfBPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->BEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->B[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintMF5LF12
**
```

```
** print the LF 12 theta spectral information
**
*/
void endfPrintMF5LF12( endfLine *line, endfMF5LF12 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  /*
  ** Print the Energy / Maximum Temperature pairs
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfMTRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfMTPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfMTRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
              (void*)&data->NumberOfMTPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
              (void*)&data->InterpolationSchemeInMTRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfMTPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   2, (void*)&data->MTEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->MaximumTemperature[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );

}
```

endfMF6.h

```
#ifndef endfMF6_h
#define endfMF6_h

#include <stdio.h>
#include <stdlib.h>

#include "endfLine.h"
#include "endfNumber.h"

/*
** Structure: endfEmissionSpectrum
*/
typedef struct endfemissionspectrum {                /* ENDF Parameter Name */
  double   IncidentEnergy;                       /* E */
  int      NumberOfDiscreteEmissions;            /* ND */
  int      NumberOfAngularParameters;            /* NA */
  int      NumberOfEntries;                      /* NW */
  int      NumberOfEmissionEnergies;             /* NEP */
  double  *EmissionEnergy;
  double **Parameters;
} endfEmissionSpectrum;

/*
** Structure: endfLaw1
*/
typedef struct endflaw1 {                            /* ENDF Parameter Name */
  int      AngularRepresentation;                /* LANG */
  int      InterpolationSchemeForSecEnergy;      /* LEP */
  int      NumberOfSecEnergyRegions;             /* NR */
  int      NumberOfSecEnergyPoints;              /* NE */
  int     *NumberOfSecEnergyPointsInRegion;
  int     *InterpolationSchemeInSecEnergyRegion;
  endfEmissionSpectrum  *ES;
} endfLaw1;

/*
** Structure: endfSecondary
*/
typedef struct endfsecondary {                      /* ENDF Parameter Name */
  double   ParticleZA;                          /* ZAP */
  double   ParticleAWR;                         /* AWP */
  int      ProductModifier;                     /* LIP */
  int      ReactionLaw;                         /* LAW */
  int      NumberOfYieldRegions;                /* NR */
  int      NumberOfYieldPoints;                 /* NP */
  int     *NumberOfYieldPointsInRegion;
  int     *InterpolationSchemeInYieldRegion;
  double  *YieldEnergy;
  double  *Yield;
  void    *LawData;
} endfSecondary;

/*
** Structure: endfMF6
*/
typedef struct endfmf6 {                      /* ENDF Parameter Name */
  double     TargetZA;                    /* ZA */
  double     TargetAWR;                   /* AWR */
  int        FrameOfReference;            /* LCT */
  int        NumberOfSubsections;         /* NK */
  endfSecondary  *Secondaries;
} endfMF6;


/*
** Function: endfReadMF6
*/
void endfReadMF6( endfLine *line, endfMF6 *data );

/*
** Function: endfReadMF6Secondary
*/
void endfReadMF6Secondary( endfLine *line, endfSecondary *data );

/*
** Function: endfReadMF6Law1
*/
void endfReadMF6Law1( endfLine *line, endfLaw1 *data );

/*
** Function: endfReadMF6EmissionSpectrum
*/
void endfReadMF6EmissionSpectrum( endfLine *line, endfEmissionSpectrum *data );

/*
** Function: endfPrintMF6
*/
void endfPrintMF6( endfLine *line, endfMF6 *data );
```

```
/*
** Function: endfPrintMF6Secondary
*/
void endfPrintMF6Secondary( endfLine *line, endfSecondary *data );

/*
** Function: endfPrintMF6Law1
*/
void endfPrintMF6Law1( endfLine *line, endfLaw1 *data );

/*
** Function: endfPrintMF6EmissionSpectrum
*/
void endfPrintMF6EmissionSpectrum( endfLine *line, endfEmissionSpectrum *data );


#endif
```

endfMF6.c

```
#include "endfMF6.h"

/*******************************************************************
** endfReadMF6
**
** load the file 6 general information
**
*/
void endfReadMF6( endfLine *line, endfMF6 *data )
{
  int  i;

  /* [ ZA, AWR, 0, LCT, NK, 0 ] */
  endfGetNumber( *line, 1, 2, &data->TargetZA );
  endfGetNumber( *line, 2, 2, &data->TargetAWR );
  endfGetNumber( *line, 4, 1, &data->FrameOfReference );
  endfGetNumber( *line, 5, 1, &data->NumberOfSubsections );


  data->Secondaries
    = (endfSecondary*)calloc( data->NumberOfSubsections, sizeof( endfSecondary ) );

  /*
  ** read the secondary subsections
  */
  for( i = 0; i < data->NumberOfSubsections; i++ )
    endfReadMF6Secondary( line, &data->Secondaries[i] );


  /*
  ** error check on end of section
  */
  endfReadLine( line );
  if( line->mf != 6 && line->mt != 0 ) {
    printf( "ERROR: expected end of section mf 6 mt 0: got mf '%d' mt '%d'\n",
           line->mf, line->mt );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfReadMF6Secondary
**
** load the MF 6 secondary information
**
*/
```

```
void endfReadMF6Secondary( endfLine *line, endfSecondary *data )
{
  int  i;

  endfReadLine( line );
  /* [ ZAP, AWP, LIP, LAW, NR, NP ] */
  endfGetNumber( *line, 1, 2, &data->ParticleZA );
  endfGetNumber( *line, 2, 2, &data->ParticleAWR );
  endfGetNumber( *line, 3, 1, &data->ProductModifier );
  endfGetNumber( *line, 4, 1, &data->ReactionLaw );
  endfGetNumber( *line, 5, 1, &data->NumberOfYieldRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfYieldPoints );


  data->NumberOfYieldPointsInRegion
    = (int*)calloc( data->NumberOfYieldRegions, sizeof( int ) );
  data->InterpolationSchemeInYieldRegion
    = (int*)calloc( data->NumberOfYieldRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfYieldRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfYieldPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInYieldRegion[i] );
    }

  data->YieldEnergy
    = (double*)calloc( data->NumberOfYieldPoints, sizeof( double ) );
  data->Yield
    = (double*)calloc( data->NumberOfYieldPoints, sizeof( double ) );

  for( i = 0; i < data->NumberOfYieldPoints; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 2, &data->YieldEnergy[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 2, &data->Yield[i] );
  }

  switch( data->ReactionLaw ) {
  case 1:
    data->LawData = (endfLaw1*)calloc( 1, sizeof( endfLaw1 ) );
    endfReadMF6Law1( line, (endfLaw1*)data->LawData );
    break;
  default:
    printf( "ERROR: don't know reaction law '%d'\n", data->ReactionLaw );
    endfPrintLine( *line );
    exit( -1 );
  }

}


/*******************************************************************
** endfReadMF6Law1
**
** load the MF 6 secondary law information
**
*/
void endfReadMF6Law1( endfLine *line, endfLaw1 *data )
{
  int  i;

  endfReadLine( line );
  /* [ 0.0, 0.0, LANG, LEP, NR, NE ] */
  endfGetNumber( *line, 3, 1, &data->AngularRepresentation );
  endfGetNumber( *line, 4, 1, &data->InterpolationSchemeForSecEnergy );
  endfGetNumber( *line, 5, 1, &data->NumberOfSecEnergyRegions );
  endfGetNumber( *line, 6, 1, &data->NumberOfSecEnergyPoints );
  data->NumberOfSecEnergyPointsInRegion
    = (int*)calloc( data->NumberOfSecEnergyRegions, sizeof( int ) );
  data->InterpolationSchemeInSecEnergyRegion
    = (int*)calloc( data->NumberOfSecEnergyRegions, sizeof( int ) );

  for( i = 0; i < data->NumberOfSecEnergyRegions; i++ ) {
    if( i % 3 == 0 )
      endfReadLine( line );
    endfGetNumber( *line, (1+2*i)%6, 1,
              &data->NumberOfSecEnergyPointsInRegion[i] );
    endfGetNumber( *line, (1+2*i)%6+1, 1,
              &data->InterpolationSchemeInSecEnergyRegion[i] );
  }

  data->ES = (endfEmissionSpectrum*)calloc( data->NumberOfSecEnergyPoints,
                                        sizeof( endfEmissionSpectrum ) );

  for( i = 0; i < data->NumberOfSecEnergyPoints; i++ )
    endfReadMF6EmissionSpectrum( line, &data->ES[i] );

}


/*********************************************************************
** endfReadMF6EmissionSpectrum
**
** load the MF 6 secondary emission spectrum information
**
*/
void endfReadMF6EmissionSpectrum( endfLine *line, endfEmissionSpectrum *data )
{
  int  i;
  int  e;    /* emission energy index */
  int  epe;  /* entries per emission energy */

  endfReadLine( line );
  /* [ 0.0, E, ND, NA, NW, NEP ] */
  endfGetNumber( *line, 2, 2, &data->IncidentEnergy );
  endfGetNumber( *line, 3, 1, &data->NumberOfDiscreteEmissions );
  endfGetNumber( *line, 4, 1, &data->NumberOfAngularParameters );
  endfGetNumber( *line, 5, 1, &data->NumberOfEntries );
  endfGetNumber( *line, 6, 1, &data->NumberOfEmissionEnergies );


  data->EmissionEnergy = (double*)calloc( data->NumberOfEmissionEnergies,
                                       sizeof( double ) );
  data->Parameters = (double**)calloc( data->NumberOfEmissionEnergies,
                                    sizeof( double* ) );
  for( i = 0; i < data->NumberOfEmissionEnergies; i++ )
    data->Parameters[i] = (double*)calloc( data->NumberOfAngularParameters + 1,
                                       sizeof( double ) );

  e = -1;  /* index get incremented to 0 on first pass before use */
  epe = data->NumberOfAngularParameters + 2;
  for( i = 0; i < data->NumberOfEntries; i++ ) {
    if( i % 6 == 0 )
      endfReadLine( line );
    if( i % epe == 0 )
      endfGetNumber( *line, (i%6)+1, 2, &data->EmissionEnergy[++e] );
    else
      endfGetNumber( *line, (i%6)+1, 2, &data->Parameters[e][(i%epe)-1] );
  }

}


/*********************************************************************
*****************
** endfPrintMF6
**
** load the file 6 general information
```

```
**
*/
void endfPrintMF6( endfLine *line, endfMF6 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ ZA, AWR, 0, LCT, NK, 0 ] */
  endfPutNumber( line, 1, 2, (void*)&data->TargetZA );
  endfPutNumber( line, 2, 2, (void*)&data->TargetAWR );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&data->FrameOfReference );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfSubsections );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );


  /*
  ** print the secondary subsections
  */
  for( i = 0; i < data->NumberOfSubsections; i++ )
    endfPrintMF6Secondary( line, &data->Secondaries[i] );


  /*
  ** print end of record marker
  */
  endfNextLine( line );
  /* [ 0.0, 0.0, 0, 0, 0, 0 ] */
  line->mt = 0;
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&intzero );
  endfPutNumber( line, 4, 1, (void*)&intzero );
  endfPutNumber( line, 5, 1, (void*)&intzero );
  endfPutNumber( line, 6, 1, (void*)&intzero );
  endfPrintLineToFile( *line );

}


/*******************************************************************
** endfPrintMF6Secondary
**
** load the MF 6 secondary information
**
*/
void endfPrintMF6Secondary( endfLine *line, endfSecondary *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;

  endfNextLine( line );
  /* [ ZAP, AWP, LIP, LAW, NR, NP ] */
  endfPutNumber( line, 1, 2, (void*)&data->ParticleZA );
  endfPutNumber( line, 2, 2, (void*)&data->ParticleAWR );
  endfPutNumber( line, 3, 1, (void*)&data->ProductModifier );
  endfPutNumber( line, 4, 1, (void*)&data->ReactionLaw );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfYieldRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfYieldPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfYieldRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6, 1,
              (void*)&data->NumberOfYieldPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
              (void*)&data->InterpolationSchemeInYieldRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfYieldPoints; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6, 2, (void*)&data->YieldEnergy[i] );
    endfPutNumber( line, (1+2*i)%6+1, 2, (void*)&data->Yield[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );


  switch( data->ReactionLaw ) {
  case 1:
    endfPrintMF6Law1( line, (endfLaw1*)data->LawData );
    break;
  default:
    printf( "ERROR: don't know reaction law '%d'\n", data->ReactionLaw );
    exit( -1 );
  }


}


/*******************************************************************
** endfPrintMF6Law1
**
** load the MF 6 secondary law information
**
*/
void endfPrintMF6Law1( endfLine *line, endfLaw1 *data )
{
  int     i;
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ 0.0, 0.0, LANG, LEP, NR, NE ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&doublezero );
  endfPutNumber( line, 3, 1, (void*)&data->AngularRepresentation );
  endfPutNumber( line, 4, 1, (void*)&data->InterpolationSchemeForSecEnergy );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfSecEnergyRegions );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfSecEnergyPoints );
  endfPrintLineToFile( *line );


  for( i = 0; i < data->NumberOfSecEnergyRegions; i++ ) {
    if( i % 3 == 0 )
      endfNextLine( line );
    endfPutNumber( line, (1+2*i)%6,   1,
              (void*)&data->NumberOfSecEnergyPointsInRegion[i] );
    endfPutNumber( line, (1+2*i)%6+1, 1,
              (void*)&data->InterpolationSchemeInSecEnergyRegion[i] );
    if( i % 3 == 2 )
      endfPrintLineToFile( *line );
  }
  if( i % 3 != 0 )
    endfPrintLineToFile( *line );
  for( i = 0; i < data->NumberOfSecEnergyPoints; i++ )
    endfPrintMF6EmissionSpectrum( line, &data->ES[i] );


}


/*****************************************************************
** endfPrintMF6EmissionSpectrum
**
** load the MF 6 secondary emission spectrum information
**
*/
void endfPrintMF6EmissionSpectrum( endfLine *line, endfEmissionSpectrum *data )
{
  int     i;
  int     e;    /* emission energy index */
  int     epe;  /* entries per emission energy */
  int     intzero = 0;
  double  doublezero = 0.0;


  endfNextLine( line );
  /* [ 0.0, E, ND, NA, NW, NEP ] */
  endfPutNumber( line, 1, 2, (void*)&doublezero );
  endfPutNumber( line, 2, 2, (void*)&data->IncidentEnergy );
  endfPutNumber( line, 3, 1, (void*)&data->NumberOfDiscreteEmissions );
  endfPutNumber( line, 4, 1, (void*)&data->NumberOfAngularParameters );
  endfPutNumber( line, 5, 1, (void*)&data->NumberOfEntries );
  endfPutNumber( line, 6, 1, (void*)&data->NumberOfEmissionEnergies );
  endfPrintLineToFile( *line );


  e = -1;  /* index get incremented to 0 on first pass before use */
  epe = data->NumberOfAngularParameters + 2;
  for( i = 0; i < data->NumberOfEntries; i++ ) {
    if( i % 6 == 0 )
      endfNextLine( line );
    if( i % epe == 0 )
      endfPutNumber( line, (i%6)+1, 2, (void*)&data->EmissionEnergy[++e] );
    else
      endfPutNumber( line, (i%6)+1, 2, (void*)&data->Parameters[e][(i%epe)-1] );
    if( i % 6 == 5 )
      endfPrintLineToFile( *line );
  }
  if( i % 6 != 0 )
    endfPrintLineToFile( *line );

}
```

Makefile

```
CC=/opt/SUNWspro/bin/cc

CFLAGS = # -g -xsb

LIBS = -lm

SRCS =         mkpnt.c \
               \
               acepnIO.c \
               \
               afeCreateNTableHeader.c \
               afeCollectEnergies.c \
               afeGetMTInformation.c \
               afeGetMTNames.c \
               afeGetMTProducts.c \
               afeMakeNTable.c \
               afeVerifyNTable.c

ENDF_SRCS =    endf6.c \
               endfLine.c \
               endfNumber.c \
               endfConvert.c \
               endfMF1.c \
               endfMF1MT451.c \
               endfMF2.c \
               endfMF3.c \
               endfMF4.c \
               endfMF5.c \
               endfMF6.c

OBJS = ${SRCS:.c=.o}
ENDF_OBJS = ${ENDF_SRCS:.c=.o}


mkpnt: ${OBJS} ${ENDF_OBJS}
       ${CC} -o mkpnt ${CFLAGS} ${INCLUDE} ${OBJS} ${ENDF_OBJS} ${LIBS}

endf: ${ENDF_OBJS}


deps:
       ${CC} -xM1 ${SRCS} ${ENDF_SRCS}


test: test.o
       ${CC} -o test ${CFLAGS} test.o ${ENDF_OBJS} ${LIBS}



tar:
       tar cvf mkpnt.tar *.c *.h mkpnt.wst Makefile
       gzip mkpnt.tar

clean:
       rm -f ${OBJS} mkpnt.o test.o mkpnt test

rmold:
       rm -f *~

clobber: clean rmold
```

APPENDIX C
PHOTONUCLEAR PATCH FILE

The following text is the photonuclear patch to the MCNP code. As per the MCNP user manual on how to modify the code base, it is in the form of a PRPR patch file. Instructions on how to build the modified version of the code are contained in the first section of the file itself.

```
*/
*/ ******************************************************************
*/  Photonuclear patch to MCNP4B2
*/ ******************************************************************
*/  The following set of changes is the frozen version of the
*/  photonuclear patch to MCNP used for this dissertation work.
*/
*/  To build this version of the code, it is necessary to obtain the
*/  CCC660 MCNP4B code package from the RSICC computer code center.
*/  (http://www-rsicc.ornl.gov/rsic.html)
*/  Once this has been done, follow the normal installation procedures
*/  with the following exceptions:
*/
*/  (1) Copy this patch file into the directory where the executable
*/      will be built, and
*/  (2) stop the installation before execution of the makemcnp script
*/      and add the lines labeled "New Line":
*/
*/  --- from makemcnp: lines removed ---
*/  Old Line:  cp mcnpf.id codef
*/  Old Line:  cp patchf patch
*/  Old Line:  prpr
*/  New Line:  rm patch compile
*/  New Line:  mv newid codef
*/  New Line:  head -1 patchf > patch
*/  New Line:  cat patchf.pn >> patch
*/  New Line:  prpr
*/  Old Line:  fsplit compile > clog
*/  Old Line:  rm -f compile codef patch newid clog
*/  --- from makemcnp: lines removed ---
*/
*/  This convolution is necessary as the photonuclear patch was built on
*/  MCNP4B release 2 and the author did not want to reconcile the changes
*/  in the install.fix file with the new patchf file.
*/
*/
*/ ******************************************************************
*/  Changes to zc
*ident zcPN
*/ ******************************************************************
*/
*d,zc4b.1
      parameter (kod='mcnp',ver='PN')
*i,zc4b.5
     1 maxsec=8,mixs=12,
*/
*/ ******************************************************************
*/  Changes to vv
*ident vvPN
*/ *********************************************************************
*/
*d,vv4a.3
     2 jrwb(17,mipt),jsf(mjsf),mfiss(22),nvs(maxv),itty,jtty
*i,vv4a.5
     3 htn*9,
*i,vv4a.10
     2 htn,
*/
*/ *********************************************************************
*/  Changes to cm
*ident cmPN
*/ *********************************************************************
*/
*d,cm4b.3
     1 lfixcm=3*mxdt+mink+11*mipt+2*maxv+2*maxf+286)
*d,cm4b.4
      parameter (nvarcm=108*mipt+144,lvarcm=mipt*(1+8*mxdx)+mcpu+329)
*i,cm.36
     4 ispn,
*i,cm4b.19
     7 lixs,lizn,llmn,lpnt,
*d,cm.63
     1 osum2(3,3),pax(6,17,mipt),prn,rani,ranj,rdum(50),rijk,rkk,
*i,cm.69
     8 npum,
*d,cm4b.49
     1 colltc(mipt),deb,dti(mlgc),eacctc(2),eg0,ergace,paxtc(6,17,mipt),
*i,cm.175
     2 totpn,
*i,cm.184
     7 ntyr,
*d,cm4b.63,cm4b.64
      parameter (ntskcm=108*mipt+40*(mxlv+1)+3*maxf+mlgc+105,
     1 ltskcm=mipt*(2+8*mxdx)+5*(mxlv+1)+2*mlgc+maxf+48,ltskpt=39)
*i,cm4b.75
     7 pnt(1),
*i,cm4b.80
     3 izn(1),lmn(1),ixs(mixs,maxsec,1),
*d,cm4b.87
*if -def,pointer,11
*i,cm4b.91
     4 pnt,
*i,cm4b.93
     2 izn,lmn,ixs,
*d,cm4b.97
*if def,pointer,23
*i,cm4b.104
     7 (kdy,pnt),
*i,cm4b.110
     2 (kdy,izn),(kdy,lmn),(kdy,ixs),
*d,cm4b.120
     2 pac(mipt,10,1),pan(3,8,1),pcc(3,1),pwb(mipt,21,1),rkpl(19,1),
*i,cm4a.97
     3 lxn(1),
*d,cor4-2.16
*if -def,pointer,4
*i,cm4a.98
     3 lxn,
*d,cor4-2.17
*if def,pointer,6
*i,cm4a.99
     3 (kdy,lxn),
*/
*/ *********************************************************************
*/  Changes to blkdat
*ident bdPN
*/ *********************************************************************
*/
```

```
*d,bd.37,bd.38
      data jrwb/
     1     0,  3,  4,  6,  7,  8,  9, 10, 11, 12,
     1     0, 15, 16, 17,  0,  0,  0,
     2     0,  3,  4,  6,  7,  8,  9, 10, 11, 12,
     2     0, 18, 19,  0,  0,  0, 20,
     3     0,  3,  4,  6,  7,  8,  9, 10, 11, 12,
     3     0,  0,  0,  0,  0,  0,  0/
*i,bd.82
     2 htn/'cdytpmgue'/,
*d,bd4b.3
     3 hsd/'sequential','direct'/,ibin/'fdusmcet'/,loddat/'01/14/00'/,
*/
*/ *********************************************************************
*/  Changes to jc
*ident jcPN
*/ *********************************************************************
*/
*d,jc4b.1
      parameter (nkcd=89,ntalmx=100,mopts=6)
*i,jc4a.5
     2 llxn,
*/
*/ *********************************************************************
*/  Changes to ibldat
*ident ibPN
*/ *********************************************************************
*/
*i,ib4b.6
      data cnm(89),(krq(i,89),i=1,7)/'mpn  ',0,0, 0,0, 2,   0,0/
*d,ib4a.10
      data hmopt/'gas','estep','nlib','plib','elib','pnlib'/
*/
*/ *********************************************************************
*/  Changes to imcn
*ident imPN
*/ *********************************************************************
*/
*d,im.26
      if(ispn.ne.0.and.kpt(2).eq.0)then
        ispn=0
        call erprnt(2,2,0,0,0,0,0,0,
     1    '50hphotonuclear turned off. photons not on mode card.')
      endif
      if(ispn.ne.0)n=n+1
      mxe1=mix*max(1,n)
      mxe1=mxe1+indt
*i,im4a.18
      m4=ichar(' ')+256*(ichar(' ')+256*ichar('u'))
*d,im4a.22
      lxd(llxd+3,i)=m3
      lxn(llxn+i)=m4
  195 pnt(lpnt+i)=huge
*/
*/ *********************************************************************
*/  Changes to newcd1
*ident nfPN
*/ *********************************************************************
*/
*d,nf.70
     2      9910,9920,9930, 110)ica-55
*d,nf4b.5
*d,nf4b.6
*/
*/ *********************************************************************
*/  Changes to nexit1
*ident nxPN
*/ *********************************************************************
*/
*d,nx4b.2
      go to( 10, 10, 10,250, 10, 10, 10, 10,255, 10,258, 10,259,260, 10,
*d,nx.14
     2      9910,9920,9930, 10)ica-55
*d,nx4a.127,nx4b.42
c          want to count the number of zaids and atom fractions
c            while ignoring material options.
c          nwc is number of zaids plus atom fractions read so far
c            (nwc is incremented automatically in routine items)
c          m1c > 0 indicates already processing material option
c
c       if already found zaid, count next atom fraction
  170 if (mod(nwc,2).eq.0) return
c
c       if processing material option, don't count "=" or option
      if (m1c.ne.0) then
         nwc=nwc-1
         if(hitm.eq.'=')return
         m1c=0
         return
      endif
c
c       count zaids while ignoring material options
      do 175 i=1,mopts
         if (hitm.eq.hmopt(i)) then
            m1c=i
            nwc=nwc-1
            return
         endif
  175 continue
      return
*i,nx.121
c
c       turn on photonuclear physics( + natural coll / - biased coll )
      if(nqp(2).ne.0.and.iitm.ne.0.and.nwc.eq.4)ispn=iitm
*d,nx.123,nx.145
*/
*/ *******************************************************************
*/  Changes to oldcd1
*ident olPN
*/ *******************************************************************
*/
*d,ol.16
     2      9910,9920,9930, 10)ica-55
*d,ol4b.2,ol4b.19
c
c      for each cell containing material, make space for nuclide summary
  240 n=0
      do 250 i=1,mxa
         if(mat(lmat+i).eq.icn)n=n+1
  250 continue
      npn=npn+n*nwc/2
c
c        increment the total number of isotopes seen so far
      mix=mix+nwc/2
c
c        update the maximum number of nuclides seen for any material
      mnnm=max(mnnm,nwc/2)
      return
*/
*/ *******************************************************************
*/  Changes to setdas
*ident sdPN
*/ *******************************************************************
*/
*d,sd.67
      lpnt=lpmg+max(0,mcal-1)*igm*npn
      lpru=lpnt+nmat1
*d,sd.96,sd.97
      lixs=lixl+3*mxe1
      liza=lixs+mixs*maxsec*mxe1
      lizn=liza+mix
      ljar=lizn+mix
```

```
*d,sd.118
      llmn=llme+mipt*mix
      llmt=llmn+mix
*d,sd.147
      lpcc=lpan+3*8*npn
*d,sd.149
      lrkp=lpwb+mipt*21*(mxa+1)
*d,sd4b.45,sd4b.46
     2 18*mxa*mgww(mipt+1)+mipt*10*mxa+3*8*npn+3*mxa*kpt(2)+
     3 mipt*21*mxa+mxxs/2)*mt)*ndp2
*d,sd4a.20
      llxn=llxd+mipt*nmat1
      lmfm=llxn+nmat1
*/
*/ ********************************************************************
*/  Changes to newcrd
*ident nePN
*/ ********************************************************************
*/
*d,ne.94
     2      9910,9920,9930, 660)ica-55
*i,ne4b.30
c
c >>>>>  photonuclear isotope override                              mpn
c if mpn has a corresonding m card, return and process card items
c   m1c is set to the material number, ie when was it seen sequentially
c   m2c is set to jmd(ljmd+mat), ie the index of the first entry in iza
c   m3c is set to npq(lnpq+mat), ie the number of entries for mat in iza
c   (note the implicit dependence that mX card comes before mpnX card)
c
  660 continue
c
c        mpn override card only legal if photonuclear physics is on
c
      if(ispn.eq.0) then
         call erprnt(2,2,0,0,0,0,0,0,
     1     '50hmpn card ignored while photonuclear physics is off')
         return
      endif
c
c        find the corresponding material card
      do 670 i=1,nmat
         if(icn.eq.nmt(lnmt+i)) m1c=i
  670 continue
c
c        if corresponding material card exists, set pointers
      if(m1c.ne.0) then
        m2c=jmd(ljmd+m1c)
        m3c=npq(lnpq+m1c)
c
c       if mpn card has no corresponding m card (or m card is after mpn)
c         print warning to user to indicate card ignored
      else
        call erprnt(2,1,1,icn,0,0,0,0,
     1   '13hmpn override ,i4,'//
     2   '56h (has no/is before) corresponding m card and is ignored.')
      endif
      return
*/
*/ ********************************************************************
*/  Changes to chekit
*ident cePN
*/ ********************************************************************
*/
*d,ce4b.1
     2      1180,9910,9920,9930,1240)ica-55
*d,ce4a.153
c        m1c > 0, already processing material option.
*d,ce4a.160,ce4a.163
      if(m1c.eq.3.and.index(' cdym',hitm(i+3:i+3)).eq.0)
     1    call erprnt(2,1,0,0,0,0,0,1,
     2    '49hdefault nuetron table set to wrong particle type.')
      if(m1c.eq.4.and.index(' pg',hitm(i+3:i+3)).eq.0)
     1    call erprnt(2,1,0,0,0,0,0,1,
     2    '53hdefault photoatomic table set to wrong particle type.')
      if(m1c.eq.5.and.index(' e',hitm(i+3:i+3)).eq.0)
     1    call erprnt(2,1,0,0,0,0,0,1,
     2    '50hdefault electron table set to wrong particle type.')
      if(m1c.eq.6.and.index(' u',hitm(i+3:i+3)).eq.0)
     1    call erprnt(2,1,0,0,0,0,0,1,
     2    '54hdefault photonuclear table set to wrong particle type.')
*d,ce4a.172
      if(kpt(2).eq.0.and.index('pgu',hitm(i+3:i+3)).ne.0)
*i,ce4a.177
c
c     check to ensure that cells do not contain none transort tables
*d,ce.383,ce.384
      if(index(htn,ht(10:10)).eq.0)call erprnt(2,1,0,0,0,0,0,1,
     1 '49hzaid must end in table class id (see Appendix F)')
*i,ce4b.27
c
c >>>>>  photonuclear isotope override card                         mpn
c
c        entries have already been checked to ensure integer type
c          ( from krq(5,mpn) set to 2 in ibldat and check above)
c
c        all entries must be positive integers or zero
c          valid za's are range 1 to 999999
 1240 if(iitm.lt.0.or.iitm.gt.999999)
     1     call erprnt(2,1,1,iitm,0,0,0,0,
     2     '52hisotope override must be valid (non-negative) za not,i7')
      return
*/
*/ ******************************************************************
*/  Changes to nextit
*ident nyPN
*/ ******************************************************************
*/
*d,ny4b.1
     2      1480,1490,1510,1620,9910,9920,9930,1660)ica-55
*d,ny4a.24,ny.389
c        m1c > 0, already processing material option.
  710 if (m1c.ne.0) then
         nwc=nwc-1
         if(hitm.eq.'=')return
c
c        process material option
         if(m1c.eq.1.and.nee.gt.0)emi(lemi+nmat)=ritm
         if(m1c.eq.2.and.nee.gt.0)nsb(lnsb+nmat)=iitm
c
c        processing identifier value for default library.
         if (m1c.ge.3.and.m1c.le.6) then
            i=index(hitm,'.')
            ht=hitm(i+1:i+3)
            if(ht(1:1).eq.' ')ht(1:1)='0'
            if(ht(2:2).eq.' ')ht(2:2)='0'
c
c        content-free entry does not change system default.
            if(ht(1:3).eq.'00 '.or.ht(1:3).eq.'000')return
c
c        set the appropriate default library
            if (m1c.eq.3) then
               if (ht(3:3).eq.' ') ht(3:3)='c'
               lxd(llxd+1,nmat)=ichar(ht(1:1))+256*(ichar(ht(2:2))+
     1              256*ichar(ht(3:3)))
            elseif (m1c.eq.4) then
               if (ht(3:3).eq.' ') ht(3:3)='p'
               lxd(llxd+2,nmat)=ichar(ht(1:1))+256*(ichar(ht(2:2))+
     1              256*ichar(ht(3:3)))
            elseif (m1c.eq.5) then
               if (ht(3:3).eq.' ') ht(3:3)='e'
               lxd(llxd+3,nmat)=ichar(ht(1:1))+256*(ichar(ht(2:2))+
     1              256*ichar(ht(3:3)))
            elseif (m1c.eq.6) then
               if (ht(3:3).eq.' ') ht(3:3)='u'
               lxn(llxn+nmat)=ichar(ht(1:1))+256*(ichar(ht(2:2))
     1              +256*ichar(ht(3:3)))
            endif
         endif
         m1c=0
         return
      endif
c
c       else check for material option
      do 715 i=1,mopts
         if (hitm.eq.hmopt(i)) then
            m1c=i
            nwc=nwc-1
            return
         endif
 715  continue
c
c        else process zaid entry or fraction entry.
      if (mod(nwc,2).ne.0) then
         mix=mix+1
         ht=' '
         if(index(hitm,'.').eq.0)hitm(nitm+1:nitm+1)='.'
         ht(8-index(hitm,'.'):10)=hitm
         if(ht(8:8).ne.' '.and.ht(9:9).eq.' ')ht(9:9)='0'
         if(ht(8:10).eq.'00 '.or.ht(8:10).eq.'000')ht(8:10)=' '
         read(ht(1:6),'(i6)')iza(liza+mix)
         izn(lizn+mix)=iza(liza+mix)
         kmm(lkmm+mix)=ichar(ht(8:8))+256*(ichar(ht(9:9))+
     1        256*ichar(ht(10:10)))
      else
         fme(lfme+mix)=ritm
         if(ritm.eq.0.)mix=mix-1
      endif
      return
*i,ny4b.20
c
c >>>>>  photonuclear isotope override                              mpn
c
c        if mpn does not correspond to an m card, ignore all entries
c
 1660 if(m1c.eq.0) return
c
c        otherwise save entries in photonuclear isotope list izn
c          m1c is the material index
c          m2c is the first isotope index for the material
c          m3c is the number of isotope entries for the material
c          only save entries in this material space (ie check index)
c
      if(nwc.le.m3c) izn(lizn+m2c+nwc-1)=iitm
      return
*/
*/ ********************************************************************
*/  Changes to oldcrd
*ident ocPN
*/ ********************************************************************
*/
*d,oc.16
     2      9910,9920,9930, 780)ica-55
*d,oc4a.14,oc4a.15
      do 395 m=jmd(ljmd+nmat),jmd(ljmd+nmat+1)-2
      do 395 i=m+1,jmd(ljmd+nmat+1)-1
*i,oc4b.37
c
c >>>>>  photonuclear isotope override                              mpn
c
c       if m1c is 0, there was no previously found corresponding m card
c          and a warning was printed from routine newcrd
c
```

```
  780 if(m1c.eq.0) return
c
c       if the number of mpn entries (nwc) was not equal to the number
c         of m entries (m3c), reset the photonuclear isotopes (izn) to
c         the corresonding material isotopes (iza) and print a warning
c
      if(nwc.ne.m3c)then
         do 790 i=m2c,m2c+m3c-1
            izn(lizn+i)=iza(liza+i)
  790    continue
         call erprnt(2,1,3,icn,m3c,nwc,0,0,
     1        '30hwrong number of entries on mpn,i4,8h wanted ,i3,'//
     2        '7h found ,i3')
c
c       otherwise print warnings about the isotope overrides
c
      else
         do 800 i=m2c,m2c+m3c-1
            if(izn(lizn+i).ne.iza(liza+i))
     1           call erprnt(1,2,3,icn,izn(lizn+i),iza(liza+i),0,0,
     2           '1hm, i5,28h:photonuclear event sees ZA=, i6,'//
     3           '16h in place of ZA=, i6')
  800    continue
      endif
      return
*/
*/ ********************************************************************
*/  Changes to iwtwnd
*ident iwPN
*/ ********************************************************************
*/
*d,iw4b.16
   15 b=b+abs(wwf(lwwf+i+mxa*(j-1+mww(ip))))
*/
*/ ********************************************************************
*/  Changes to stuff
*ident stPN
*/ ********************************************************************
*/
*d,st.49,st.109
c
c *******************************************************************
c        set up the list of cross-section tables needed by the problem.
c
c *****************************************
c     first add the tables by particle type
c
      mn=1
      do 240 km=1,mix
         if (km.ge.jmd(ljmd+mn+1)) mn=mn+1
         do 230 m=1,mipt
c
c        ****************************************************
c        this section determines if the tabular data is needed
c          (it should eventually be rewritten as a sequence
c           of boolean function calls, e.g. needElectronTbl())
c
c           if not transporting this particle, do not load its tables
c             special case: electron tables are needed if photon
c             thick-target bremsstrahlung production is on
            if (kpt(m).eq.0.and.
     1           (m.ne.3.or.ides.ne.0.or.kpt(2).eq.0)) go to 230
c
c           make sure that this material and table are really needed
            do 120 i=1,mxa
               if(mat(lmat+i).eq.mn.and.(fim(lfim+m,i).ne.0..or.m.eq.3
     1              .and.kpt(3).eq.0.and.fim(lfim+2,i).ne.0))go to 150
 120        continue
            do 140 i=1,nmfm,2
               if(mfm(lmfm+i).ne.nmt(lnmt+mn))go to 140
               do 130 j=1,ntal
               if(jptal(ljpt+1,j).eq.mfm(lmfm+i+1).and.
     1                  ktp(lktp+m,j).ne.0)go to 150
 130        continue
            do 135 j=1,npert
               k=mod(iptb(lipb+2,j)/2**(m-1),2)
               if(iptb(lipb+1,j).eq.-mfm(lmfm+i+1)
     1                  .and.k.ne.0)go to 150
 135        continue
 140      continue
c       ignore table if not needed
          go to 230
c
c       ******************************************************
c       if tabular data is requested, request a reasonable table
c       tabular data only exist for some particle types
c       currently only load tables for neutrons, photons,
c         electrons & protons
c
c          ****************************
c          identify the requested table
 150      l=kmm(lkmm+km)
          write(ht(1:7),'(i6,1h.)')iza(liza+km)
          ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1           //char(l/65536)
c
c          *******************
c          check neutron table
          if (m.eq.1) then
c
             if (.not.(ht(8:10).ne.' '.and.
     1                   index(' cdym',ht(10:10)).ne.0)) then
c               if not explicitly a neutron table, use the default.
                l=lxd(llxd+1,mn)
                ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1               //char(l/65536)
             endif
c
             if (mcal.ne.0) then
c               currently only allow type 'm' multigroup tables
                ht(10:10)='m'
             else
c
                if (ht(10:10).eq.'y') then
c                  ignore during this check sequence
c                  routine chekit prevents 'y' table use in a cell
                   continue
c
                else
c                  reset invalid tables to default
                   if (index('cd',ht(10:10)).eq.0) ht(10:10)='c'
c
c                  process requests for discrete tables
                   if (kdr(lkdr+1).lt.0) then
                      ht(10:10)='d'
                   else
                      do 170 i=1,mxe1
                         if(kdr(lkdr+i).eq.iza(liza+km))ht(10:10)='d'
 170                  continue
                   endif
                endif
             endif
c          end of check neutron table
c          **********************
c          check photoatomic table
          elseif (m.eq.2) then
             ht(4:6)='000'
c
             if (index('pg',ht(10:10)).eq.0) then
c               if not explicitly a photoatomic table, use a default.
                l=lxd(llxd+2,mn)
                ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1                 //char(l/65536)
              endif
c
c             force correct table type for the problem type
c             should add a warning to user if override request
              if (mcal.ne.0) then
                 ht(10:10)='g'
              else
                 ht(10:10)='p'
              endif
c           end of check photoatomic table
c           *******************
c           check electron table
            elseif (m.eq.3) then
              ht(4:6)='000'
c
              if(index('e',ht(10:10)).eq.0) then
c                if not explicitly an electron table, use a default.
                 l=lxd(llxd+3,mn)
                 ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1                 //char(l/65536)
              endif
c
c             force correct table type for the problem type
c             should add a warning to user if override request
c             NOTE: electron multigroup problems masquerade as neutron
c               problems using mode n and type 'm' tables; therefore
c               expire if multigroup problem reqeusts electron table
              if (mcal.ne.0) then
                 call expire (0,'stuff',
     1                 'multigroup electron problems must be run as '//
     2                 'neutron problem (see Manual)')
              else
                 ht(10:10)='e'
              endif
c           end of check electron table
c           *************************************************
            else
              go to 230
            endif
c
c           ***************************************************
c           add the table to the list if it is not already there
            do 210 i=1,mxe
              call zaid(2,hs,ixl(lixl+1,i))
              if (hs.eq.ht) then
                 lme(llme+m,km)=i
c                use fortran90 do-exit construct when available
              endif
 210        continue
c
            if (lme(llme+m,km).eq.0) then
              mxe=mxe+1
              if(mxe.gt.mxe1)call erprnt(1,1,0,0,0,0,0,0,
     1              '33hmaterial (mxe) overflow in stuff.')
              call zaid(1,ht,ixl(lixl+1,mxe))
              lme(llme+m,km)=mxe
            endif
c
c         *******************************
c         end of loop over particle types
 230      continue
c      end of loop over material entries
 240   continue
c
c **********************************
c      now add the supplemental tables
c
c         ************************************************
c         add the photonuclear table names to the master list
c         note that photon transport is only photoatomic if ispn.eq.0
```

```
      if (ispn.ne.0) then
         mn=1
         do 247 km=1,mix
            if (km.ge.jmd(ljmd+mn+1)) mn=mn+1
c
            if (izn(lizn+km).ne.0) then
               write(ht(1:7),'(i6,1h.)')izn(lizn+km)
               l=kmm(lkmm+km)
               ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1              //char(l/65536)
c
               if (index('u',ht(10:10)).eq.0) then
c                 if not explicitly a photonuclear table, use a default.
                  l=lxn(llxn+mn)
                  ht(8:10)=char(mod(l,256))//char(mod(l/256,256))
     1                 //char(l/65536)
               endif
c
               do 244 i=1,mxe
                  call zaid(2,hs,ixl(lixl+1,i))
                  if (hs.eq.ht) then
                     lmn(llmn+km)=i
c                    use fortran90 do-exit construct when available and
                  endif
 244           continue
c
               if (lmn(llmn+km).eq.0) then
                  mxe=mxe+1
                  if(mxe.gt.mxe1)call erprnt(1,1,0,0,0,0,0,0,
     1                 '33hmaterial (mxe) overflow in stuff.')
                  call zaid(1,ht,ixl(lixl+1,i))
                  lmn(llmn+km)=mxe
               endif
c
            endif
 247     continue
         endif
c
c        ****************************************************
c        add the thermal s(a,b) table names to the master list
         if (indt.ne.0) then
            if (mcal.ne.0) then
               call expire (0,'stuff',
     1             'cannot use thermal tables in a multigroup problem')
            else
               do 260 km=1,indt
                  call zaid(2,ht,kmt(lkmt+1,km))
c
c                 force correct table type for thermal table
c                 should add a warning to user if override request
                  ht(10:10)='t'
c
                  m=0
                  do 250 i=1,mxe
                     call zaid(2,hs,ixl(lixl+1,i))
                     if (hs.eq.ht) then
                        m=i
c                       use fortran90 do-exit construct when available
                     endif
 250              continue
c
c                 add the table to the master list if it is new
                  if (m.eq.0) then
                     mxe=mxe+1
                     if (mxe.gt.mxe1) call erprnt(1,1,0,0,0,0,0,0,
     1                    '33hmaterial (mxe) overflow in stuff.')
                     call zaid(1,ht,ixl(lixl+1,mxe))
                  endif
 260           continue
            endif
         endif
```

```
c
c ********************************************************************
*d,st.118
      mt=index(htn,hs(10:10))
*d,st.120
      nt=index(htn,ht(10:10))
*i,st.132
      if(ispn.eq.0)go to 300
      do 295 i=1,mix
      l=lmn(llmn+i)
      if(l.eq.ie)lmn(llmn+i)=je
  295 if(l.eq.je)lmn(llmn+i)=ie
*/
*/ ******************************************************************
*/  Changes to ixsdir
*ident ixPN
*/ ******************************************************************
*/
*d,ix.20
      nt=index(htn,ha)
*d,cor4-2.130
      call expire(0,'ixsdir',
     1 'cannot continue without valid xsdir file')
*d,ix4b.53
  290 nty(lnty+je)=index(htn,ha)
*i,ix4b.95
      do 435 m=1,mix
      if(lmn(llmn+m).eq.je)lmn(llmn+m)=ie
  435 if(lmn(llmn+m).gt.je)lmn(llmn+m)=lmn(llmn+m)-1
*i,ix.205
      call expire(0,'ixsdir',
     1 'cannot continue with missing cross-section table(s).')
*/
*/ ******************************************************************
*/  Changes to xact
*ident xaPN
*/ ******************************************************************
*/
*d,xa.11
   10 if(nty(lnty+i).eq.9)nt=nt+1
*/
*/ ******************************************************************
*/  Changes to getxst
*ident gtPN
*/ ******************************************************************
*/
*d,gt.102
      go to(140,140,140,200,290,310,310,301,360)nty(lnty+iex)
*d,gt.106
  140 if(nty(lnty+iex).ne.3)go to 145
c
c        if secondary particle information exists, set up ixs
      if(nxs(lnxs+7,iex).eq.0)go to 145
c
c        if nxs(7).gt.maxsec, exit to avoid memory error
      if(nxs(lnxs+7,iex).gt.maxsec)call expire(0,'getxst',
     1 'nxs(7) greater than maxsec for table '//ht(1:10)//'.')
c
c         load ixs array.
      do 142 i=1,10
        do 142 j=1,nxs(lnxs+7,iex)
  142     ixs(lixs+i,j,iex)=nint(xss(jxs(ljxs+32,iex)+i+10*(j-1)-1))
c
c         change locators by lp.
      do 143 i=1,10
        do 143 j=1,nxs(lnxs+7,iex)
  143     if(ixs(lixs+i,j,iex).ne.0)
     1       ixs(lixs+i,j,iex)=ixs(lixs+i,j,iex)+lp
c
c         find parameter for dosimetry table
  145 em=max(em,zero+xss(jxs(ljxs+1,iex)))
```

```
c
c         remove unneeded data from table
*d,gt.145
  200 esa(lesa+iex)=xss(jxs(ljxs+1,iex)+nint(xss(jxs(ljxs+1,iex))))
*d,gt.147
     1 min(zero+xss(jxs(ljxs+4,iex)+nint(xss(jxs(ljxs+4,iex)))),
*i,gt.185
c
c >>>>>  photonuclear table.
c
c        if secondary particle information exists, set up ixs
  301 if(nxs(lnxs+5,iex).eq.0)go to 306
c
c        if nxs(5).gt.maxsec, exit to avoid memory error
      if(nxs(lnxs+5,iex).gt.maxsec)call expire(0,'getxst',
     1 'nxs(5) greater than maxsec for table '//ht(1:10)//'.')
c
c         load ixs array.
      do 303 i=1,mixs
        do 303 j=1,nxs(lnxs+5,iex)
  303     ixs(lixs+i,j,iex)=nint(xss(jxs(ljxs+10,iex)+i+mixs*(j-1)-1))
c
c         change locators by lp.
      do 304 i=3,mixs
        do 304 j=1,nxs(lnxs+5,iex)
  304     if(ixs(lixs+i,j,iex).ne.0)
     1      ixs(lixs+i,j,iex)=ixs(lixs+i,j,iex)+lp
c
c        find minimum energy for each material
  306 do 307 i=1,nmat
        do 307 j=jmd(ljmd+i),jmd(ljmd+i+1)-1
  307     if(iex.eq.lmn(llmn+j))
     1      pnt(lpnt+i)=min(pnt(lpnt+i),xss(jxs(ljxs+1,iex)))
c
c        remove unneeded data from table
      call expgpn
c
c        print table information
      write(iuo,309)ht,nxs(lnxs+1,iex),hk,hm,hd
  309 format(1x,a10,i8,2x,a70,4x,a10,4x,a10)
      go to 380
*i,gt.218
c
      if(nty(lnty+iex).eq.1.or.nty(lnty+iex).eq.2)then
         ms=1
         me=10
         ns=nxs(lnxs+7,iex)
      else if(nty(lnty+iex).eq.8)then
         ms=3
         me=12
         ns=nxs(lnxs+5,iex)
      else
         ns=0
      endif
      do 395 j=1,ns
      do 395 i=ms,me
  395 if(ixs(lixs+i,j,iex).ne.0)ixs(lixs+i,j,iex)=ixs(lixs+i,j,iex)+lxs
*/
*/ ******************************************************************
*/  Changes to sread
*ident srPN
*/ ******************************************************************
*/
*d,sr.33,sr.34
      if(nty(lnty+iex).ne.9)read(iux,70,err=200)(xss(lp+i),i=1,ly(3))
      if(nty(lnty+iex).eq.9)read(iux,70,err=200)(exs(lp+i),i=1,ly(3))
*d,sr.47,sr.48
      if(k.ne.9)read(iux,rec=ly(2)+i,err=200)(xss(j),j=j1,j2)
   90 if(k.eq.9)read(iux,rec=ly(2)+i,err=200)(exs(j),j=j1,j2)
*/
*/ ******************************************************************
*/  Changes to utask
*ident utPN
*/ **********************************************************************
*/
*d,ut.40,ut.43
      kpan=kpac+mk*mipt*10*mxa+ktask*3*8*npn
      kpcc=kpan+mk*3*8*npn+ktask*3*mxa*kpt(2)
      kpwb=kpcc+mk*3*mxa*kpt(2)+ktask*mipt*21*mxa
      kwns=kpwb+mk*mipt*21*mxa+ktask*(mxxs/2)
*/
*/ **********************************************************************
*/  Changes to vtask
*ident vtPN
*/ **********************************************************************
*/
*d,vt.13
      do 30 j=1,17
*d,vt.81,vt.82
      do 210 j=1,8
      do 210 k=1,3
*d,vt.90
      do 230 j=1,21
*/
*/ **********************************************************************
*/  Changes to msgcon
*ident mePN
*/ **********************************************************************
*/
*d,me4a.2
*i,me4a.4
*if def,multp
*i,me4b.526
*endif
*d,me4a.584
*/
*/ **********************************************************************
*/  Changes to hstory
*ident hsPN
*/ **********************************************************************
*/
*i,hs.223
      if(nter.eq.17)tmavtc(2,2)=tmavtc(2,2)+tme*wgt
*/
*/ **********************************************************************
*/  Changes to dxtran
*ident dxPN
*/ **********************************************************************
*/
*d,dx4a.2,dx.25
      go to(        130, 50, 80,130,130,130,130,130,
     1      130,130,130, 80, 80, 80,130) ipsc-2
      go to(130,130, 90, 80,100,130) ipsc-100
      call expirx(1,'dxtran','illegal value for ipsc.')
      return
*i,dx4a.4
c        ipsc=16 -- neutron from law 61 (tabulated energies / angles)and
*i,dx.73
c        *****************************************************
*d,dx.124
      vel=slite*sqrt(erg*(erg+2.*gpt(ipt)))/(erg+gpt(ipt))
*/
*/ **********************************************************************
*/  Changes to acegam
*ident agPN
*/ **********************************************************************
*/
*d,ag.113
  200 if(ixre.eq.nint(xss(jxs(ljxs+32,iex)+2*ik-2)))go to 210
*d,ag.116,ag.117
  220 l=jxs(ljxs+15,iex)+nint(xss(jxs(ljxs+14,iex)+ixre-1))+1
      if(nint(xss(l-2)).eq.13)go to 250
```

```
*d,ag.123
      ix=nint(xss(l-1))
*d,ag.129,ag4b.12
      if(ix.gt.0)is=jxs(ljxs+7,iex)+nint(xss(jxs(ljxs+6,iex)+ix-1))
      ic=min(ktc(kktc+1,iex)-nint(xss(is-1))+1,nint(xss(is)))
*d,ag.132
      ic=min(ic+1,nint(xss(is)))
*d,ag4b.14
  250 ic=ktc(kktc+1,iex)-nint(xss(l-1))+1
*d,ag.141
      if(ic.gt.nint(xss(l)))go to 290
*d,ag.144
  260 if(ic.ge.nint(xss(l)))go to 290
*d,ag.166
  310 ixre=nint(xss(jxs(ljxs+32,iex)+ik*2-2))
*d,ag.186
      mtp=nint(xss(jxs(ljxs+13,iex)+ixre-1))
      ia=jxs(ljxs+17,iex)
      if(jxs(ljxs+16,iex).ne.0)then
        ka=nint(xss(jxs(ljxs+16,iex)+ixre-1))
      else
        ka=0
      endif
      id=jxs(ljxs+19,iex)
      kd=nint(xss(jxs(ljxs+18,iex)+ixre-1))
      call acecas(1,1,zero,ia,ka,id,kd)
*d,ag.188
      if(ixcos.ne.0)ipsc=8
c
c        if photon energy below cell cutoff, ignore it.
      if(colout(1,1).lt.elc(2))go to 450
*d,ag.191,ag.192
      if(nint(xss(jxs(ljxs+13,iex)+ixre-1)).lt.18000)go to 390
      if(nint(xss(jxs(ljxs+13,iex)+ixre-1)).gt.19999)go to 390
*d,ag.221,ag.223
      pan(kpan+1,6,kp)=pan(kpan+1,6,kp)+1.
      pan(kpan+1,7,kp)=pan(kpan+1,7,kp)+wgt
      pan(kpan+1,8,kp)=pan(kpan+1,8,kp)+wgt*erg
*/
*/ ********************************************************************
*/  Changes to acecol
*ident acPN
*/ ********************************************************************
*/
*d,ac.25
      j=l+2+ktc(kktc+2,iex)-nint(xss(l))
*d,ac.46,ac.47
      c=acecos(ixcos,jxs(ljxs+9,iex),nint(xss(jxs(ljxs+8,iex))))
*d,ac.65
      j=l+2+ktc(kktc+2,iex)-nint(xss(l))
*d,ac4b.2
      if(jq.ne.2.eqv.nint(xss(jxs(ljxs+5,iex)+ixre-1)).eq.19)go to 120
*d,ac.75,ac.77
  110 is=jxs(ljxs+7,iex)+nint(xss(jxs(ljxs+6,iex)+ixre-1))
      ic=ktc(kktc+2,iex)+1-nint(xss(is-1))
      if(ic.lt.1.or.ic.gt.nint(xss(is)).or.ic.eq.nint(xss(is)).and.
*d,ac.105
  150 ntyn=nint(xss(jxs(ljxs+5,iex)+ixre-1))
*i,ac4b.4
      q=xss(jxs(ljxs+4,iex)+ixre-1)
      ia=jxs(ljxs+9,iex)
      ka=nint(xss(jxs(ljxs+8,iex)+ixre))
      id=jxs(ljxs+11,iex)
      kd=nint(xss(jxs(ljxs+10,iex)+ixre-1))
*d,ac.109,ac.111
      l=jxs(ljxs+11,iex)+2+nint(xss(jxs(ljxs+10,iex)+ixre-1))
      ie=2*nint(xss(l))+l
      if(erg.le.xss(ie+2).or.erg.ge.xss(ie+1+nint(xss(ie+1))))go to 180
*d,ac.119
      call acecas(i,1,q,ia,ka,id,kd)
*d,ac4a.15
```

```
      call acecas(i,1,q,ia,ka,id,kd)
*d,ac4a.59
      call acecas(1,1,q,ia,ka,id,kd)
*/
*/ ********************************************************************
*/  Changes to acecas
*ident asPN
*/ ********************************************************************
*/
*d,as.2,as.5
      subroutine acecas(ls,ip,q,ia,ka,id,kd)
c        sample the emission parameters from the appropriate law data.
c
c        the subroutine takes in all the information necessary to
c        sample (or debug inability to sample) the emission
c        energy and scattering angle in the laboratory coordinate
c        system.
c
c     Preconditions:
c
c        explicitly passed in variables:
c        ls - the current particle index in the colout array
c        ip - the ipt particle type for the incident particle
c         q - the q value for the reaction being sampled
c        ia - the first word of the relevant AND block in the xss array
c        ka - the offset to the first word of the table in AND block
c        id - the first word of the relevant DLW block in the xss array
c        kd - the offset to the first word of the table in DLW block
c
c        implicitly uses variables:
c        awn(lawn+iex) - the awr of the current target isotope
c        colout(1,ls) - the sampled emission energy
c        colout(2,ls) - the sampled emission scattering angle
c        erg - the incident particle energy in the lab system
c        gpt - array of particle awr's
c        iex - the table index of the current target isotope
c        ipsc - index to indicate what kind of law was sampled
c        ipt - the ipt particle type of the emitted particle
c        ixl - the ZAID storage array
c        ixre - the current reaction index being sampled
c        kdb - fatal error flag
c        mtp - the reaction mt number being sampled
c        ntyn - the coordinate system of the sampled parameters
c        tpd(1,2) - storage for correlated energy/angle parameters
c        wgt - the weight of the emitted particle
c        xss - data array containing sampling distributions etc.
c
c        read only input variables:
c          ls, ip, q, ia, ka, id, kd, awn, erg, gpt, iex, ipt, ixl,
c          ixre, mtp, ntyn and all xss(i)
c
c     Postconditions:
c
c        returns the sample emission parameters in the lab system:
c          colout(1,ls) - the emission energy
c          colout(2,ls) - the emission scattering angle
c
c        Law 4/44/61 makes use of a biased distribution which affects
c        the outgoing particle weight
c
c        kdb is set on fatal error (inability to sample reasonable
c        emission parameters) during call to expirx
c
c        modified variables: colout(1,ls), colout(2,ls), ipsc, kdb,
c          tpd and wgt
c
*i,as.6
      parameter (ep=0.000001)
*d,as4a.1,as4a.3
*i,as.8
c*******************************************************************
```

```
*d,as.12,as.15
      n=id+kd
      go to 25
   20 n=id+nint(xss(n-1))
   25 if(nint(xss(n-1)).eq.0)go to 30
      t1=t1-acefcn(n+2,erg,ln)
*d,as.18
c*********************************************************************
c        use the selected law to sample the energy [and possibly angle].
c        if law samples without error, go to sample angle or
c        coordinate transform as appropriate
*d,as.20,as4a.9
      lw=nint(xss(n))
      iw=id-1+nint(xss(n+1))
      if(lw.eq.1) go to 40
      if(lw.eq.2) go to 60
      if(lw.eq.3) go to 70
      if(lw.eq.4) go to 80
      if(lw.eq.5) go to 160
      if(lw.eq.7) go to 170
      if(lw.eq.9) go to 190
      if(lw.eq.11)go to 210
      if(lw.eq.22)go to 230
      if(lw.eq.24)go to 242
      if(lw.eq.33)go to 70
      if(lw.eq.44)go to 80
      if(lw.eq.61)go to 80
      if(lw.eq.66)go to 245
      if(lw.eq.67)go to 255
      go to 300
*d,as.28,as.31
c >>>>>  law 1 (From ENDF Law 1) -- tabular equi-probable energy bins.
   40 call acetbl(iw,ic,r,ln)
      nt=nint(xss(iw+ln))
      iw=iw+ln+nt*(ic-1)
      k=int(rang()*(nt-1)+1)
*d,as.46,as.48
c        this law applies to photon production only and should
c        not be used by any other emission particle type
   60 if(ipt.ne.2)go to 300
      colout(1,ls)=xss(iw+1)
      if(nint(xss(iw)).eq.2)colout(1,ls)=colout(1,ls)+
     1 erg*awn(lawn+iex)/(awn(lawn+iex)+1.)
      go to 260
*d,as.50
c >>>>>  law 3 & 33 (From ENDF Law 3) -- level scattering.
*d,as.54,as4a.11
c >>>>>  law 4 (From ENDF Law 1) -- continuous erg tabular distribution
c >>>>>  law 44 (From ENDF Law 1) -- Kalbach-87 correlated formalism
c >>>>>  law 61 (From ENDF Law 1) -- correlated tab energy-angle dist
   80 call acetbl(iw,ic,r,ln)
      nr=nint(xss(iw))
      lb=iw-1+ln+ic
      lc=id+nint(xss(lb))
*d,as4a.14
c
c        xss(lc,ld,lf) is an overloaded variable.  it contains the
c        number of points in the emission distribution and if part
c        of the continuum has been expunged, it contains 0.5 times
c        the cumulative probability of the portion expunged.
      np=int(xss(lc)+ep)
*d,as4a.16
      jj=nint(xss(lc-1))
*d,as4a.19
*d,as4a.21,as4a.22
      ld=id+nint(xss(lb+1))
      mp=int(xss(ld)+ep)
*d,as4a.28
      jj=nint(xss(ld-1))
*d,as4a.53
   97 np=int(xss(lf)+ep)
```

```
*d,as4a.58
      ns=nint(xss(lf-1))-jj*10000
*d,as.80,as.89
      call bnsrch(r1,ic,ib,ig)
      ln=ic-2*np
      fa=xss(ln+np)
      ea=xss(ln)
*d,as.93
      bb=(xss(ln+np+1)-fa)/(xss(ln+1)-ea)
*d,as4a.64,as4a.65
      if(lw.ne.44)go to 150
      fb=(t-xss(ln))/(xss(ln+1)-xss(ln))
*d,as4a.68
      if(lw.ne.44)go to 150
*d,as4a.72,as4a.74
*d,as4a.75,as4a.78
c
c        don't scale for photons?? why not?!? this should be removed
c        but is left in here in order to track the test suite
c        it only affects test problem 11 (mcw 1/7/99)
  150 if(r.ne.0..and.ipt.ne.2)t=t1+(t-xss(lf+nd+1))*(t2-t1)
     1 /(xss(lf+np)-xss(lf+nd+1))
  152 colout(1,ls)=t
      if(lw.eq.4)go to 260
      if(lw.eq.44)go to 153
      if(lw.eq.61)go to 156
      go to 295
*d,as4a.80
c        sample law 44 -- kalbach-87 angular systematics
*d,as4a.82
  153 if(ka.ne.-1.or.ntyn.ge.0)go to 295
*i,as4a.91
c
c        sample law 61 -- tabulated angular distribution
  156 ipsc=16
c
c        if jj=1 (i.e., histogram on e-primes) always use "ic."
c        if jj=2 (lin-lin on e-primes), use the distribution for
c        the e-prime closest to "r1" (in cdf space).
      lb=ic+np
      if(jj.ne.1.and.(xss(ib)-r1.lt.r1-xss(ic)))lb=lb+1
c
c        sample from appropriate distribution
c        unlike the AND block, in law 61 only isotropic or tabular
c        angular information is passed
      if(nint(xss(lb)).gt.0)go to 157
      ixcos=0
      call angiso(colout(2,ls))
      go to 280
  157 ixcos=id-1+nint(xss(lb))
      call anglw2(ixcos,colout(2,ls))
      ixcos=-ixcos
      go to 280
*d,as.112,as.113
c >>>>>  law 5 (From ENDF Law 5) -- general evaporation spectrum.
  160 t1=acefcn(iw,erg,ln)
      i=iw+ln+1+int(rang()*(nint(xss(iw+ln))-1))
*d,as.117,as.118
c >>>>>  law 7 (From ENDF Law 7) -- simple Maxwell fission spectrum.
  170 t1=acefcn(iw,erg,ln)
      t3=erg-xss(iw+ln)
*d,as.129,as.130
c >>>>>  law 9 (From ENDF Law 9) -- evaporation spectrum.
  190 t1=acefcn(iw,erg,ln)
      t2=erg-xss(iw+ln)
*i,as.133
c        reject if outside range 0 ... e-u
*d,as.137,as.139
c >>>>>  law 11 (From ENDF Law 11) -- energy dependent Watt spectrum.
  210 t1=acefcn(iw,erg,ln)
      t2=acefcn(iw+ln,erg,lb)
      if(erg.le.xss(iw+ln+lb))go to 260
*d,as.146,as4a.93
c >>>>>  law 22 (From UK Law 2) -- tabular linear functions.
  230 call acetbl(iw,ic,r,ln)
      ie=id-1+nint(xss(iw+ln+ic-1))
      nf=nint(xss(ie))
*d,as.156,as4a.97
c >>>>>  law 24 (From UK Law 6) -- tabular energy multipliers.
  242 call acetbl(iw,ic,r,ln)
      i=iw+ln+1+nint(xss(iw+ln))*(ic-1)+int(rang()*(nint(xss(iw+ln))-1))
*d,as4a.100,as4a.101
c >>>>>  law 66 (From ENDF Law 6) -- n-body phase space distribution.
  245 nb=nint(xss(iw))
*i,as4a.102
      if(ipt.gt.2)ap=ap*gpt(1)/gpt(ipt)
*d,as4a.117,as4a.118
      colout(1,ls)=t*((ap-1.)/ap)*(erg*aw/(aw+1.)+q)
*d,as4a.126,as4a.127
  255 call acetbl(iw,ic,r,ln)
      cs=acecos(ixcos,ia,ka)
*d,as4a.130
      colout(1,ls)=acecs6(0,id,iw,ic,r,cs)
*d,as4b.3,as.180
c*******************************************************************
c        if not correlated energy-angle, calculate the cosine.
  260 colout(2,ls)=acecos(ixcos,ia,ka)
*i,as.181
c*******************************************************************
*d,as.185,as.189
c        Formulas below are from p. 2 of X-6:RES-93-68.
c        These formulas assume two-body kinematics.
c        Seamon's formulas specify atomic weight ratios (to neutron)
c        For incident particle (a), AWR=GPT(IPT_INCIDENT)/GPT(1)
c        For exiting particle (b), AWR=GPT(IPT)/GPT(1)
c        For target (A), AWR=AWN(LAWN+IEX)
      a1=gpt(ip)/gpt(1)
      a2=gpt(ipt)/gpt(1)
      a3=awn(lawn+iex)
      t1=colout(1,ls)
      t2=erg*a1*a2/(a3+a1)**2
      t3=2.*sqrt(a2*a1*erg*colout(1,ls))*colout(2,ls)/(a3+a1)
      t4=t1+t2+t3
      s1=colout(2,ls)*sqrt(colout(1,ls)/t4)
      s2=sqrt(a1*a2*erg/t4)/(a3+a1)
      colout(1,ls)=t4
      colout(2,ls)=s1+s2
*i,as4a.137
c*******************************************************************
*d,as4a.139
  290 if(colout(1,ls).le.emx(ipt))return
*d,as4a.142
     1 'energy of particle from inelastic collision > emx')
*i,as.191
c*******************************************************************
*d,as4a.145
  295 colout(1,ls)=huge
*d,as.195,as.196
      write(iuo,310)ht,erg,ixre,mtp,ntyn,lw,colout(1,ls)
*d,as.203
     1 'emission energy was negative.')
      if(colout(1,ls).eq.huge)call expirx(1,'acecas',
     1 'faulty cross-section data.')
*d,as.205
     1 'emission energy exceeds incident energy.')
*d,as.207,as.216
*/
*/ *******************************************************************
*/  Changes to acefcn
*ident aiPN
*/ *******************************************************************
*/
```

```
*d,ai.2,ai.54
      function acefcn(it,eg,ln)
c        interpolate value in table at xss(it) for energy eg
c
c        Preconditions:
c          xss(it) is the first word of an appropriate table
c          table data are in the format:
c            nr - number of regions
c            nbt(i=1..nr) - number of points in int. region i
c            int(i=1..nr) - interpolation scheme for region i
c            (nbt and int don't exist if nr is zero and a
c             linear-linear int. scheme is used across all points)
c            nf - number of energy points
c            e(j=1..nf) - energy values
c            f(j=1..nf) - function values on which to interpolate
c          the interpolation values are picked using energy value eg
c
c          it, eg and all xss(i) are read only variables
c
c        Postconditions:
c          returns value f(eg) appropriately interpolated
c          return value ln is the length of the table,
c            i.e. xss(it+ln-1) = f(ne)
c
c          if eg is not within the bounds of the table e(i)
c            i.e. bnsrch returns with a non-zero value of ig
c            the extreme edge boundary value is returned
c            e.g. erg < e(1)  ->  acefcn = f(1)
c
c          ln and acefcn are modified return values
c
*call cm
c
c        get key parameter values for table
      nr=nint(xss(it))
      ie=it+2*nr+1
      nf=nint(xss(ie))
      ln=2*(nr+nf+1)
c
c        binary search for the location of eg in the table.
      il=ie+1
      ih=ie+nf
      call bnsrch(eg,il,ih,ig)
c
c        set up the parameters for interpolation
      ea=xss(il)
      eb=xss(ih)
      fa=xss(il+nf)
      fb=xss(ih+nf)
c
c        if outside bounds of table, use extreme edge value
      if(ig.ne.0)go to 30
c
c        find out which kind of interpolation should be used.
      if(nr.eq.0)go to 40
      do 10 n=1,nr
   10 if(ih-ie.le.nint(xss(it+n)))go to 20
      n=nr
c
c        interpolate between table entries.
   20 go to(30,40,50,60,70)nint(xss(it+nr+n))
c
c        histogram interpolation
   30 acefcn=fa
      go to 80
c
c        linear-linear interpolation
   40 acefcn=fa+(fb-fa)*(eg-ea)/(eb-ea)
      go to 80
c
c        log-linear interpolation
```

```
   50 acefcn=fa+(fb-fa)*log(eg/ea)/log(eb/ea)
      go to 80
c
c        linear-log interpolation
   60 acefcn=fa*(fb/fa)**((eg-ea)/(eb-ea))
      go to 80
c
c        log-log interpolation
   70 acefcn=fa*(fb/fa)**(log(eg/ea)/log(eb/ea))
      go to 80
c
   80 return
*/
*/ *****************************************************************
*/  Changes to acetbl
*ident abPN
*/ *****************************************************************
*/
*d,ab.2,ab.42
      subroutine acetbl(it,il,r,ln)
c        returns the interpolation parameters and table length
c
c        Preconditions:
c          xss(it) is the first word of an appropriate interpolated
c          energy region with data in the format:
c            nr - number of regions
c            nbt(i=1..nr) - number of points in int. region i
c            int(i=1..nr) - interpolation scheme for region i
c            (nbt and int don't exist if nr is zero and a
c             linear-linear int. scheme is used across all points)
c            nf - number of energy points
c            e(j=1..nf) - energy values
c          the interpolation values are picked using erg
c
c          it, erg and all xss(i) are read only variables
c
c        Postconditions:
c          return value il is the lower indice on which to interpolate
c          return value r is the interpolation factor such that
c            e(il) + r * ( e(ih) - e(il) )  =  erg
c            acetbl ignores interpolation schemes other than
c            linear-linear or histogram
c          return value ln is the length of the table,
c            i.e. xss(it+ln-1) = e(ne)
c
c          if erg is not within the bounds of the table e(i)
c            i.e. bnsrch returns with a non-zero value of ig
c            the extreme edge boundary is returned in il
c            and the interpolation factor is returned as zero
c            e.g. erg < e(1)  ->  ii = 1 and r = 0.0
c
c          il, r and ln are modified return values
c
*call cm
c
      nr=nint(xss(it))
      ie=it+2*nr+1
      nf=nint(xss(ie))
      ln=2*(nr+1)+nf
      r=0.
c
c        binary search for the location of the energy in the table.
      il=ie+1
      ih=ie+nf
      call bnsrch(erg,il,ih,ig)
      if(ig.ne.0)go to 40
c
c        calculate interpolation fraction r
c          use histogram interpolation if int(i) = 1
c          use linear-linear interpolation for all else
      if(nr.eq.0)go to 30
      do 10 n=1,nr
   10 if(ih-ie.le.nint(xss(it+n)))go to 20
      n=nr
   20 if(nint(xss(it+nr+n)).eq.1)go to 40
   30 if(erg-xss(il).lt.1.e-6*(xss(ih)-xss(il)))go to 40
      r=(erg-xss(il))/(xss(ih)-xss(il))
c
   40 il=il-ie
      return
*/
*/ *******************************************************************
*/  Changes to acecos
*ident aoPN
*/ *******************************************************************
*/
*d,ao.2,ao.39
      function acecos(it,ia,ka)
c        returns the scattering angle mu from a set of angular dist.
c
c        Preconditions:
c          xss(ia) is the first word of the appropriate AND block
c          if no AND block exists, ka must be set to zero
c          ka is the offset such that:
c            ka = 0 indicates an isotropic distribution OR
c            xss(ia+ka-1) is the first word of the appropriate dist.
c          the distribution always contains the following data:
c          nm, e(i=1..nm), lmu(i=1..nm), [mu data for non-zero lmu]
c          the table is picked using erg, the incident particle energy,
c          off the energy list e corresponding to the table offset lmu
c          lmu = 0 indicates isotropic distribution
c          lmu > 0 indicates Angular Law 1 binned data at xss(ia-1+lmu)
c          lmu < 0 indicates Angular Law 2 tabular data at xss(ia-1-lmu)
c
c          ia, ka, erg and all xss(i) are read only input variables
c
c        Postconditions:
c          return value acecos is scattering angle mu in radians
c          mu value should be between -1.0 and 1.0 but isn't checked
c          it is the offset to the first word of the table sampled
c          it = 0 indicates an isotropic distribution sampled
c          it > 0 indicates a binned distribution sampled
c          it < 0 indicates a tabular distribution sampled
c          non-isotropic distribution are found at xss(abs(it))
c
c          it and acecos are modified return values
c
*call cm
c
c        handle case of no table data
      if(ka.eq.0)go to 10
c
c        find the cosine table by binary search on the energy table.
      nm=nint(xss(ia+ka-1))
      il=ia+ka
      ih=il+nm-1
      call bnsrch(erg,il,ih,ig)
c
c        sample between adjoining tables by interpolation fraction.
      if(rang()*(xss(ih)-xss(il)).lt.erg-xss(il))il=ih
c
c        select appropriate sampling distribution
      lm=nint(xss(il+nm))
c
c >>>>>  Isotropic Angular Distribution
      if(lm.ne.0)go to 20
   10 call angiso(t)
      it=0
      go to 40
c
c >>>>>  Law 1 Equiprobable Binned Angular Distribution
   20 if(lm.lt.0)go to 30
```

```
      it=ia-1+lm
      call anglw1(it,t)
      go to 40
c
c >>>>>  Law 2 Tabular Probability Angular Distribution
   30 it=ia-1-lm
      call anglw2(it,t)
      it=-it
c
   40 acecos=t
      return
*/
*/ *********************************************************************
*/  Changes to acecs6
*ident aePN
*/ *********************************************************************
*/
*d,ae4a.2
      function acecs6(ii,id,iw,jc,r,cs)
*d,ae4a.10
      save id0,iw0,ic0,rr0,rnb
*i,ae4a.11
      id0=id
*d,ae4a.17,ae4a.19
   10 jw=iw0+2*nint(xss(iw0))+1
      ne=nint(xss(jw))
      lx(1)=id0+nint(xss(jw+ne+ic0))-1
*d,ae4a.24
      lx(2)=id0+nint(xss(jw+ne+ic0+1))-1
*d,ae4a.31,ae4a.32
      mu=nint(xss(le))
      nm=nint(xss(le+1))
*d,ae4a.44,ae4a.46
   50 lb=id0+nint(xss(le+nm+iq+1))
      jj=nint(xss(lb-1))
      np=nint(xss(lb))
*d,ae4a.54,ae4a.60
      call bnsrch(rnb(ir),ic,ib,ig)
*/
*/ *********************************************************************
*/  Changes to photot
*ident ptPN
*/ *********************************************************************
*/
*i,pt.9
c
c        if photonuclear physics is on, calculate photonuclear xs
      if (ispn.ne.0) call pnctot(mk)
*/
*/ *********************************************************************
*/  Changes to colidp
*ident cpPN
*/ *********************************************************************
*/
*d,cp.10,cp.11
*i,cp.12
c
c*********************  photonuclear events  ************************
c
c     if photonuclear physics is on and the photon is above the pn.
c        energy threshold for this material, totpn is non-zero
c        and photonuclear secondary particles are sampled first
      if (totpn.gt.zero) then
         call coldpn
         if (nter.ne.0.or.kdb.ne.0) return
      endif
c
c*********************  photoatomic events  *************************
c
c        reset default parameters
      ntyn=0
      jsu=0
*/
*/ *********************************************************************
*/  Changes to mgcoln
*ident gnPN
*/ *********************************************************************
*/
*d,gn4a.2,gn4a.3
      pan(kpan+1,6,mpan)=pan(kpan+1,6,mpan)+n
      pan(kpan+1,7,mpan)=pan(kpan+1,7,mpan)+wgt*n
*d,gn4a.6
      pan(kpan+1,8,mpan)=pan(kpan+1,8,mpan)+wgt*erg
*/
*/ *********************************************************************
*/  Changes to mgcolp
*ident gpPN
*/ *********************************************************************
*/
*d,gp4a.7
      pan(kpan+1,7,mpan)=pan(kpan+1,7,mpan)+wgt*(n-1)
*/
*/ *********************************************************************
*/  Changes to tallyd
*ident tdPN
*/ *********************************************************************
*/
*d,td4a.2,td4a.3
      go to(        110, 40, 70,110,110,110,110, 80,
     1      110,110,110, 70, 70, 70,110) ipsc-2
      go to(110,110,90,70,100,110) ipsc-100
      call expirx(1,'tallyd','illegal value for ipsc.')
      return
*i,td4a.5
c        ipsc=16 -- neutron from law 61 (tabulated energies/angles) and
*i,td.100
c     *******************************************
*/
*/ *********************************************************************
*/  Changes to calcps
*ident ctPN
*/ *********************************************************************
*/
*d,ct4a.1
      go to(         10, 20, 30,120,150,200,260,400,
     1      540,580,590,800, 25, 26,900) ipsc-2
      call expirx(1,'calcps','illegal value for ipsc.')
      return
*d,ct.18
      if(t3.le.0..and.tpd(1)**2.lt.abs(tpd(3)))go to 1000
*d,ct.20
      if(t1.lt.0.)go to 1000
*d,ct4a.5
      erg=acecs6(1,0,0,0,zero,cs)
*i,ct4a.7
c
c >>>>>  ipsc=16 -- neutron from law 61 (tabulated energies / angles)
c
c        if isotropic and lab system, can return (psc=0.5).
   26 if(ixcos.eq.0.and.ntyn.ge.0) return
c
c        cs is laboratory cosine to next event estimator.
      cs=uold(1)*uuu+uold(2)*vvv+uold(3)*www
c
c        if lab system, but anisotropic, go to table lookup for psc.
c        since we're in the lab system, erg for next-event direction
c        is no different than erg in actual as-sampled direction.
      if(ntyn.ge.0) goto 60
      a1=1.+awn(lawn+iex)
c
c        scattering distributions are in the cm system.
c
```

```
c        if pass the following test, then all lab cosines are possible.
c        if fail, then must check on allowed lab cosines.
c        check is from cases 2,3 on pp. 68,69 of carter and cashwell.
c        ergace is previously-sampled cm energy. eg0 is incident energy.
      if(ergace.gt.eg0/(a1**2)) goto 27
c
c        to find valid lab cosines, start with eq. in case 3 of p. 69
c        from carter and cashwell.  substitute in for q based on
c        their eq. 5.8.  then wind up with following condition,
c        which is identical to that used by hendricks later in
c        this routine for ipsc=14.  if t3 > cs**2, cannot scatter
c        toward next-event position.
      t3=1.-ergace*a1**2/eg0
      if(t3.gt.cs**2) goto 1000
c
c        scattering is valid.  calculate lab energy (erg) via following
c        equation, from hendricks' ipsc=14 code.  this equation is
c        equivalent to eq. 5.14 of carter and cashwell.
   27 t1=1./a1
      erg=eg0*(t1*(cs+sqrt(cs**2-t3)))**2
c
c      return if this energy is below particle's energy cutoff.
      if(erg.lt.elc(ipt)) return
c
c        now, we need to calculate the d-cm cosine by d-lab cosine
c        and apply this factor to the psc.  start with eq. 5.13 of
c        carter and cashwell, which reduces to the following for
c        psc (the extra 0.5 is a starting assumption of isotropy).
c        formalism is identical to ipsc=14 and ipsc=5.
      t2=sqrt(erg/ergace)
      t4=t1*sqrt(eg0/erg)
      psc=0.5*t2/(1.-cs*t4)
c
c        if isotropic in cm, return.
      if(ixcos.eq.0) return
c
c        otherwise, determine cm cosine (overwrite cs with it).  start
c        with eq. 5.10 of cashwell and carter.  formalism is same as
c        for ipsc=5 and ipsc=14.
      cs=t2*(cs-t4)
c
c        now (finally) go to table lookup to actually determine psc
c        for scattering toward the next-event estimator.
      goto 60
*d,ct.40
      if(t3.gt.0.)go to 1000
*d,ct.54
      if(cs.lt.0.)go to 1000
*i,ct.67
c
c        updates to allow sampling of tabular angular distributions
c        updates are for mcnpx_2.1.4
c        written by rcl (june, 1998)
c        changes to subroutine calcps:
c          in two sections of calcps we have allowed for the
c          possibility of calculating the psc by looking up the
c          value in a tabulated cosine distribution.  this is
c          an alternative to the "normal" manner of calculating
c          the psc from a table of 32 equally-likely cosine bins.
c          changes are made immediately below for neutron-induced
c          neutrons and in the ipsc=8 section for neutron-induced
c          photons.  new capability is flagged via ixcos<0.
c          factor of 2 in psc calculation below is to account for fact
c          that psc is initialized to 0.5 before calling calcps.
c          similar calculation for ipsc=8 does not need multiplier
c          because it does not have form of psc=psc*value.
c
c          also added a section for ipsc=16, which is for law=61
c          correlated tabular-energy, tabular cosine distributions.
c
   60 if(ixcos.gt.0) then
```

```
*d,ct.70,ct.79
            ic=ixcos
            ib=ixcos+32
   70       if(ib-ic.eq.1)go to 90
            ih=(ic+ib)/2
            if(cs.lt.xss(ih))go to 80
            ic=ih
            go to 70
   80       ib=ih
            go to 70
   90       psc=.0625*psc/(xss(ib)-xss(ic))
c
c        calculate psc from tabulated cosine distribution
      else
            ic=-ixcos
            jj=nint(xss(ic))
            np=nint(xss(ic+1))
            if(xss(ic+2).gt.cs.or.xss(ic+1+np).lt.cs) go to 1000
            ic=ic+3
   92       if(xss(ic).gt.cs) go to 94
            ic=ic+1
            go to 92
   94       if(jj.eq.1) then
                  psc=2.*psc*xss(ic+np-1)
            else
                  psc=2.*psc*(xss(ic+np-1)+(cs-xss(ic-1))*
     1            (xss(ic+np)-xss(ic+np-1))/(xss(ic)-xss(ic-1)))
            endif
      endif
*d,ct.85
      l=nint(xss(jxs(ljxs+1,iex)+nint(xss(jxs(ljxs+16,iex)))+ixcos-2))
*d,ct.87,ct.88
      j=jxs(ljxs+1,iex)+nint(xss(jxs(ljxs+17,iex)))+l-2
      if(cs.ge.xss(j+nxs(lnxs+3,iex)-1).or.cs.lt.xss(j))go to 1000
*d,ct4a.9
      if(t3.ge.vco(mcoh))go to 1000
*d,ct.128,ct.137
      if(ixcos.gt.0) then
            ic=ixcos
            ib=ic+32
  210       if(ib-ic.eq.1)go to 230
            ih=(ic+ib)/2
            if(cs.lt.xss(ih))go to 220
            ic=ih
            go to 210
  220       ib=ih
            go to 210
  230       psc=.03125/(xss(ib)-xss(ic))
      else
            ic=-ixcos
            np=nint(xss(ic))
            jj=nint(xss(ic+1))
            if(xss(ic+2).gt.cs.or.xss(ic+1+np).lt.cs) go to 1000
            ic=ic+3
  192       if(xss(ic).gt.cs) go to 194
            ic=ic+1
            go to 192
  194       if(jj.eq.1) then
                  psc=xss(ic+np-1)
            else
                  psc=xss(ic+np-1)+(cs-xss(ic-1))*
     1            (xss(ic+np)-xss(ic+np-1))/(xss(ic)-xss(ic-1))
            endif
      endif
*d,ct.141
  240 n=nint(xss(jxs(ljxs+15,iex)))
*d,ct.143
      l=nint(xss(jxs(ljxs+1,iex)+nint(xss(jxs(ljxs+16,iex)+1))+ixcos-2))
*d,ct.145,ct.146
      j=jxs(ljxs+1,iex)+nint(xss(jxs(ljxs+17,iex)+1))+l-2
      if(cs.ge.xss(j+n-1).or.cs.lt.xss(j))go to 1000
```

```
*d,ct.156
      if(ixcos.eq.0)go to 1000
*d,ct.160
      if(cs.lt.xss(-ixcos).or.cs.gt.xss(ll-ixcos))go to 1000
*d,ct.200
  350 nc=nint(xss(jxs(ljxs+4,iet)))
*d,ct.223
      if(cs*erg.lt.b0-db)go to 1000
*d,ct.243
      if(am.lt.q(6).or.am.gt.q(7))go to 1000
*d,ct.252
      go to 1000
*d,ct.303
      if(b.lt.0.)go to 1000
*d,ct.311
      if(r.gt.q(7)**2.or.r.lt.q(6)**2)go to 1000
*d,ct.322
      if(j.ne.0)go to 1000
*d,ct.336
      go to 1000
*d,ct4a.20
     1 go to 1000
*d,ct4b.32
  800 cs=uold(1)*uuu+uold(2)*vvv+uold(3)*www
*d,ct4a.31
      if(t3.ge.cs**2)go to 1000
*d,ct4a.41
      if(t5.le.0.)go to 1000
*i,ct4a.45
c
c >>>>>  ipsc=17 --  photonuclear kalbach-87 endf/b-vi coupled energy-
c        angle collision (law 44).
c        kalbach-87 psc=(.5*a/sinh(a))*(cosh(a*cs)+r*sinh(a*cs))
c        tpd(1)=r, tpd(2)=a
  900 cs=uold(1)*uuu+uold(2)*vvv+uold(3)*www
      psc=.5*tpd(2)/sinh(tpd(2))
     1      *(cosh(tpd(2)*cs)+tpd(1)*sinh(tpd(2)*cs))
      return
*d,ct.351
 1000 psc=0.
*/
*/ ********************************************************************
*/  Changes to eventp
*ident PN
*/ ********************************************************************
*/
*d,et4b.1,et4a.2
      character ha*10,he(8)*9,hf(3)*15,hg*48,hl*20,hp(0:5)*3,hs(27)*9,
     1 ht(17)*16,hz*10
*d,et.11
     3 'dead fiss. or pp',3*' ','photoabsorption'/
*d,et4b.2
     4 '(n,xf) mg','(n,xn) mg','(g,xg) mg','adj split','wwt split',
     5 2*' ','(gamma,x)'/
*/
*/ ********************************************************************
*/  Changes to kcalc
*ident kcPN
*/ ********************************************************************
*/
*d,kc4b.145
      do 260 j=1,17
*d,kc4b.150
      do 280 i=1,8
*d,kc4b.153
      do 290 i=1,21
*/
*/ ********************************************************************
*/  Changes to sumary
*ident eqPN
*/ ********************************************************************
```

```
*/
*d,eq4a.1
      character ha(2)*26,hg*16,hp(2,7,mipt)*17,ht(4)*14,hw(2,10)*17,
*d,eq.6,eq.19
c
c     common summary headers
      data hw/
     1     'source',            'escape',
     2     ' ',                 'energy cutoff',
     3     ' ',                 'time cutoff',
     4     'weight window',     'weight window',
     5     'cell importance',   'cell importance',
     6     'weight cutoff',     'weight cutoff',
     7     'energy importance', 'energy importance',
     8     'dxtran',            'dxtran',
     9     'forced collisions', 'forced collisions',
     1     'exp. transform',    'exp. transform'/
c
c     neutron summary headers
      data ((hp(i,j,1),i=1,2),j=1,7)/
     1     'upscattering',  'downscattering',
     2     ' ',             'capture',
     3     '(n,xn)',        'loss to (n,xn)',
     4     'fission',       'loss to fission',
     5     ' ',             ' ',
     6     ' ',             ' ',
     7     '(gamma,xn)',    ' '/
c
c     photon summary headers
      data ((hp(i,j,2),i=1,2),j=1,7)/
     1     'from neutrons',    'compton scatter',
     2     'bremsstrahlung',   'capture',
     3     'p-annihilation',   'pair production',
     4     'electron x-rays',  ' ',
     5     '1st fluorescence', ' ',
     6     '2nd fluorescence', ' ',
     7     '(gamma,xgamma)',   'loss to pn. abs.'/
c
c     electron summary headers
      data ((hp(i,j,3),i=1,2),j=1,7)/
     1     'pair production', 'scattering',
     2     'compton recoil',  'bremsstrahlung',
     3     'photo-electric',  ' ',
     4     'photon auger',    ' ',
     5     'electron auger',  ' ',
     6     'knock-on',        ' ',
     7     ' ',               ' '/
*d,eq.66
      do 110 j=1,17
*d,eq.69
*i,eq4a.22
      if(ha(1)(1:1).eq.' '.and.ha(2)(1:1).eq.' ')go to 110
*d,eq.97,eq.103
      if(pax(5,1,ip).gt.zero)tpp(1)=tmav(ip,1)/pax(5,1,ip)
      if((pax(5,12,ip)+pax(5,17,ip)).gt.zero)
     1     tpp(2)=tmav(ip,2)/(pax(5,12,ip)+pax(5,17,ip))
      if((pax(5,1,ip)+pax(5,12,ip)+pax(5,17,ip)).gt.zero)
     1     tpp(3)=(tmav(ip,1)+tmav(ip,2))
     2            /(pax(5,1,ip)+pax(5,12,ip)+pax(5,17,ip))
      do 150 i=1,17
*d,eq.185,eq.186
      if(npum.ne.0)write(iuo,365)npum
  365 format(/48h photonuclear production reaction mt loop failed,
     1       i4,7h times.)
      if(nppm.ne.0)write(iuo,370)nppm
  370 format(/49h neutron-induced photon production mt loop failed,
     1       i4,7h times.)
*/
*/ ********************************************************************
*/  Changes to action
*ident azPN
```

```
*/ ***********************************************************************
*/
*d,az.5
*d,az.7
*i,az4a.9
c
c        print the repeated structure/lattice table.
*d,az.43,az.95
      if(ink(130).ne.0)call tbl130
*d,az.98,az.211
      if(ink(140).ne.0.and.mxe.gt.0)call tbl140
*/
*/ ***********************************************************************
*/  Changes to disbug
*ident dbPN
*/ ***********************************************************************
*/
*d,db.2
*i,db.6
*if def,disspla
*i,db.40
*endif
*d,db.43
*/
*/ ***********************************************************************
*/  Changes to ratspl
*ident rlPN
*/ ***********************************************************************
*/
*d,rl.2,rl.3
*i,rl.5
*if def,mcplot
*if def,disspla
*i,rl.20
*endif
*d,rl.23
*/
*/ ***********************************************************************
*/  Changes to plot3d
*ident pyPN
*/ ***********************************************************************
*/
*d,py.2,py.3
*i,py.5
*if def,mcplot
*if def,disspla
*i,py.64
*endif
*d,py.67
*/
*/ ***********************************************************************
*/  Changes to x3dmat
*ident xyPN
*/ ***********************************************************************
*/
*d,xy.2,xy.3
*i,xy.6
*if def,mcplot
*if def,disspla
*i,xy.16
*endif
*d,xy.19
*/
*/ ***********************************************************************
*/  Changes to cgsdci
*ident gcPN
*/ ***********************************************************************
*/
*d,gc.2
*i,gc.6
*if def,cgs.or.disscgs
```

```
*i,gc.38
*endif
*d,gc.41
*/
*/ *********************************************************************
*/  Changes to menrl
*ident mfPN
*/ *********************************************************************
*/
*d,mf4a.2
*i,mf4a.4
*if def,multp
*i,mf4a.176
*endif
*d,mf4a.179
*/
*/ *********************************************************************
*/  Changes to getjdt
*ident gjPN
*/ *********************************************************************
*/
*d,gj.2
*i,gj.4
*if def,aix
*i,gj.6
*endif
*d,gj.9
*/
*/ *********************************************************************
*/  Add new routines after deck za
*/ *********************************************************************
*/
*addfile, za
*/
*/ *********************************************************************
*/  Add function acepxs
*/ *********************************************************************
*/
*deck fx
      function acepxs(it,kx,r,nt,lb)
c        interpolate value at index kx in table at xss(it)
c
c        Preconditions:
c
c          xss(it) is the first word of an appropriate table
c
c          kx is the current index in the table
c          r is the linear interpolation factor, 0.0 to 1.0,
c            between values at index kx and kx+1
c
c          table data are in the format:
c            xss(it)              = ie - index of first value
c            xss(it+1)            = nv - number of values given
c              ie+nv-1            = il - index of last value
c            xss(it+2..it+2+nv-1) = values(ie..il)
c            values are interpolated lin-lin from the current table
c
c            nt is interpolation scheme for an index out of bounds:
c              (index out of bounds is kx < ie or kx > il)
c              nt =  0 - return zero for no value out of energy bounds
c              nt =  1 - return end point value for energy out of bounds
c              nt = -1 - return zero below and end point above bounds
c
c          it, kx, r, nt and all xss(i) are read only variables
c
c
c        Postconditions:
c
c          returns value acepxs(kx) appropriately interpolated
c
c          returns lb for index error checking:
```

```
c            lb =  0 - value interpolated within table bounds
c            lb =  1 - value above table index bound
c            lb = -1 - value below table index bound
c
c          acepxs and lb are modified return values
c
*call cm
c
c        default return value is zero
      acepxs=0
c
c        find table bounds and check incoming index
      ie=nint(xss(it))
      nv=nint(xss(it+1))
      il=ie+nv-1
c
c        if index below table, return appropriate value
      if (kx.lt.ie) then
         if (nt.eq.1) then
            acepxs=xss(it+2)
         endif
         lb=-1
         return
      endif
c
c       if index above table, return appropriate value
      if (kx.gt.il .or. (kx.eq.il .and. r.gt.0)) then
         if (nt.eq.1 .or. nt.eq.-1) then
            acepxs=xss(it+2+nv-1)
         endif
         lb=1
         return
      endif
c
c       else interpolate the appropriate value from the table
      if (r.eq.0.) then
         acepxs=xss(it+2+kx-ie)
      else
         acepxs=xss(it+2+kx-ie)+r*(xss(it+2+kx-ie+1)-xss(it+2+kx-ie))
      endif
      lb=0
      return
      end
*/
*/ ********************************************************************
*/  Add subroutine angiso
*/ ********************************************************************
*/
*deck a0
      subroutine angiso(sa)
c        returns the scattering angle from an isotropic dist.
c
c        Preconditions:
c          None.
c
c        Postconditions:
c          return value sa is scattering angle mu in radians
c          mu value is always between -1.0 and 1.0
c
c          sa is a modified return value
c
*call cm
c
c >>>>>  sample from isotropic cosine scattering angular distribution
      sa=2.*rang()-1.
      return
      end
*/
*/ ********************************************************************
*/  Add subroutine anglw1
*/ ********************************************************************
```

```
*/
*deck a1
      subroutine anglw1(ld,sa)
c        returns the scattering angle from an equiprobable binned dist.
c
c        Preconditions:
c          xss(ld) is the first word of the appropriate distribution
c          32 bin equiprobable cosine scattering angle data in format:
c          mu(j=1..33) with mu(1)=-1.0 and mu(33)=1.0 and appropriately
c          spaced values in between to give equiprobability to each bin
c
c          ld and all xss(i) are read only variables
c
c        Postconditions:
c          return value sa is scattering angle mu in radians
c          mu value should be between -1.0 and 1.0 but isn't checked
c
c          sa is a modified return value
c
*call cm
c
c >>>>>  sample from table of 32 equiprobable binned cosine scat. angles
      rb=rang()*32.
      kb=int(rb)
      sa=xss(ld+kb)+(rb-kb)*(xss(ld+kb+1)-xss(ld+kb))
      return
      end
*/
*/ ********************************************************************
*/  Add subroutine anglw2
*/ ********************************************************************
*/
*deck a2
      subroutine anglw2(ld,sa)
c        returns the scattering angle from a tabular binned dist.
c
c        Preconditions:
c          xss(ld) is the first word of the appropriate distribution
c          tabulated angular probability distribution data in format:
c          mu(j=1..2+3*np) contains
c          jj - interpolation flag for cosine distribution
c            jj = 1 is for histogram interpolation
c            jj = 2 is for linear-linear interpolation
c          np - number of cosine points in distribution
c          cosine(k=1..np), pdf(k=1..np), cfd(k=1..np)
c
c          ld and all xss(i) are read only variables
c
c        Postconditions:
c          return value sa is scattering angle mu in radians
c          mu value should be between -1.0 and 1.0 but isn't checked
c
c          sa is a modified return value
c
*call cm
c
c >>>>>  sample from tabular cosine scattering angular distribution
      jj=nint(xss(ld))
      np=nint(xss(ld+1))
      kl=ld+2+2*np
      kh=kl+np-1
      rn=rang()
      call bnsrch(rn,kl,kh,ig)
      fa=xss(kl-np)
      ca=xss(kl-2*np)
      if(jj.eq.1) go to 10
      bb=(xss(kl-np+1)-fa)/(xss(kl-2*np+1)-ca)
      if(bb.eq.0) go to 10
      sa=ca+(sqrt(max(zero,fa**2+2.*bb*(rn-xss(kl))))-fa)/bb
      go to 20
   10 sa=ca+(rn-xss(kl))/fa
   20 return
      end
*/
*/ *********************************************************************
*/  Add subroutine bnsrch
*/ *********************************************************************
*/
*deck bn
      subroutine bnsrch(tv,il,ih,ig)
c        find the value tv in the array slice xss(il..ih)
c
c     Preconditions:
c        tv is a real value within the array
c        xss(il..ih) is an numerically ascending ordered array list
c        il is the first word of the list (index low)
c        ih is the last word of the list (index high)
c
c        tv and all xss(i) are read only variables
c
c     Postconditions:
c        il/ih bracket the test value tv such that ih-il.eq.1 and
c          xss(il) < tv < xss(ih)
c
c        Occasionally a value outside the array will be called for
c        and a reasonable value, the extreme edge of the array,
c        will be returned along with an non-zero error flag ig
c          ig =  0 is normal state - value found in array slice
c          ig = -1 is warning state - value below first array value
c          ig =  1 is warning state - value above last array value
c          ig =  2 is warning state - incoming indices equal tv=xss(il)
c          ig = -3 is warning state - incoming indices equal tv<xss(il)
c          ig =  3 is warning state - incoming indices equal tv>xss(il)
c        also il = ih = edge value for exit warning condition
c
c        il, ih and ig are modified return values
c
*call cm
c
c        warning state - value below table
      if(tv.lt.xss(il))then
         ig=-1
         ih=il
c
c        warning state - value above table
      else if(tv.gt.xss(ih))then
         ig=1
         il=ih
c
c        warning state - incoming indices are equal
      else if(il.eq.ih)then
         if(tv.eq.xss(il))then
            ig=2
         else if(tv.lt.xss(il))then
            ig=-3
         else
            ig=3
         endif
c
c        else handle normal value within table
      else
         ig=0
 10      if(ih-il.eq.1)return
         im=(il+ih)/2
         if(tv.lt.xss(im))go to 20
         il=im
         go to 10
 20      ih=im
         go to 10
c
c        end of cases
      endif
      return
      end
*/
*/ *********************************************************************
*/  Add subroutine coldpn
*/ *********************************************************************
*/
*deck dn
      subroutine coldpn
c        generate and bank particles from a photonuclear collision
c
*call cm
c
c     local character variable for printing warning about table fault
      character ht*10
c
c
c     *****************************************************************
c        sample photonuclear event
c     *****************************************************************
c
c     ****************
c     biased collision
      if (ispn.lt.0) then
c
c        compute photon weight lost to absorption
         wg=totpn/totm*wgt
c
c        save photoatomic portion of particle for further transport
         wgt=wgt-wg
         totm=totm-totpn
         call savept
         nt=0
c
c        account for forced photoabsorption in summaries
         tmavtc(2,2)=tmavtc(2,2)+tme*wg
         tmavtc(2,3)=tmavtc(2,3)+tme*wg
         paxtc(5,17,2)=paxtc(5,17,2)+wg
         paxtc(6,17,2)=paxtc(6,17,2)+erg*wg
         pwb(kpwb+2,20,icl)=pwb(kpwb+2,20,icl)-wg
c
c     *****************
c     natural collision
      else
c
c        determine if collision is photonuclear
         if (rang()*totm.lt.totpn) then
c
c           and set parameters to indicate photoabsorption
c           (summaries are updated all the way back in hstory)
            call savept
            nt=17
            wg=wgt
         else
c
c           or account for non-sampled photonuclear event
c           and return to colidp to sample photoatomic event
            totm=totm-totpn
            return
         endif
c
      endif
c
c     ***********************************************************
c     sample cumulative pn x-section to find the collision nuclide
c
      im=mat(lmat+icl)
      if (npq(lnpq+im).eq.1) then
         nn=jmd(ljmd+im)
         iex=lmn(llmn+nn)
      else
```

```
        mc=0
 10     rt=rang()*totpn
        do 20 nn=jmd(ljmd+im),jmd(ljmd+im+1)-1
           iex=lmn(llmn+nn)
           if (iex.eq.0) go to 20
           rt=rt-rtc(krtc+2,iex)*fme(lfme+nn)
           if (rt.lt.zero) then
c             nuclide nn is chosen for collision
              go to 30
           endif
 20     continue
c
c       if problem sampling nuclide, print warning
        call errprn(0,mc,2,zero+nps,zero+nmt(lnmt+im),'NPS','MAT',
     1       'problem sampling collision nuclide. resampling.')
c
        if (mc.lt.100) go to 10
c       if isotope still not found, exit complaining about data
        call expirx(1,'coldpn',
     1       'contact nucldata@lanl.gov for assitance')
        return
      endif
c
 30   continue
c
c     ************************************
c     update the nuclide collision summary
      mpan=ipan(lipa+icl)+nn-jmd(ljmd+im)
      pan(kpan+3,1,mpan)=pan(kpan+3,1,mpan)+1.0
      pan(kpan+3,2,mpan)=pan(kpan+3,2,mpan)+wg
c
c     *****************************************
c     reset default parameters for new particle
      ncp=0
      jsu=0
c
c     ******************************
c     save needed incident parameters
      ei=erg
c     uold(1:3)=uuu,vvv,www of incident photon; set in colidp
c
c     *******************************************
c     sample from each species of emitted particle
c     -------------------------------------------
      do 120 jp=1,nxs(lnxs+5,iex)
c
c        ----------------------------------------
c        get particle type and check for transport
         ipt=ixs(lixs+1,jp,iex)
         if (kpt(ipt).eq.0.or.fiml(ipt).eq.zero.or.ipt.gt.3)
     1       go to 120
c
c        -------------------------------------------------------------
c        determine the total production cross-section for this particle
         lb=0
         tp=acepxs(ixs(lixs+3,jp,iex),ktc(kktc+1,iex),
     1            rtc(krtc+1,iex),0,lb)
c
c        ---------------------------------------------
c        if outside valid range, try next particle type
         if (lb.ne.0) go to 120
c
c        ------------------------------------------------
c        determine the average number of particles emitted
         fp=tp/rtc(krtc+2,iex)
         np=int(rang()+fp)
c
c        --------------------------------------------------
c        if no particles are sampled, try next particle type
         if (np.eq.0) go to 120
c
```

```
c        ------------------------------------------------------
c        check to see if new particle is inside a dxtran sphere
         idx=0
         do 40 nd=1,ndx(ipt)
            ds= (xxx-dxx(ipt,1,nd))**2+
     1          (yyy-dxx(ipt,2,nd))**2+
     2          (zzz-dxx(ipt,3,nd))**2
            if (ds.lt.dxx(ipt,5,nd)) idx=nd
 40      continue
c
c        ------------------------
c        reset the default weight
         wgt=wg
c
c        ---------------------------------------------------------
c        use weight windows to control weight of emitted particles
c          works here only if there is a single weight-window
c          energy-group for the secondary particle
c          (otherwise, i.e. for multiple energy-groups,
c          check after sampling)
         if (idx.eq.0 .and. nww(ipt).eq.1 .and.
     1        (abs(wwp(ipt,4)).eq.one.or.wwp(ipt,5).lt.zero)) then
            ww=wwval(ipt,icl,1,1)
            if (ww.lt.zero) then
c              negative weight window kills particle
               paxtc(1,4,ipt)=paxtc(1,4,ipt)+1.0
               paxtc(2,4,ipt)=paxtc(2,4,ipt)+wgt
               paxtc(4,4,ipt)=paxtc(4,4,ipt)+1.0
               paxtc(5,4,ipt)=paxtc(5,4,ipt)+wgt
               go to 120
            elseif (ww.eq.zero) then
c              zero weight window does nothing
               continue
            elseif (wgt.lt.ww) then
c              russian roulette particles below the weight window
               ws=min(wgt*wwp(ipt,3),ww*wwp(ipt,2))
               mr=0
               do 50 ir=1,np
                  if (rang()*ws.gt.wgt) mr=mr+1
 50            continue
               np=np-mr
               wgt=ws
            elseif (wgt.gt.wwp(ipt,1)*ww) then
c              split particles above the weight window
               nw=nint(min(wwp(ipt,3),wgt/(ww*wwp(ipt,2))))
               wgt=wgt/nw
               np=np*nw
            endif
c
         endif
         if (np.eq.0) go to 120
c
c        -------------------------------------------
c        sample emission parameters for each particle
c        ============================================
         do 110 ii=1,np
c
c           ====================================================
c           determine from which reaction to sample the particle
            nr=ixs(lixs+2,jp,iex)
            if (nr.eq.1) then
               ixre=1
            else
c              generate a random production xs to sample too
               mp=0
               is=ixs(lixs+8,jp,iex)
 60            rx=tp*rang()
c
c              ===================================================
c              loop over the available reactions until xs fulfilled
c              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
              do 90 ixre=1,nr
c
c                retrieve the next yield block offset locator
                 ks=nint(xss(ixs(lixs+7,jp,iex)+ixre-1))
c
c                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c                handle the different reaction yield blocks by MFTYPE
c
                 if (nint(xss(is+ks-1)).eq.6  .or.
     1               nint(xss(is+ks-1)).eq.12 .or.
     2               nint(xss(is+ks-1)).eq.16) then
c                    MFTYPE 6, 12 or 16 - erg. dependent MT multiplier
c                      data is in format:
c                      xss(is+ks-1) = MFTYPE - reaction type indicator
c                      xss(is+ks)   = MTMULT - MT reaction applied to
c                      xss(is+ks+...) std. energy/data table entries
c                        NR, [NBT(i:i=1..NR), INT(i:i=1..NR),]
c                        NV, E(i:i=1..NV), Y(i:i=1..NV)
c
c                   get yield value for incident photon energy
                    yd=acefcn(is+ks+1,ei,ln)
c
c                   look up pointer to MT reaction xs data
                    do 70 im=1,nxs(lnxs+4,iex)
                       if ( nint(xss(is+ks)) .eq.
     1                      nint(xss(jxs(ljxs+6,iex)+im-1)) ) then
                          go to 80
                       endif
 70                 continue
c
c                   if fell through loop, exit complaining about data
                    im=0
                    call zaid(2,ht,ixl(lixl+1,iex))
                    call errprn(1,im,2,zero+ixre,xss(is+ks),
     1                   'IXR','MT','cound not find mt for yield'
     2                   //' multiplier. zaid = '//ht)
                    call expirx(1,'coldpn',
     1                   'contact nucldata@lanl.gov for assitance')
                    return
c
c                   look up reaction XS using MT index im found above
 80                 continue
                    ix=jxs(ljxs+9,iex)
                    kx=nint(xss(jxs(ljxs+8,iex)+im-1))
                    xs=acepxs(ix+kx-1,ktc(kktc+1,iex),
     1                        rtc(krtc+1,iex),0,lb)
c
c                   compute partial prod xs from yield and reaction xs
                    px=yd*xs
c
c                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                 elseif (xss(is+ks).eq.13) then
c                  MFTYPE 13 - Partial production cross section data
c                    data is in format:
c                    xss(is+ks)   = MFTYPE - reaction type indicator
c                    xss(is+ks+1) = ie - index of first value
c                    xss(is+ks+2) = nv - number of values listed
c                    xss(is+ks+3..is+ks+2+nv-1) = pxs(ie..ie+nv-1)
c
c                   find the partial production xs for this reaction
                    px=acepxs(is+ks+1,ktc(kktc+1,iex),
     1                        rtc(krtc+1,iex),0,lb)
c
c                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c                handle an undefined reaction type MFTYPE
                 else
                    im=0
                    call zaid(2,ht,ixl(lixl+1,iex))
                    call errprn(1,im,2,zero+ixre,xss(is+ks-1),
     1                   'IXR','MFT','cound not find MFTYPE.'
     2                   //' zaid = '//ht)
```

```
                  call expirx(1,'coldpn',
     1                 'contact nucldata@lanl.gov for assitance')
                  return
               endif
c
c              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c              if the partial XS has been fulfilled, use this rx
               rx=rx-px
               if (rx.lt.0.) go to 100
c
c              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c              if problem sampling reaction, print a warning
               if (ixre.ne.ixs(lixs+2,jp,iex)) go to 90
               call zaid(2,ht,ixl(lixl+1,iex))
               call errprn(1,mp,4,erg,zero,'ERG',' ',
     1              ' reaction mt not found. collision resampled.'
     2              //' zaid = '//ht//'.')
               npum=npum+1
               if (mp.lt.100) go to 60
c
c              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c              if reaction still not found,
c                 exit complaining about data
               call expirx(1,'coldpn',
     1              'contact nucldata@lanl.gov for assitance')
               return
c
c           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
c           end of loop over partial reactions
c           ==================================
 90         continue
         endif
c
c        ====================================
c        ixre now set to the sampled reaction
 100     continue
c
c        ==================================
c        set up pointers for call to acecas
         erg=ei
         ipsc=0
         npa=1
         mtp=nint(xss(ixs(lixs+5,jp,iex)+ixre-1))
         ntyn=nint(xss(ixs(lixs+6,jp,iex)+ixre-1))
         ia=ixs(lixs+10,jp,iex)
         ka=nint(xss(ixs(lixs+9,jp,iex)+ixre-1))
         id=ixs(lixs+12,jp,iex)
         kd=nint(xss(ixs(lixs+11,jp,iex)+ixre-1))
c
c        =====================
c        sample a new particle
         call acecas(1,2,zero,ia,ka,id,kd)
         erg=colout(1,1)
         vel=slite*sqrt(erg*(erg+2.*gpt(ipt)))/(erg+gpt(ipt))
         call rotas(colout(2,1),uold,uuu,lev,irt)
         if (kdb.ne.0) return
c
c        =================================
c        update the various summary tables
c
c        update summary for particle creation
         paxtc(1,17,ipt)=paxtc(1,17,ipt)+1.0
         paxtc(2,17,ipt)=paxtc(2,17,ipt)+wgt
         paxtc(3,17,ipt)=paxtc(3,17,ipt)+wgt*erg
c
c        update the particle creation weight balance for this cell
         pwb(kpwb+ipt,21,icl)=pwb(kpwb+ipt,21,icl)+wgt
c
c        update the nuclide activity summaries
         if      (ipt.eq.1) then
c           update photoneutron production summary
```

```
               pan(kpan+3,6,mpan)=pan(kpan+3,6,mpan)+1.0
               pan(kpan+3,7,mpan)=pan(kpan+3,7,mpan)+wgt
               pan(kpan+3,8,mpan)=pan(kpan+3,8,mpan)+wgt*erg
            else if (ipt.eq.2) then
c              update photophoton production summary
               pan(kpan+3,3,mpan)=pan(kpan+3,3,mpan)+1.0
               pan(kpan+3,4,mpan)=pan(kpan+3,4,mpan)+wgt
               pan(kpan+3,5,mpan)=pan(kpan+3,5,mpan)+wgt*erg
            endif
c
c           =======================
c           check for energy cutoff
            if (erg.lt.elc(ipt)) then
               paxtc(4,2,ipt)=paxtc(4,2,ipt)+1.0
               paxtc(5,2,ipt)=paxtc(5,2,ipt)+wgt
               paxtc(6,2,ipt)=paxtc(6,2,ipt)+wgt*erg
               pwb(kpwb+ipt,4,icl)=pwb(kpwb+ipt,4,icl)-wgt
               go to 110
            endif
c
c           ==============================================
c           force kalbach-chadwick or isotropic for calcps
c             as called in dxtran or tallyd
c             ipsc=999 will fatal error any other reaction
            if (ipsc.eq.14) then
               ipsc=17
            elseif (ixcos.eq.0) then
               ipsc=106
            else
               ipsc=999
            endif
c
c           ========================================
c           check for contribution to dxtran spheres
            if (ndx(ipt).ne.0) then
c              only if not in the sphere or more than one exists
               if (ndx(ipt).gt.1.or.idx.eq.0) call dxtran
               if (kdb.ne.0) return
            endif
c
c           ==========================================
c           check for contribution to detector tallies
            if (ndet(ipt).ne.0) then
               call tallyd
               if (kdb.ne.0) return
            endif
c
c           ======================================================
c           use weight-windows to control weight of banked particles
c             only do this is there are multiple energy-groups for
c             this secondary particle type
            if (idx.eq.0 .and. nww(ipt).gt.1 .and.
     1          (abs(wwp(ipt,4)).eq.one.or.wwp(ipt,5).lt.zero)) then
               ic=0
               iw=nww(ipt)
 1000          if (iw-ic.ne.1) then
                  ih=(ic+iw)/2
                  if (erg.lt.wwe(lwwe+ih+mww(ipt))) then
                     iw=ih
                  else
                     ic=ih
                  endif
                  go to 1000
               endif
               ww=wwval(ipt,icl,iw,1)
               if (ww.lt.zero) then
c                 negative weight window kills particle
c                 update summaries to indicate weight window activity
                  paxtc(4,4,ipt)=paxtc(4,4,ipt)+1.0
                  paxtc(5,4,ipt)=paxtc(5,4,ipt)+wgt
                  paxtc(6,4,ipt)=paxtc(6,4,ipt)+wgt*erg
```

```
                 pwb(kpwb+ipt,6,icl)=pwb(kpwb+ipt,6,icl)-wgt
                 go to 110
              elseif (ww.eq.zero) then
c                zero weight window does nothing
                 continue
              elseif (wgt.lt.ww) then
c                russian roulette particles below the weight window
                 ws=min(wgt*wwp(ipt,3),ww*wwp(ipt,2))
                 rr=rang()
                 if (rr*ws.lt.wgt) then
                    paxtc(2,4,ipt)=paxtc(2,4,ipt)+(ws-wgt)
                    paxtc(3,4,ipt)=paxtc(3,4,ipt)+(ws-wgt)*erg
                    pwb(kpwb+ipt,6,icl)=pwb(kpwb+ipt,6,icl)+(ws-wgt)
                    wgt=ws
                 else
                    paxtc(4,4,ipt)=paxtc(4,4,ipt)+1.0
                    paxtc(5,4,ipt)=paxtc(5,4,ipt)+wgt
                    paxtc(6,4,ipt)=paxtc(6,4,ipt)+wgt*erg
                    pwb(kpwb+ipt,6,icl)=pwb(kpwb+ipt,6,icl)-wgt
                    go to 110
                 endif
              elseif (wgt.gt.wwp(ipt,1)*ww) then
c                split particles above the weight window
                 npa=nint(min(wwp(ipt,3),wgt/(ww*wwp(ipt,2))))
                 paxtc(4,4,ipt)=paxtc(4,4,ipt)+(npa-1.0)
                 wgt=wgt/npa
              endif
c
           endif
c
c          ===================================
c          bank the new particle for transport
           call bankit(27)
c
c       ============================================
c       end of loop generate np particles of type ipt
c       --------------------------------------------
 110    continue
c
c
c     ---------------------------------------
c     end of loop over all particle emissions
c     ***************************************
 120  continue
c
c     *****************************************************************
c        retrieve incident photon and continue
c     *****************************************************************
      call retrpt
      nter=nt
      return
      end
*/
*/ *******************************************************************
*/  Add subroutine expgpn
*/ *******************************************************************
*/
*deck xu
      subroutine expgpn
c        remove unneeded data from table type u
c        still to be written
*call cm
c
      return
      end
*/
*/ *******************************************************************
*/  Add subroutine pnctot
*/ *******************************************************************
*/
*deck nt
```

```
      subroutine pnctot(mk)
c        calculate the photonuclear cross section in material mk
c
c        local variables: ii - isotope index, it - table index,
c          ih - upper energy index, il - lower energy index,
c
c        global variables:
c          jmd - material isotope indices
c          lmn - photonuclear isotope table indices
c          erg - current photon energy
c          pnt(mk) - the lowest photonuclear threshold in material mk
c          ktc(1,table) - table index above current energy
c          ktc(2,table) - flag to warn about energy out of table bounds
c          rtc(1,table) - current linear table interpolation factor
c          rtc(2,table) - current total cross section
c          rtc(6,table) - current energy being interpolated
c
*call cm
c
      totpn=zero
c
c     if below photonuclear threshold for material, return
      if (erg.le.pnt(lpnt+mk)) return
c
c     calculate/accumulate the cross sections by nuclide in material
      do 20 ii=jmd(ljmd+mk),jmd(ljmd+mk+1)-1
         it=lmn(llmn+ii)
         if(it.eq.0)go to 20
         if(erg.eq.rtc(krtc+6,it))go to 10
         rtc(krtc+6,it)=erg
c
c        find index of current energy
         il=jxs(ljxs+1,it)
         ih=il+nxs(lnxs+3,it)-1
         call bnsrch(erg,il,ih,ktc(kktc+2,it))
c
c        if in the table range, store the values normally
         if(ktc(kktc+2,it).eq.0)then
            ktc(kktc+1,it)=ih-jxs(ljxs+1,it)
            rtc(krtc+1,it)=(erg-xss(il))/(xss(ih)-xss(il))
            k=ktc(kktc+1,it)+jxs(ljxs+2,it)
            rtc(krtc+2,it)=(xss(k)-xss(k-1))*rtc(krtc+1,it)+xss(k-1)
c
c        if above last energy value, use boundary values
         else if(ktc(kktc+2,it).gt.0)then
            ktc(kktc+1,it)=il-jxs(ljxs+1,it)
            rtc(krtc+1,it)=0.0
            rtc(krtc+2,it)=xss(jxs(ljxs+2,it)+ktc(kktc+1,it))
c
c        if below photonuclear threshold, use zero values
         else
            ktc(kktc+1,it)=0
            rtc(krtc+1,it)=0.
            rtc(krtc+2,it)=0.
         endif
c
c        accumulate the photonuclear microscopic xs for the material
 10      continue
         totpn=totpn+rtc(krtc+2,it)*fme(lfme+ii)
 20    continue
c
c      add totpn to totm for distance to collision calculations
       totm=totm+totpn
       return
       end
*/
*/ ********************************************************************
*/  Add subroutine retrpt
*/ ********************************************************************
*/
*deck qr
```

```
      subroutine retrpt
c        retrieve the current particle parameters from the particle bank
c
*call cm
c
c        retrieve the integer variable parameters
      do 10 i=1,lpblcm
         jpblcm(i)=jpb9cm(npb,i)
 10   continue
c
c        retrieve the real variable parameters
      do 20 i=1,npblcm
         gpblcm(i)=gpb9cm(npb,i)
 20   continue
c
c        decrement the stack counter
      npb=npb-1
c
      return
      end
*/
*/ *********************************************************************
*/  Add subroutine savept
*/ *********************************************************************
*/
*deck qs
      subroutine savept
c        save the current particle parameters to the particle bank
c
*call cm
c
c        increment the stack counter
      npb=npb+1
c
c        save the real variable parameters
      do 10 i=1,npblcm
         gpb9cm(npb,i)=gpblcm(i)
 10   continue
c
c        save the integer variable parameters
      do 20 i=1,lpblcm
         jpb9cm(npb,i)=jpblcm(i)
 20   continue
c
      return
      end
*/
*/ *********************************************************************
*/  Add subroutine tbl130
*/ *********************************************************************
*/
*deck t3
      subroutine tbl130
c        print weight balance for each particle in each cell
*call cm
c
c     ****************************************************************
c        print table 130 is the weight balance for each particle type
c          in each cell such that one table exists for each particle
c
c        the table consists of a set of header information, an external
c          events section, a variance reduction event section, a
c          physical events section and footer information for each cell
c        print table 130 currently consists of:
c          lh    =  3 lines of header information
c          le    =  9 lines of external event information
c          lv    = 11 lines of variance reduction event information
c          physical events are particle specific
      dimension lp(mipt)
c          lp(1) = 10 lines of neutron physical event information
c          lp(2) = 13 lines of photon physical event information
```

```
c          lp(3) = 10 lines of electron physical event information
c          lf    =  4 lines of footer information
c          max     40 total lines in the print matrix
      data nr/40/
c
c        cells are printed across the page left to right with a total
c          over all cells being printed last; only x number of cell
c          columns can be printed per page and therefore multiple prints
c          of the table may be necessary to print all cells
c          standard row width is 132 char. columns
c          the row header is 16 characters (1 space and 15 chars.)
c          that leaves room for 9 12 char. data values to be written
c          this sets the max number of data column values, nc, to 9
      data nc/9/
c
c          there are 3 subtotals and 1 total computed for each
c          column; these values are stored only per matrix print
c          *** make sure the 9 below matches nc above
      dimension st(4,9)
c
c        the partial tables are constructed in a character variable
c          matrix containing the maximum number of cells; the partial
c          table is then dumped to output as needed when it is full
c          thus the total storage for the maximum entire matrix is
c          40 rows         (check against nr above)
c           9 data columns (check against nc above)
c          row headers are printed into character variable hb
c          data values are printed into character variable ha
      character ha(40,9)*12,hb(40)*15
c
c     *****************************************************************
c
c     loop over each particle type
      do 110 ip=1,mipt
c
c        skip particle type if not being transported
         if(kpt(ip).eq.0)go to 110
c
c        write the appropriate print table header
         write(iuo,10)hnp(ip)
 10      format(1h1,a8,28h weight balance in each cell,80x,
    1          15hprint table 130)
c
c        compute the total weight balance over all cells
         do 20 j=1,21
            pwb(lpwb+ip,j,mxa+1)=zero
 20      continue
         do 30 j=1,21
         do 30 k=1,mxa
c           the next line is needed to prevent partial weights (e.g.
c              weight entering zero importance cell) from being added
c              to the totals
            if(fim(lfim+ip,k).eq.zero)go to 30
            pwb(lpwb+ip,j,mxa+1)=pwb(lpwb+ip,j,mxa+1)+pwb(lpwb+ip,j,k)
 30      continue
c
c        initialize the print matrix, sub/totals and column counter
         do 40 i=1,nr
         do 40 j=1,nc
            ha(i,j)=' '
            hb(i)=' '
 40      continue
         do 50 i=1,4
         do 50 j=1,nc
            st(i,j)=zero
 50      continue
         kc=0
c
c        loop over all cells (and the total), printing the
c           matrix as it fills
         do 100 ic=1,mxa+1
```

```
c
c           skip any cells with a zero importance
            if(fim(lfim+ip,ic).eq.zero.and.ic.ne.mxa+1)go to 100
            kc=kc+1
c
c           **********************************
c           write the table header information
c           ls is the start line for this table section
            ls=0
            if(kc.eq.1)then
c              lh is the number of table header lines
               lh=3
c              leave hb(ls+1) blank
               write(hb(ls+2),'(a15)')'     cell index'
               write(hb(ls+3),'(a15)')'    cell number'
            endif
c           the header is the index/name or total
            if(ic.lt.mxa+1)then
c              leave ha(ls+1,kc) blank
               write(ha(ls+2,kc),'(4x,i5,3x)')ic
               write(ha(ls+3,kc),'(4x,i5,3x)')ncl(lncl+ic)
            else
c              leave ha(ls+1,kc) blank
c              leave ha(ls+2,kc) blank
               write(ha(ls+3,kc),'(4x,a5,3x)')'total'
            endif
c
c           *****************************************
c           write the external event headers and data
            ls=ls+lh
            if(kc.eq.1)then
c              le is the number of external event lines
               le=9
c              leave hb(ls+1) blank
               write(hb(ls+2),'(a15)')'external events'
               write(hb(ls+3),'(a15)')'       entering'
               write(hb(ls+4),'(a15)')'         source'
               write(hb(ls+5),'(a15)')'  energy cutoff'
               write(hb(ls+6),'(a15)')'    time cutoff'
               write(hb(ls+7),'(a15)')'        exiting'
               write(hb(ls+8),'(a15)')' '
               write(hb(ls+9),'(a15)')'     subtotal  '
            endif
c           leave ha(ls+1,kc) blank
c           leave ha(ls+2,kc) blank
            write(ha(ls+3,kc),'(1pe12.4)')pwb(lpwb+ip,1,ic)*fpi
            write(ha(ls+4,kc),'(1pe12.4)')pwb(lpwb+ip,2,ic)*fpi
            write(ha(ls+5,kc),'(1pe12.4)')pwb(lpwb+ip,3,ic)*fpi
            write(ha(ls+6,kc),'(1pe12.4)')pwb(lpwb+ip,4,ic)*fpi
            write(ha(ls+7,kc),'(1pe12.4)')pwb(lpwb+ip,5,ic)*fpi
            write(ha(ls+8,kc),'(1a12)')'  ----------'
            st(1,kc)=fpi*(
     1           pwb(lpwb+ip,1,ic)+
     2           pwb(lpwb+ip,2,ic)+
     3           pwb(lpwb+ip,3,ic)+
     4           pwb(lpwb+ip,4,ic)+
     5           pwb(lpwb+ip,5,ic))
            write(ha(ls+9,kc),'(1pe12.4)')st(1,kc)
c
c           *********************************************
c           write the variance reduction headers and data
            ls=ls+le
            if(kc.eq.1)then
c              lv is the number of variance reduction event lines
               lv=11
c              leave hb(ls+1) blank
               write(hb(ls+2),'(a15)')'var.red. events'
               write(hb(ls+3),'(a15)')'  weight window'
               write(hb(ls+4),'(a15)')'      cell imp.'
               write(hb(ls+5),'(a15)')'  weight cutoff'
               write(hb(ls+6),'(a15)')'    energy imp.'
               write(hb(ls+7),'(a15)')'         dxtran'
               write(hb(ls+8),'(a15)')'   forced coll.'
               write(hb(ls+9),'(a15)')'    exp. trans.'
c              leave hb(ls+10) blank
               write(hb(ls+11),'(a15)')'     subtotal  '
            endif
c           leave ha(ls+1,kc) blank
c           leave ha(ls+2,kc) blank
            write(ha(ls+3,kc),'(1pe12.4)')pwb(lpwb+ip,6,ic)*fpi
            write(ha(ls+4,kc),'(1pe12.4)')pwb(lpwb+ip,7,ic)*fpi
            write(ha(ls+5,kc),'(1pe12.4)')pwb(lpwb+ip,8,ic)*fpi
            write(ha(ls+6,kc),'(1pe12.4)')pwb(lpwb+ip,9,ic)*fpi
            write(ha(ls+7,kc),'(1pe12.4)')pwb(lpwb+ip,10,ic)*fpi
            write(ha(ls+8,kc),'(1pe12.4)')pwb(lpwb+ip,11,ic)*fpi
            write(ha(ls+9,kc),'(1pe12.4)')pwb(lpwb+ip,12,ic)*fpi
            write(ha(ls+10,kc),'(a12)')'  ----------'
            st(2,kc)=fpi*(
     1           pwb(lpwb+ip,6,ic)+
     2           pwb(lpwb+ip,7,ic)+
     3           pwb(lpwb+ip,8,ic)+
     4           pwb(lpwb+ip,9,ic)+
     5           pwb(lpwb+ip,10,ic)+
     6           pwb(lpwb+ip,11,ic)+
     7           pwb(lpwb+ip,12,ic))
            write(ha(ls+11,kc),'(1pe12.4)')st(2,kc)
c
c           **************************************************
c           write the physical event section by particle type
c
c           write the neutron physical event header and data
            if(ip.eq.1)then
               ls=ls+lv
               if(kc.eq.1)then
c                 lp(1) is the number of neutron physical event lines
                  lp(1)=10
c                 leave hb(ls+1) blank
                  write(hb(ls+2),'(a15)')'physical events'
                  write(hb(ls+3),'(a15)')'         (n,xn)'
                  write(hb(ls+4),'(a15)')'        fission'
                  write(hb(ls+5),'(a15)')'        capture'
                  write(hb(ls+6),'(a15)')' loss to (n,xn)'
                  write(hb(ls+7),'(a15)')'loss to fission'
                  write(hb(ls+8),'(a15)')'     (gamma,xn)'
c                 leave hb(ls+9) blank
                  write(hb(ls+10),'(a15)')'     subtotal  '
               endif
c              leave ha(ls+1,kc) blank
c              leave ha(ls+2,kc) blank
               write(ha(ls+3,kc),'(1pe12.4)')pwb(lpwb+ip,13,ic)*fpi
               write(ha(ls+4,kc),'(1pe12.4)')pwb(lpwb+ip,14,ic)*fpi
               write(ha(ls+5,kc),'(1pe12.4)')pwb(lpwb+ip,15,ic)*fpi
               write(ha(ls+6,kc),'(1pe12.4)')pwb(lpwb+ip,16,ic)*fpi
               write(ha(ls+7,kc),'(1pe12.4)')pwb(lpwb+ip,17,ic)*fpi
               write(ha(ls+8,kc),'(1pe12.4)')pwb(lpwb+ip,21,ic)*fpi
               write(ha(ls+9,kc),'(a12)')'  ----------'
               st(3,kc)=fpi*(
     1              pwb(lpwb+ip,13,ic)+
     2              pwb(lpwb+ip,14,ic)+
     3              pwb(lpwb+ip,15,ic)+
     4              pwb(lpwb+ip,16,ic)+
     5              pwb(lpwb+ip,17,ic)+
     6              pwb(lpwb+ip,21,ic))
               write(ha(ls+10,kc),'(1pe12.4)')st(3,kc)
c
c           write the photon physical event header and data
            else if(ip.eq.2)then
               ls=ls+lv
               if(kc.eq.1)then
c                 lp(2) is the number of photon physical event lines
                  lp(2)=13
c                 leave hb(ls+1) blank
```

```
               write(hb(ls+2), '(a15)')'physical events'
               write(hb(ls+3), '(a15)')'  from neutrons'
               write(hb(ls+4), '(a15)')' bremsstrahlung'
               write(hb(ls+5), '(a15)')' p-annihilation'
               write(hb(ls+6), '(a15)')'electron x-rays'
               write(hb(ls+7), '(a15)')'   flourescence'
               write(hb(ls+8), '(a15)')'    pe. capture'
               write(hb(ls+9), '(a15)')'pair production'
               write(hb(ls+10),'(a15)')' pn. absorbtion'
               write(hb(ls+11),'(a15)')' (gamma,xgamma)'
c              leave hb(ls+12) blank
               write(hb(ls+13),'(a15)')'     subtotal  '
            endif
c           leave ha(ls+1,kc) blank
c           leave ha(ls+2,kc) blank
            write(ha(ls+3,kc), '(1pe12.4)')pwb(lpwb+ip,13,ic)*fpi
            write(ha(ls+4,kc), '(1pe12.4)')pwb(lpwb+ip,14,ic)*fpi
            write(ha(ls+5,kc), '(1pe12.4)')pwb(lpwb+ip,15,ic)*fpi
            write(ha(ls+6,kc), '(1pe12.4)')pwb(lpwb+ip,16,ic)*fpi
            write(ha(ls+7,kc), '(1pe12.4)')pwb(lpwb+ip,17,ic)*fpi
            write(ha(ls+8,kc), '(1pe12.4)')pwb(lpwb+ip,18,ic)*fpi
            write(ha(ls+9,kc), '(1pe12.4)')pwb(lpwb+ip,19,ic)*fpi
            write(ha(ls+10,kc),'(1pe12.4)')pwb(lpwb+ip,20,ic)*fpi
            write(ha(ls+11,kc),'(1pe12.4)')pwb(lpwb+ip,21,ic)*fpi
            write(ha(ls+12,kc),'(a12)')'  ----------'
            st(3,kc)=fpi*(
     1           pwb(lpwb+ip,13,ic)+
     2           pwb(lpwb+ip,14,ic)+
     3           pwb(lpwb+ip,15,ic)+
     4           pwb(lpwb+ip,16,ic)+
     5           pwb(lpwb+ip,17,ic)+
     6           pwb(lpwb+ip,18,ic)+
     7           pwb(lpwb+ip,19,ic)+
     8           pwb(lpwb+ip,20,ic)+
     9           pwb(lpwb+ip,21,ic))
            write(ha(ls+13,kc),'(1pe12.4)')st(3,kc)
c
c        write the electron physical event header and data
         else if(ip.eq.3)then
            ls=ls+lv
            if(kc.eq.1)then
c              lp(3) is the number of photon physical event lines
               lp(3)=10
c              leave hb(ls+1) blank
               write(hb(ls+2), '(a15)')'physical events'
               write(hb(ls+3), '(a15)')'pair production'
               write(hb(ls+4), '(a15)')' compton recoil'
               write(hb(ls+5), '(a15)')' photo-electric'
               write(hb(ls+6), '(a15)')'   photon auger'
               write(hb(ls+7), '(a15)')' electron auger'
               write(hb(ls+8), '(a15)')'        knock-on'
c              leave hb(ls+9) blank
               write(hb(ls+10),'(a15)')'     subtotal  '
            endif
c           leave ha(ls+1,kc) blank
c           leave ha(ls+2,kc) blank
            write(ha(ls+3,kc),'(1pe12.4)')pwb(lpwb+ip,13,ic)*fpi
            write(ha(ls+4,kc),'(1pe12.4)')pwb(lpwb+ip,14,ic)*fpi
            write(ha(ls+5,kc),'(1pe12.4)')pwb(lpwb+ip,15,ic)*fpi
            write(ha(ls+6,kc),'(1pe12.4)')pwb(lpwb+ip,16,ic)*fpi
            write(ha(ls+7,kc),'(1pe12.4)')pwb(lpwb+ip,17,ic)*fpi
            write(ha(ls+8,kc),'(1pe12.4)')pwb(lpwb+ip,18,ic)*fpi
            write(ha(ls+9,kc),'(a12)')'  ----------'
            st(3,kc)=fpi*(
     1           pwb(lpwb+ip,13,ic)+
     2           pwb(lpwb+ip,14,ic)+
     3           pwb(lpwb+ip,15,ic)+
     4           pwb(lpwb+ip,16,ic)+
     5           pwb(lpwb+ip,17,ic)+
     6           pwb(lpwb+ip,18,ic))
            write(ha(ls+10,kc),'(1pe12.4)')st(3,kc)
            endif
c
c           **********************************
c           write the table footer information
            ls=ls+lp(ip)
            if(kc.eq.1)then
c              lf is the number of table footer lines
               lf=4
c              leave hb(ls+1) blank
c              leave hb(ls+2) blank
               write(hb(ls+3),'(a15)')'        total  '
c              leave hb(ls+4) blank
            endif
c           leave ha(ls+1,kc) blank
            write(ha(ls+2,kc),'(a12)')'  ----------'
            st(4,kc)=st(1,kc)+st(2,kc)+st(3,kc)
            if(abs(st(4,kc)).lt.1e-10)st(4,kc)=zero
            write(ha(ls+3,kc),'(1pe12.4)')st(4,kc)
c           leave ha(ls+4,kc) blank
c
c           ****************************************************
c           if print matrix full or last cell reached, write the
c              print matrix to the output file and reinitialize
            if(kc.eq.9.or.ic.eq.mxa+1)then
               do 70 il=1,lh+le+lv+lp(ip)+lf
                  write(iuo,60)hb(il),(ha(il,i),i=1,9)
 60               format(1x,a15,9a12)
 70            continue
               do 80 i=1,nr
               do 80 j=1,nc
                  ha(i,j)=' '
 80            continue
               do 90 i=1,4
               do 90 j=1,nc
                  st(i,j)=zero
 90            continue
               kc=0
            endif
c
c        **********************
c        end of loop over cells
 100     continue
c
c     **************************
c     end of loop over particles
 110  continue
      return
      end
*/
*/ **********************************************************************
*/  Add subroutine tbl140
*/ **********************************************************************
*/
*deck t4
      subroutine tbl140
c        print neutron/photon activity of each nuclide in each cell
*call cm
c
c     ****************************************************************
c        print table 140 is the activity of each nuclide in each cell
c          for neutron and/or photon collision statistics
c          current nuclide table types are:
c            type 'c', 'd' & 'm' for neutron events
c            type 'p' & 'g' for photoatomic events
c            type 'u' for photonuclear events
c
c     ****************************************************************
c        local variable declarations
c
c     array for holding sum totals
      parameter (kt=9)
      dimension t(9)
c
c     character storage for cell headers
      character hh(2)*6
c
c     character storage for current table name
      character hn*10
c
c     character storage of common header labels
      parameter (ka=4)
      character ha(2,4)*12
      data  ha/
     1     '  cell',       ' index',
     2     '  cell',       '  name',
     3     '   nuclides',  ' ',
     4     '     atom',    ' fraction'/
c
c     character storage for neutron table header labels
      parameter (kb=8)
      character hb(2,8)*12
      data hb/
     1     '       total', '  collisions',
     2     '  collisions', '    * weight',
     3     '   wgt. lost', '  to capture',
     4     '   wgt. gain', '  by fission',
     5     '   wgt. gain', '   by (n,xn)',
     6     '       tot p', '    produced',
     7     '     wgt. of', '  p produced',
     8     '       avg p', '      energy'/
c
c     character storage for photoatomic table header labels
      parameter (kc=3)
      character hc(2,3)*12
      data hc/
     1     '       total', '  collisions',
     2     '  collisions', '    * weight',
     3     '   wgt. lost', '  to capture'/
c
c     character storage for photonuclear table header labels
      parameter (kd=8)
      character hd(2,8)*12
      data hd/
     1     '       total', '  collisions',
     2     '  collisions', '    * weight',
     3     '       tot p', '    produced',
     4     '     wgt. of', '  p produced',
     5     '       avg p', '      energy',
     6     '       tot n', '    produced',
     7     '     wgt. of', '  n produced',
     8     '       avg n', '      energy'/
c
c     ****************************************************************
c        if there are no table nuclides in problem, don't print
      if(mxe.eq.0)return
c
c     ****************************************************************
c        if neutrons are transported, print the neutron nuclide info
      if(kpt(1).ne.0)then
c
c     ************************************************
c        neutron nuclide table header for print by cell
         write(iuo,'(1h1,a47,a19,50x,a15/)')
     1        'neutron activity of each nuclide in each cell, ',
     2        'per source particle',
     3        'print table 140'
         write(iuo,'(1x,2a6,a11,a9,8a12)')
     1        (ha(1,i),i=1,ka),(hb(1,i),i=1,kb)
         write(iuo,'(1x,2a6,a11,a9,8a12)')
     1        (ha(2,i),i=1,ka),(hb(2,i),i=1,kb)
c
c        initialize the sum totals
```

```
         do 10 i=1,kt
            t(i)=zero
 10      continue
c
c        loop over all cells
         do 20 ix=1,mxa
            im=mat(lmat+ix)
c
c           if void material or zero importance cell, go to next cell
            if(im.eq.0.or.fim(lfim+1,ix).eq.0)go to 20
c
c           set up for next cell print
            write(iuo,'(1x)')
            write(hh(1),'(i6)')ix
            write(hh(2),'(i6)')ncl(lncl+ix)
            ip=ipan(lipa+ix)
c
c           loop over all nuclides
            do 30 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
               it=lme(llme+1,ii)
c
c              if nuclide has no table, go to next nuclide
               if(it.eq.0)go to 30
               call zaid(2,hn,ixl(lixl+1,it))
c
c              print each nuclide in the cell with associated data
               if(pan(lpan+1,7,ip).ne.zero)then
                  e1=pan(lpan+1,8,ip)/pan(lpan+1,7,ip)
               else
                  e1=zero
               endif
               write(iuo,'(1x,2a6,1x,a10,1pe9.2,i12,1p4e12.4,
     1              i12,1p2e12.4)')hh(1),hh(2),hn,fme(lfme+ii),
     2              nint(pan(lpan+1,1,ip)),
     3              fpi*pan(lpan+1,2,ip),
     4              fpi*pan(lpan+1,3,ip),
     5              fpi*pan(lpan+1,4,ip),
     6              fpi*pan(lpan+1,5,ip),
     7              nint(pan(lpan+1,6,ip)),
     8              fpi*pan(lpan+1,7,ip),
     9              e1
c
c              add components to sum total over all cells & nuclides
               do 40 i=1,kb
                  t(i)=t(i)+pan(lpan+1,i,ip)
 40            continue
c
c              don't print cell info for remaining nuclides
               hh(1)=' '
               hh(2)=' '
               ip=ip+1
c
c           end loop over current nuclide
 30         continue
c
c        end loop over cells printing info by nuclide
 20      continue
c
c        print the total over all nuclides and all cells
         if(t(7).ne.zero)then
            e1=t(8)/t(7)
         else
            e1=zero
         endif
         write(iuo,'(/8x,5htotal,20x,i12,1p4e12.4,i12,2e12.4)')
     1        nint(t(1)),(fpi*t(i),i=2,5),nint(t(6)),
     2        fpi*t(7),e1
c
c     ******************************************************
c        compute and print the sum over all cells by nuclide
c
```

```
c        header for totals by nuclide
         write(iuo,'(//1x,a32,8a12)')
     1        'total over all cells by nuclide',
     2        (hb(1,i),i=1,kb)
         write(iuo,'(33x,8a12/)')
     1        (hb(2,i),i=1,kb)
c
c        loop over each nuclide printing totals
         do 50 in=1,mxe
            call zaid(2,hn,ixl(lixl+1,in))
c
c           if not a neutron nuclide, go to next table
            if(hn(10:10).ne.'c'.and.
     1         hn(10:10).ne.'d'.and.
     2         hn(10:10).ne.'m'    )go to 50
c
c           reinitialize the sum totals
            do 60 i=1,kt
               t(i)=zero
 60         continue
c
c           loop over all cells, summing this nuclide
            do 70 ix=1,mxa
               im=mat(lmat+ix)
               ip=ipan(lipa+ix)
c
c              if void material or zero importance cell, go to next cell
               if(im.eq.0.or.fim(lfim+1,ix).eq.0)go to 70
c
c              loop over all nuclides
               do 80 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
                  it=lme(llme+1,ii)
c
c                 if not current nuclide, go to next nuclide
                  if(it.ne.in)go to 80
c
c                 add current cell total to nuclide sum
                  do 90 i=1,kb
                     t(i)=t(i)+pan(lpan+1,i,ip+ii-jmd(ljmd+im))
 90               continue
c
c              end of loop over nuclides in cell
 80            continue
c
c           end of loop over cells
 70         continue
c
c           print the nuclide information summed over all cells
            if(t(7).ne.zero)then
               e1=t(8)/t(7)
            else
               e1=zero
            endif
            write(iuo,'(14x,a10,9x,i12,1p4e12.4,i12,2e12.4)')hn,
     1        nint(t(1)),(fpi*t(i),i=2,5),nint(t(6)),
     2        fpi*t(7),e1
c
c        end summation and print by nuclide over all cells
 50      continue
      endif
c
c     *****************************************************************
c        if photons are transported, print the photoatomic nuclide info
      if(kpt(2).ne.0)then
c
c     ****************************************************
c        photoatomic nuclide table header for print by cell
         write(iuo,'(1h1,a51,a19,46x,a15/)')
     1        'photoatomic activity of each nuclide in each cell, ',
     2        'per source particle',
     3        'print table 140'
```

```
         write(iuo,'(1x,2a6,a11,a9,3a12)')
     1        (ha(1,i),i=1,ka),(hc(1,i),i=1,kc)
         write(iuo,'(1x,2a6,a11,a9,3a12)')
     1        (ha(2,i),i=1,ka),(hc(2,i),i=1,kc)
c
c        initialize the sum totals
         do 110 i=1,kt
            t(i)=zero
 110     continue
c
c        loop over all cells
         do 120 ix=1,mxa
            im=mat(lmat+ix)
c
c           if void material or zero importance cell, go to next cell
            if(im.eq.0.or.fim(lfim+2,ix).eq.0)go to 120
c
c           set up for next cell print
            write(iuo,'(1x)')
            write(hh(1),'(i6)')ix
            write(hh(2),'(i6)')ncl(lncl+ix)
            ip=ipan(lipa+ix)
c
c           loop over all nuclides
            do 130 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
               it=lme(llme+2,ii)
c
c              if nuclide has no table, go to next nuclide
               if(it.eq.0)go to 130
               call zaid(2,hn,ixl(lixl+1,it))
c
c              print each nuclide in the cell with associated data
               write(iuo,'(1x,2a6,1x,a10,1pe9.2,i12,1p3e12.4)')
     1              hh(1),hh(2),hn,fme(lfme+ii),
     2              nint(pan(lpan+2,1,ip)),
     3              fpi*pan(lpan+2,2,ip),
     4              fpi*pan(lpan+2,3,ip)
c
c              add components to sum total over all cells & nuclides
               do 140 i=1,kc
                  t(i)=t(i)+pan(lpan+2,i,ip)
 140           continue
c
c              don't print cell info for remaining nuclides
               hh(1)=' '
               hh(2)=' '
               ip=ip+1
c
c           end loop over current nuclide
 130        continue
c
c        end loop over cells printing info by nuclide
 120     continue
c
c        print the total over all nuclides and all cells
         write(iuo,'(/8x,5htotal,20x,i12,1p2e12.4)')
     1       nint(t(1)),(fpi*t(i),i=2,3)
c
c     *******************************************************
c        compute and print the sum over all cells by nuclide
c
c        header for totals by nuclide
         write(iuo,'(//1x,a32,3a12)')
     1       'total over all cells by nuclide',
     2       (hc(1,i),i=1,kc)
         write(iuo,'(33x,3a12/)')
     1       (hc(2,i),i=1,kc)
c
c        loop over each nuclide printing totals
         do 150 in=1,mxe
            call zaid(2,hn,ixl(lixl+1,in))
```

```
c
c           if not a photoatomic nuclide, go to next table
            if(hn(10:10).ne.'p'.and.
     1         hn(10:10).ne.'g'    )go to 150
c
c           reinitialize the sum totals
            do 160 i=1,kt
               t(i)=zero
 160        continue
c
c           loop over all cells, summing this nuclide
            do 170 ix=1,mxa
               im=mat(lmat+ix)
               ip=ipan(lipa+ix)
c
c              if void material or zero importance cell, go to next cell
               if(im.eq.0.or.fim(lfim+2,ix).eq.0)go to 170
c
c              loop over all nuclides
               do 180 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
                  it=lme(llme+2,ii)
c
c                 if not current nuclide, go to next nuclide
                  if(it.ne.in)go to 180
c
c                 add current cell total to nuclide sum
                  do 190 i=1,kc
                     t(i)=t(i)+pan(lpan+2,i,ip+ii-jmd(ljmd+im))
 190              continue
c
c              end of loop over nuclides in cell
 180           continue
c
c           end of loop over cells
 170        continue
c
c           print the nuclide information summed over all cells
            write(iuo,'(14x,a10,9x,i12,1p2e12.4)')hn,
     1        nint(t(1)),(fpi*t(i),i=2,3)
c
c        end summation and print by nuclide over all cells
 150     continue
      endif
c
c     *****************************************************************
c        if photons are transported and photonuclear physics is on,
c           print the photonuclear nuclide info
      if(kpt(2).ne.0.and.ispn.ne.0)then
c
c     ******************************************************
c        photonuclear nuclide table header for print by cell
         write(iuo,'(1h1,a52,a19,45x,a15/)')
     1        'photonuclear activity of each nuclide in each cell, ',
     2        'per source particle',
     3        'print table 140'
         write(iuo,'(1x,2a6,a11,a9,8a12)')
     1        (ha(1,i),i=1,ka),(hd(1,i),i=1,kd)
         write(iuo,'(1x,2a6,a11,a9,8a12)')
     1        (ha(2,i),i=1,ka),(hd(2,i),i=1,kd)
c
c        initialize the sum totals
         do 210 i=1,kt
            t(i)=zero
 210     continue
c
c        loop over all cells
         do 220 ix=1,mxa
            im=mat(lmat+ix)
c
c           if void material or zero importance cell, go to next cell
            if(im.eq.0.or.fim(lfim+2,ix).eq.0)go to 220
```

```
c
c           set up for next cell print
            nl=1
            write(hh(1),'(i6)')ix
            write(hh(2),'(i6)')ncl(lncl+ix)
            ip=ipan(lipa+ix)
c
c           loop over all nuclides
            do 230 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
               it=lmn(llmn+ii)
c
c              if nuclide has no table, go to next nuclide
               if(it.eq.0)go to 230
               call zaid(2,hn,ixl(lixl+1,it))
c
c              print each nuclide in the cell with associated data
               if(pan(lpan+3,4,ip).ne.zero)then
                  e1=pan(lpan+3,5,ip)/pan(lpan+3,4,ip)
               else
                  e1=zero
               endif
               if(pan(lpan+3,7,ip).ne.zero)then
                  e2=pan(lpan+3,8,ip)/pan(lpan+3,7,ip)
               else
                  e2=zero
               endif
               if (nl.ne.0) then
                    write(iuo,'(1x)')
                    nl=0
               endif
               write(iuo,'(1x,2a6,1x,a10,1pe9.2,i12,
     1                   e12.4,2(i12,2e12.4))')
     2              hh(1),hh(2),hn,fme(lfme+ii),
     3              nint(pan(lpan+3,1,ip)),
     4              fpi*pan(lpan+3,2,ip),
     5              nint(pan(lpan+3,3,ip)),
     6              fpi*pan(lpan+3,4,ip),
     7              e1,
     8              nint(pan(lpan+3,6,ip)),
     9              fpi*pan(lpan+3,7,ip),
     1              e2
c
c              add components to sum total over all cells & nuclides
               do 240 i=1,kd
                  t(i)=t(i)+pan(lpan+3,i,ip)
 240           continue
c
c              don't print cell info for remaining nuclides
               if (nl.eq.0) then
                  hh(1)=' '
                  hh(2)=' '
               endif
               ip=ip+1
c
c           end loop over current nuclide
 230        continue
c
c        end loop over cells printing info by nuclide
 220     continue
c
c        print the total over all nuclides and all cells
         if(t(4).ne.zero)then
            e1=t(5)/t(4)
         else
            e1=zero
         endif
         if(t(7).ne.zero)then
            e2=t(8)/t(7)
         else
            e2=zero
         endif
         write(iuo,'(/8x,5htotal,20x,i12,1pe12.4,2(i12,2e12.4))')
     1        nint(t(1)),fpi*t(2),
     2        nint(t(3)),fpi*t(4),e1,
     3        nint(t(6)),fpi*t(7),e2
c
c     ****************************************************
c        compute and print the sum over all cells by nuclide
c
c        header for totals by nuclide
         write(iuo,'(//1x,a32,8a12)')
     1        'total over all cells by nuclide',
     2        (hd(1,i),i=1,kd)
         write(iuo,'(33x,8a12/)')
     1        (hd(2,i),i=1,kd)
c
c        loop over each nuclide printing totals
         do 250 in=1,mxe
            call zaid(2,hn,ixl(lixl+1,in))
c
c           if not a photonuclear nuclide, go to next table
            if(hn(10:10).ne.'u')go to 250
c
c           reinitialize the sum totals
            do 260 i=1,kt
               t(i)=zero
 260        continue
c
c           loop over all cells, summing this nuclide
            do 270 ix=1,mxa
               im=mat(lmat+ix)
               ip=ipan(lipa+ix)
c
c              if void material or zero importance cell, go to next cell
               if(im.eq.0.or.fim(lfim+2,ix).eq.0)go to 270
c
c              loop over all nuclides
               do 280 ii=jmd(ljmd+im),jmd(ljmd+im+1)-1
                  it=lmn(llmn+ii)
c
c                 if not current nuclide, go to next nuclide
                  if(it.ne.in)go to 280
c
c                 add current cell total to nuclide sum
                  do 290 i=1,kd
                     t(i)=t(i)+pan(lpan+3,i,ip+ii-jmd(ljmd+im))
 290              continue
c
c              end of loop over nuclides in cell
 280           continue
c
c           end of loop over cells
 270        continue
c
c           print the nuclide information summed over all cells
            if(t(4).ne.zero)then
               e1=t(5)/t(4)
            else
               e1=zero
            endif
            if(t(7).ne.zero)then
               e2=t(8)/t(7)
            else
               e2=zero
            endif
            write(iuo,'(14x,a10,9x,i12,1pe12.4,2(i12,2e12.4))')hn,
     1        nint(t(1)),fpi*t(2),
     2        nint(t(3)),fpi*t(4),e1,
     3        nint(t(6)),fpi*t(7),e2
c
c        end summation and print by nuclide over all cells
 250     continue
```

```
      endif
      return
      end
```

APPENDIX D
MISCELLANEOUS DATA FROM VALIDATION STUDIES

Introduction

This appendix contains the MCNP input decks and tabular listings of the numerical data contained in Chapter 4. They are presented in the same order as the figures in the text with one deck and table corresponding to each graph. The input deck for each graph is shown at a particular incident electron energy corresponding to a single point. The input decks for the remaining points are identical except for the value of the source energy. The table lists the energy, reported value and calculated value for each point on the corresponding graph.

The reported values come from the two reports discussed in Chapter 4. Swanson reported his neutron yields per kilowatt second in tabular form. These numbers have been converted to neutrons per electron. The reported values are listed as (value ± estimated error) with a 20 percent estimated error as given by Swanson. The Barber and George results were reported as neutrons per electron but only graphically. Their figures were digitized and the best estimate of energy-yield pairs is presented in the table. Reported values are listed as above except the estimated error is 15 percent as given by Barber and George. The calculations were performed using the prototype code version MCNP4BPN as described in the text. The calculated values are listed as (value ± one sigma absolute error). Note that the calculated uncertainty is a measure of the precision of the Monte Carlo simulation. It does not include an estimate of the uncertainty in the data or the model.

The last three input decks and tables contain the information used to study the effects of variations in actual versus reported parameters for the Barber and George study. The baseline case is the Ta-I target at the 28.3 MeV incident beam energy. Incident beam energy, target thickness and beam radius were all varied over the range ±10 percent. For variations in beam radius and target thickness, change the appropriate parameter to create the variations of the input deck.

“Semi-infinite” Aluminum Target (Al-XX)

```
Neutron emissions per electron incident on an Al target.
1   1   -2.699   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    177.9
11 py   -177.9
12 py    177.9
21 pz   -177.9
22 pz    177.9

mode e p n
m1   13027  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=20      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  8.2721
cut:e   j  8.2721
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-1. Reported and calculated yields for a “semi-infinite” aluminum target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
15[a]	$4.59 \cdot 10^{7}$	$(1.10 \pm 0.22) \cdot 10^{-7}$	$(5.3509 \pm 0.0701) \cdot 10^{-7}$	4.85
20	$8.52 \cdot 10^{9}$	$(2.73 \pm 0.55) \cdot 10^{-5}$	$(3.7381 \pm 0.0478) \cdot 10^{-5}$	1.37
25	$5.07 \cdot 10^{10}$	$(2.03 \pm 0.41) \cdot 10^{-4}$	$(2.1565 \pm 0.0151) \cdot 10^{-4}$	1.06
34	$1.61 \cdot 10^{11}$	$(8.77 \pm 1.75) \cdot 10^{-4}$	$(8.9338 \pm 0.0313) \cdot 10^{-4}$	1.02
50	$3.13 \cdot 10^{11}$	$(2.51 \pm 0.50) \cdot 10^{-3}$	$(2.5247 \pm 0.0048) \cdot 10^{-3}$	1.01
100	$5.14 \cdot 10^{11}$	$(8.24 \pm 1.65) \cdot 10^{-3}$	$(8.2647 \pm 0.0083) \cdot 10^{-3}$	1.00

[a] In order to achieve acceptable statistics, it was necessary to run the 15 MeV case for 40,000,000 particles.

“Semi-infinite” Iron Target (Fe-XX)

```
Neutron emissions per electron incident on an Fe target.
1   1   -7.875   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    35.15
11 py   -35.15
12 py    35.15
21 pz   -35.15
22 pz    35.15

mode  e p n
m1   26056  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=15      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  7.6142
cut:e   j  7.6142
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-2. Reported and calculated yields for a “semi-infinite” iron target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
15	$1.13\cdot10^{10}$	$(2.72\pm0.54)\cdot10^{-5}$	$(4.1825\pm0.0339)\cdot10^{-5}$	1.54
20	$9.65\cdot10^{10}$	$(3.09\pm0.62)\cdot10^{-4}$	$(4.3125\pm0.0160)\cdot10^{-4}$	1.39
25	$2.42\cdot10^{11}$	$(9.69\pm1.94)\cdot10^{-4}$	$(1.2558\pm0.0030)\cdot10^{-3}$	1.30
34	$4.31\cdot10^{11}$	$(2.35\pm0.47)\cdot10^{-3}$	$(3.1284\pm0.0044)\cdot10^{-3}$	1.33
50	$6.02\cdot10^{11}$	$(4.82\pm0.96)\cdot10^{-3}$	$(6.4964\pm0.0058)\cdot10^{-3}$	1.35
100	$7.62\cdot10^{11}$	$(1.22\pm0.24)\cdot10^{-2}$	$(1.6392\pm0.0010)\cdot10^{-2}$	1.34

“Semi-infinite” Copper Target (Cu-XX)

```
Neutron emissions per electron incident on a Cu target.
1   1   -8.96    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    28.7
11 py   -28.7
12 py    28.7
21 pz   -28.7
22 pz    28.7

mode e p n
m1   29063  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=15      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  5.7775
cut:e   j  5.7775
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-3. Reported and calculated yields for a “semi-infinite” copper target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
15	$2.00\cdot10^{10}$	$(4.81\pm0.96)\cdot10^{-5}$	$(4.5743\pm0.0389)\cdot10^{-5}$	0.95
20	$1.56\cdot10^{11}$	$(5.00\pm1.00)\cdot10^{-4}$	$(4.7199\pm0.0179)\cdot10^{-4}$	0.94
25	$3.54\cdot10^{11}$	$(1.42\pm0.28)\cdot10^{-3}$	$(1.3070\pm0.0031)\cdot10^{-3}$	0.92
34	$6.35\cdot10^{11}$	$(3.46\pm0.69)\cdot10^{-3}$	$(3.1881\pm0.0045)\cdot10^{-3}$	0.92
50	$8.76\cdot10^{11}$	$(7.02\pm1.40)\cdot10^{-3}$	$(6.4424\pm0.0064)\cdot10^{-3}$	0.92
100	$1.09\cdot10^{12}$	$(1.75\pm0.35)\cdot10^{-2}$	$(1.5892\pm0.0010)\cdot10^{-2}$	0.91

“Semi-infinite” Tantalum Target (Ta-XX)

```
Neutron emissions per electron incident on a Ta target.
1   1   -16.6    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    8.217
11 py   -8.217
12 py    8.217
21 pz   -8.217
22 pz    8.217

mode  e p n
m1   73181  1  elib=01e plib=02p nlib=60c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=10      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-4. Reported and calculated yields for a “semi-infinite” tantalum target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
10	$1.06 \cdot 10^{10}$	$(1.70 \pm 0.34) \cdot 10^{-5}$	$(1.8431 \pm 0.0162) \cdot 10^{-5}$	1.09
15	$3.07 \cdot 10^{11}$	$(7.38 \pm 1.48) \cdot 10^{-4}$	$(6.5321 \pm 0.0202) \cdot 10^{-4}$	0.89
20	$8.80 \cdot 10^{11}$	$(2.82 \pm 0.56) \cdot 10^{-3}$	$(2.6127 \pm 0.0047) \cdot 10^{-3}$	0.93
25	$1.32 \cdot 10^{12}$	$(5.29 \pm 1.06) \cdot 10^{-3}$	$(4.9969 \pm 0.0060) \cdot 10^{-3}$	0.95
34	$1.68 \cdot 10^{12}$	$(9.15 \pm 1.83) \cdot 10^{-3}$	$(9.1243 \pm 0.0073) \cdot 10^{-3}$	1.00
50	$1.90 \cdot 10^{12}$	$(1.52 \pm 0.30) \cdot 10^{-2}$	$(1.5752 \pm 0.0009) \cdot 10^{-2}$	1.03
100	$2.04 \cdot 10^{12}$	$(3.27 \pm 0.65) \cdot 10^{-2}$	$(3.4548 \pm 0.0014) \cdot 10^{-2}$	1.06

“Semi-infinite” Tungsten Target (W-XX)

```
Neutron emissions per electron incident on a W target.
1   1   -19.3    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    7.005
11 py   -7.005
12 py    7.005
21 pz   -7.005
22 pz    7.005

mode e p n
m1   74184  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=10      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-5. Reported and calculated yields for a “semi-infinite” tungsten target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
10	$3.12\cdot10^{10}$	$(5.00\pm1.00)\cdot10^{-5}$	$(1.9389\pm0.0147)\cdot10^{-5}$	0.39
15	$3.61\cdot10^{11}$	$(8.68\pm1.74)\cdot10^{-4}$	$(6.5626\pm0.0203)\cdot10^{-4}$	0.76
20	$1.00\cdot10^{12}$	$(3.20\pm0.64)\cdot10^{-3}$	$(2.7477\pm0.0049)\cdot10^{-3}$	0.86
25	$1.50\cdot10^{12}$	$(6.01\pm1.20)\cdot10^{-3}$	$(5.2763\pm0.0063)\cdot10^{-3}$	0.88
34	$1.92\cdot10^{12}$	$(1.05\pm0.21)\cdot10^{-2}$	$(9.5910\pm0.0077)\cdot10^{-3}$	0.92
50	$2.17\cdot10^{12}$	$(1.74\pm0.35)\cdot10^{-2}$	$(1.6518\pm0.0010)\cdot10^{-2}$	0.95
100	$2.33\cdot10^{12}$	$(3.73\pm0.75)\cdot10^{-2}$	$(3.6195\pm0.0014)\cdot10^{-2}$	0.97

“Semi-infinite” Lead Target (Pb-XX)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    11.22
11 py   -11.22
12 py    11.22
21 pz   -11.22
22 pz    11.22

mode e p n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 vec=1 0 0 dir=1 par=3 erg=10      $ <-- Incident erg.
c
c
fcl:p   1  0
phys:p 3j -1
phys:n  j  150
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 200
print
```

Table D-6. Reported and calculated yields for a “semi-infinite” lead target.

Energy (MeV)	Reported Yield (n / kW / s)	Reported Yield (n / e)	Calculated Yield (n / e)	Calc. Yield / Rept. Yield
10	$2.01\cdot10^{10}$	$(3.22\pm0.64)\cdot10^{-5}$	$(2.8299\pm0.0175)\cdot10^{-5}$	0.88
15	$4.09\cdot10^{11}$	$(9.83\pm1.97)\cdot10^{-4}$	$(7.4992\pm0.0217)\cdot10^{-4}$	0.76
20	$1.02\cdot10^{12}$	$(3.27\pm0.65)\cdot10^{-3}$	$(2.7242\pm0.0044)\cdot10^{-3}$	0.83
25	$1.43\cdot10^{12}$	$(5.73\pm1.15)\cdot10^{-3}$	$(4.8769\pm0.0054)\cdot10^{-3}$	0.85
34	$1.77\cdot10^{12}$	$(9.64\pm1.93)\cdot10^{-3}$	$(8.5571\pm0.0068)\cdot10^{-3}$	0.89
50	$1.97\cdot10^{12}$	$(1.58\pm0.32)\cdot10^{-2}$	$(1.4505\pm0.0009)\cdot10^{-2}$	0.92
100	$2.10\cdot10^{12}$	$(3.36\pm0.67)\cdot10^{-2}$	$(3.1491\pm0.0013)\cdot10^{-2}$	0.94

One Radiation-Length Thick Aluminum Target (Al-I)

```
Neutron emissions per electron incident on a Al target.
1   1   -2.699   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    8.96
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   13027  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=22.2  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  8.2721
cut:e   j  8.2721
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-7. Reported and calculated yields for an approximately one radiation-length thick aluminum target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
22.2	$(4.60\pm0.69)\cdot10^{-5}$	$(3.80366\pm0.03537)\cdot10^{-5}$	0.83
28.3	$(2.10\pm0.32)\cdot10^{-4}$	$(1.64582\pm0.00823)\cdot10^{-4}$	0.78
34.3	$(4.30\pm0.65)\cdot10^{-4}$	$(3.40027\pm0.01122)\cdot10^{-4}$	0.79

One Radiation-Length Thick Copper Target (Cu-I)

```
Neutron emissions per electron incident on a Cu target.
1   1   -8.96    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    1.48
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   29063  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=16.1  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  5.7775
cut:e   j  5.7775
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-8. Reported and calculated yields for an approximately one radiation-length thick copper target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
16.1	$(3.00\pm0.45)\cdot10^{-5}$	$(3.20974\pm0.02118)\cdot10^{-5}$	1.07
21.2	$(2.60\pm0.39)\cdot10^{-4}$	$(2.18209\pm0.00698)\cdot10^{-4}$	0.84
28.3	$(8.20\pm1.23)\cdot10^{-4}$	$(6.26824\pm0.01128)\cdot10^{-4}$	0.76
34.3	$(1.29\pm0.19)\cdot10^{-3}$	$(9.63159\pm0.01348)\cdot10^{-4}$	0.75
35.5	$(1.39\pm0.21)\cdot10^{-3}$	$(1.01545\pm0.00132)\cdot10^{-3}$	0.73

Two Radiation-Length Thick Copper Target (Cu-II)

```
Neutron emissions per electron incident on a Cu target.
1   1   -8.96    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    2.96
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   29063  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=16.1  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  5.7775
cut:e   j  5.7775
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-9. Reported and calculated yields for an approximately two radiation-length thick copper target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
16.1	$(5.00\pm0.75)\cdot10^{-5}$	$(5.37943\pm0.03550)\cdot10^{-5}$	1.08
21.2	$(4.30\pm0.65)\cdot10^{-4}$	$(3.73710\pm0.01196)\cdot10^{-4}$	0.87
28.3	$(1.39\pm0.21)\cdot10^{-3}$	$(1.12271\pm0.00202)\cdot10^{-3}$	0.81
34.3	$(2.37\pm0.36)\cdot10^{-3}$	$(1.80496\pm0.00253)\cdot10^{-3}$	0.76

Three Radiation-Length Thick Copper Target (Cu-III)

```
Neutron emissions per electron incident on a Cu target.
1   1   -8.96    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    4.45
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e p n
m1   29063  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=16.1  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  5.7775
cut:e   j  5.7775
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-10. Reported and calculated yields for an approximately three radiation-length thick copper target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
16.1	$(7.00\pm1.05)\cdot10^{-5}$	$(6.76092\pm0.04462)\cdot10^{-5}$	0.97
21.2	$(5.30\pm0.80)\cdot10^{-4}$	$(4.71127\pm0.01508)\cdot10^{-4}$	0.89
28.3	$(1.80\pm0.27)\cdot10^{-3}$	$(1.43027\pm0.00257)\cdot10^{-3}$	0.79
34.3	$(2.93\pm0.44)\cdot10^{-3}$	$(2.32709\pm0.00326)\cdot10^{-3}$	0.79

Four Radiation-Length Thick Copper Target (Cu-IV)

```
Neutron emissions per electron incident on a Cu target.
1   1   -8.96    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    5.93
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   29063  1  elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=16.1  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  5.7775
cut:e   j  5.7775
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-11. Reported and calculated yields for an approximately four radiation-length thick copper target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
16.1	$(1.00\pm0.15)\cdot10^{-5}$	$(7.62722\pm0.05034)\cdot10^{-5}$	0.76
21.2	$(6.00\pm0.90)\cdot10^{-4}$	$(5.31440\pm0.01701)\cdot10^{-4}$	0.89
28.3	$(2.13\pm0.32)\cdot10^{-3}$	$(1.61939\pm0.00291)\cdot10^{-3}$	0.76
34.3	$(3.35\pm0.50)\cdot10^{-3}$	$(2.64744\pm0.00371)\cdot10^{-3}$	0.79

One Radiation-Length Thick Tantalum Target (Ta-I)

```
Neutron emissions per electron incident on a Ta target.
1   1   -16.6    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    0.374
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   73181  1  elib=01e plib=02p nlib=60c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=10.3  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-12. Reported and calculated yields for an approximately one radiation-length thick tantalum target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
10.3	$(8.00\pm1.20)\cdot10^{-5}$	$(6.88482\pm0.05095)\cdot10^{-6}$	0.09
18.7	$(5.20\pm0.78)\cdot10^{-4}$	$(5.20541\pm0.01041)\cdot10^{-4}$	1.00
28.3	$(1.38\pm0.21)\cdot10^{-3}$	$(1.39451\pm0.00153)\cdot10^{-3}$	1.01
34.3	$(1.81\pm0.27)\cdot10^{-3}$	$(1.70717\pm0.00171)\cdot10^{-3}$	0.94

One Radiation-Length Thick Lead Target (Pb-I)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    0.518
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=18.7  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-13. Reported and calculated yields for an approximately one radiation-length thick lead target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
18.7	$(7.30\pm1.10)\cdot10^{-4}$	$(5.62755\pm0.01126)\cdot10^{-4}$	0.77
28.3	$(1.69\pm0.25)\cdot10^{-3}$	$(1.33409\pm0.00160)\cdot10^{-3}$	0.79
34.5	$(2.12\pm0.32)\cdot10^{-3}$	$(1.60021\pm0.00160)\cdot10^{-3}$	0.75

Two Radiation-Length Thick Lead Target (Pb-II)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    1.01
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=18.7  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-14. Reported and calculated yields for an approximately two radiation-length thick lead target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
18.7	$(1.32\pm0.20)\cdot10^{-3}$	$(1.02040\pm0.00194)\cdot10^{-3}$	0.77
28.3	$(3.45\pm0.52)\cdot10^{-3}$	$(2.78846\pm0.00279)\cdot10^{-3}$	0.81
34.5	$(4.72\pm0.71)\cdot10^{-3}$	$(3.66458\pm0.00293)\cdot10^{-3}$	0.78

Three Radiation-Length Thick Lead Target (Pb-III)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    1.52
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=18.7  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-15. Reported and calculated yields for an approximately three radiation-length thick lead target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
18.7	$(1.77\pm0.27)\cdot10^{-3}$	$(1.34637\pm0.00256)\cdot10^{-3}$	0.76
28.3	$(4.69\pm0.70)\cdot10^{-3}$	$(3.81410\pm0.00381)\cdot10^{-3}$	0.81
34.5	$(6.46\pm0.97)\cdot10^{-3}$	$(5.14784\pm0.00412)\cdot10^{-3}$	0.80

Four Radiation-Length Thick Lead Target (Pb-IV)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    2.02
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e p n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=18.7  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-16. Reported and calculated yields for an approximately four radiation-length thick lead target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
18.7	$(2.10\pm0.32)\cdot10^{-3}$	$(1.56998\pm0.00283)\cdot10^{-3}$	0.75
28.3	$(5.37\pm0.81)\cdot10^{-3}$	$(4.51358\pm0.00451)\cdot10^{-3}$	0.84
34.5	$(7.77\pm1.17)\cdot10^{-3}$	$(6.16382\pm0.00493)\cdot10^{-3}$	0.79

Six Radiation-Length Thick Lead Target (Pb-VI)

```
Neutron emissions per electron incident on a Pb target.
1   1   -11.35   1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    3.03
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   82206 24.1  82207 22.1  82208 52.4
     elib=01e plib=02p nlib=22c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=18.7  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  6.5
cut:e   j  6.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 90
print
```

Table D-17. Reported and calculated yields for an approximately six radiation-length thick lead target.

Energy (MeV)	Reported Yield (n / e)	Calculated Yield (n / e)	Calculated Yield / Reported Yield
18.7	$(2.50\pm0.38)\cdot10^{-3}$	$(1.84432\pm0.00332)\cdot10^{-3}$	0.74
28.3	$(6.67\pm1.00)\cdot10^{-3}$	$(5.36690\pm0.00537)\cdot10^{-3}$	0.80
34.5	$(9.00\pm1.35)\cdot10^{-3}$	$(7.40813\pm0.00593)\cdot10^{-3}$	0.82

Variation of Beam Energy

```
Neutron emissions per electron incident on a Ta target.
1   1   -16.6    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    0.374
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode  e  p  n
m1   73181  1  elib=01e plib=02p nlib=60c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=28.3  $ <-- Incident erg.
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 50
print
```

Table D-18. Effect of changes in beam energy over a ten percent variation.

Energy (MeV)	Energy Percent Variation	Calculated Yield (n / e)	Calculate Yield / Baseline Yield
31.13	-10%	$(1.55991\pm0.00203)\cdot10^{-3}$	1.11861
30.281	-7%	$(1.51520\pm0.00182)\cdot10^{-3}$	1.08655
29.715	-5%	$(1.48466\pm0.00178)\cdot10^{-3}$	1.06465
29.149	-3%	$(1.44717\pm0.00174)\cdot10^{-3}$	1.03776
28.866	-2%	$(1.43089\pm0.00172)\cdot10^{-3}$	1.02609
28.583	-1%	$(1.41444\pm0.00170)\cdot10^{-3}$	1.01429
28.3	Baseline	$(1.39451\pm0.00153)\cdot10^{-3}$	---------
28.017	+1%	$(1.37554\pm0.00151)\cdot10^{-3}$	0.98640
27.734	+2%	$(1.36208\pm0.00150)\cdot10^{-3}$	0.97674
27.451	+3%	$(1.34194\pm0.00148)\cdot10^{-3}$	0.96230
26.885	+5%	$(1.29716\pm0.00143)\cdot10^{-3}$	0.93019
26.319	+7%	$(1.25727\pm0.00138)\cdot10^{-3}$	0.90159
25.47	+10%	$(1.19254\pm0.00131)\cdot10^{-3}$	0.85517

Variation of Target Thickness

```
Neutron emissions per electron incident on a Ta target.
1   1   -16.6    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    0.374  $ <-- Target thickness
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   73181  1  elib=01e plib=02p nlib=60c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=28.3
si1  0.635
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 50
print
```

Table D-19. Effect of changes in target thickness over a ten percent variation.

Target Thickness (cm)	Thickness Percent Variation	Calculated Yield (n / e)	Calculate Yield / Baseline Yield
0.4114	-10%	$(1.57986 \pm 0.00190) \cdot 10^{-3}$	1.13291
0.40018	-7%	$(1.52489 \pm 0.00183) \cdot 10^{-3}$	1.09350
0.3927	-5%	$(1.48796 \pm 0.00179) \cdot 10^{-3}$	1.06701
0.38522	-3%	$(1.45059 \pm 0.00174) \cdot 10^{-3}$	1.04021
0.38148	-2%	$(1.43197 \pm 0.00172) \cdot 10^{-3}$	1.02686
0.37774	-1%	$(1.41321 \pm 0.00155) \cdot 10^{-3}$	1.01341
0.374	Baseline	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	---------
0.37026	+1%	$(1.37553 \pm 0.00151) \cdot 10^{-3}$	0.98639
0.36652	+2%	$(1.35660 \pm 0.00149) \cdot 10^{-3}$	0.97281
0.36278	+3%	$(1.33756 \pm 0.00147) \cdot 10^{-3}$	0.95916
0.3553	+5%	$(1.29953 \pm 0.00143) \cdot 10^{-3}$	0.93189
0.34782	+7%	$(1.26117 \pm 0.00139) \cdot 10^{-3}$	0.90438
0.3366	+10%	$(1.20330 \pm 0.00132) \cdot 10^{-3}$	0.86288

Variation of Beam Radius

```
Neutron emissions per electron incident on a Ta target.
1   1   -16.6    1 -2   11 -12   21 -22    imp:e,n,p=1
2   0           -1: 2: -11: 12: -21: 22    imp:e,n,p=0

1  px    0
2  px    0.374
11 py   -5.715
12 py    5.715
21 pz   -5.715
22 pz    5.715

mode e p n
m1   73181  1  elib=01e plib=02p nlib=60c pnlib=03n
sdef pos=0 0 0 sur=1 rad=d1 vec=1 0 0 dir=1 par=3 erg=28.3
si1  0.635   $ <-- Beam radius
c
c
fcl:p   1  0
phys:p 3j -1
cut:n  2j  0 0
cut:p   j  7.5
cut:e   j  7.5
c
c
f1:n   1 2 11 12 21 22 (1 2 11 12 21 22)
tf1    7
c
c
nps  2500000
ctme 50
print
```

Table D-20. Effect of changes in beam radius over a ten percent variation.

Beam Radius (cm)	Radius Percent Variation	Calculated Yield (n / e)	Calculate Yield / Baseline Yield
0.6985	-10%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.67945	-7%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.66675	-5%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.65405	-3%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.6477	-2%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.64135	-1%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.635	Baseline	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	---------
0.62865	+1%	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	1.00000
0.6223	+2%	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	1.00000
0.61595	+3%	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	1.00000
0.60325	+5%	$(1.39451 \pm 0.00153) \cdot 10^{-3}$	1.00000
0.59055	+7%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999
0.5715	+10%	$(1.39450 \pm 0.00153) \cdot 10^{-3}$	0.99999

APPENDIX E
MISCELLANEOUS DATA FROM APPLICATION STUDIES

Introduction

This chapter contains descriptions from the MCNP input decks used to perform the applications studies. They are labeled according to their appropriate use. A full listing of all input decks would duplicate many of these sub-sections therefore they are listed individually.

Activation Calculations

Geometry With Block and Ingot

```
c *******************
c  Cell Descriptions
c *******************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199  17 -0.001225 102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291  17 -0.001225 312  -311    214  -211
  298  17 -0.001225 319  -313    215  -212
  299  17 -0.001225 319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391  17 -0.001225 321  -111    233  -219
  392  17 -0.001225 111  -112    231  -219
  393  17 -0.001225 112  -113    232  -219
  394  17 -0.001225 113  -121    239  -219
c
c  Air surrounding target and filters
  399  17 -0.001225       121    239  -201
                    410  -412    420  -422
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402  17 -0.001225 ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412  17 -0.001225 ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422  17 -0.001225 ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432  17 -0.001225 ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
c  Gold ingot at isocenter
  501  19  -19.32    -511  512    513 -514    515 -516
  502  20   -1.12    -521  511    523 -524    525 -526
  503  20   -1.12    -512  522    523 -524    525 -526
  598  17 -0.001225            ( -513: 514 : -515: 516 )
                    (-511  512    523 -524    525 -526 )
  599  17 -0.001225 ( 521:-522 : -523: 524 : -525: 526 )
                    (-401  599    410 -412    420 -422 )
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
```

```
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644  17 -0.001225  630 -632  -656 662  -601 603
  645  17 -0.001225  630 -631  -654 656  -601 603
  646  17 -0.001225  628 -631  -651 654  -601 603
c
c  Ingot in maze
c
  670  19 -19.32      670 -671    672 -673    674 -675
  647  17 -0.001225 (-670: 671:  -672: 673:  -674: 675 )
                    ( 628 -629   -654  659   -601  603 )
c
  648  17 -0.001225  626 -629  -659 661  -601 603
  649  17 -0.001225  626 -627  -655 659  -601 603
  650  17 -0.001225  625 -626  -652 661  -601 603
  651  17 -0.001225  624 -625  -653 660  -602 603
  652  17 -0.001225  623 -624  -653 657  -602 603
  653  17 -0.001225  623 -624  -658 660  -602 603
  654  17 -0.001225  622 -623  -653 660  -602 603
  655  17 -0.001225  621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656  17 -0.001225  623 -624  -657 658  -602 201
  657  17 -0.001225  623 -624  -657 658  -599 605
  658  17 -0.001225  422 -624  -657 658  -201 599
  659  17 -0.001225  420 -422  -657 412  -201 599
  660  17 -0.001225  420 -422  -410 658  -201 599
  661  17 -0.001225  623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                 -606
99996   0            606 -600  -620
99997   0            606 -600        632
```

```
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c ***********************
c  END Cell Descriptions
c ***********************
c

c
c **********************
c  Surface Descriptions
c **********************
c
c --------------------------------------
c  Target, Primary Collimator & Filters
c --------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c  Collimator opening set for a 10x10 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  461  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.0000000E-03
  465  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.5001000E+01
c
```

```
  470  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  471  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.0000000E-03
  475  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -2.7100000E+01
  481  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -2.7100000E+01
  491  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  511  pz   -99.91819559
  512  pz  -100.08180441
  513  px    -1.2
  514  px     1.2
  515  py    -2.05
  516  py     2.05
c
  521  pz   -92.0
  522  pz  -108.0
  523  px   -15.0
  524  px    15.0
  525  py   -15.0
  526  py    15.0
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
  670  px   564
  671  px   564.163608812
  672  py    30.95
  673  py    35.05
  674  pz  -106.7
  675  pz  -104.3
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Geometry Without Block But With Ingot

```
c *******************
c  Cell Descriptions
c *******************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199  17 -0.001225 102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291  17 -0.001225 312  -311    214  -211
  298  17 -0.001225 319  -313    215  -212
  299  17 -0.001225 319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391  17 -0.001225 321  -111    233  -219
  392  17 -0.001225 111  -112    231  -219
  393  17 -0.001225 112  -113    232  -219
  394  17 -0.001225 113  -121    239  -219
c
c  Air surrounding target and filters
  399  17 -0.001225       121    239  -201
                    410  -412    420  -422
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402  17 -0.001225 ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412  17 -0.001225 ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422  17 -0.001225 ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432  17 -0.001225 ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
c  Gold ingot at isocenter
  501  19  -19.32    -511  512    513 -514    515 -516
  599  17 -0.001225 ( 511:-512 : -513: 514 : -515: 516 )
                    (-401  599    410 -412    420 -422 )
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
```

```
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644  17 -0.001225  630 -632  -656 662  -601 603
  645  17 -0.001225  630 -631  -654 656  -601 603
  646  17 -0.001225  628 -631  -651 654  -601 603
c
c  Ingot in maze
c
  670  19 -19.32      670 -671    672 -673    674 -675
  647  17 -0.001225 (-670: 671:  -672: 673:  -674: 675 )
                    ( 628 -629   -654  659   -601  603 )
c
  648  17 -0.001225  626 -629  -659 661  -601 603
  649  17 -0.001225  626 -627  -655 659  -601 603
  650  17 -0.001225  625 -626  -652 661  -601 603
  651  17 -0.001225  624 -625  -653 660  -602 603
  652  17 -0.001225  623 -624  -653 657  -602 603
  653  17 -0.001225  623 -624  -658 660  -602 603
  654  17 -0.001225  622 -623  -653 660  -602 603
  655  17 -0.001225  621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656  17 -0.001225  623 -624  -657 658  -602 201
  657  17 -0.001225  623 -624  -657 658  -599 605
  658  17 -0.001225  422 -624  -657 658  -201 599
  659  17 -0.001225  420 -422  -657 412  -201 599
  660  17 -0.001225  420 -422  -410 658  -201 599
  661  17 -0.001225  623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                 -606
```

```
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c ***********************
c  END Cell Descriptions
c ***********************
c

c
c **********************
c  Surface Descriptions
c **********************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c  Collimator opening set for a 10x10 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  461  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.0000000E-03
  465  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.5001000E+01
c
  470  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  471  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.0000000E-03
  475  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -2.7100000E+01
  481  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -2.7100000E+01
  491  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  511  pz   -99.91819559
  512  pz  -100.08180441
  513  px    -1.2
  514  px     1.2
  515  py    -2.05
  516  py     2.05
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
  670  px   564
  671  px   564.163608812
  672  py    30.95
  673  py    35.05
  674  pz  -106.7
  675  pz  -104.3
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Geometry With Block But Without Ingot

```
c *******************
c  Cell Descriptions
c *******************
c
c --------------------------------------
c  Target, Primary Collimator & Filters
c --------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199  17 -0.001225 102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291  17 -0.001225 312  -311    214  -211
  298  17 -0.001225 319  -313    215  -212
  299  17 -0.001225 319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
```

```
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391  17 -0.001225 321  -111    233  -219
  392  17 -0.001225 111  -112    231  -219
  393  17 -0.001225 112  -113    232  -219
  394  17 -0.001225 113  -121    239  -219
c
c  Air surrounding target and filters
  399  17 -0.001225       121    239  -201
                    410  -412    420  -422
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402  17 -0.001225 ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412  17 -0.001225 ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422  17 -0.001225 ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432  17 -0.001225 ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
c  A-150 plastic block at isocenter
  501  20 -1.12      -521  522    523 -524    525 -526
  599  17 -0.001225 ( 521:-522 : -523: 524 : -525: 526 )
                    (-401  599    410 -412    420 -422 )
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644  17 -0.001225  630 -632  -656 662  -601 603
  645  17 -0.001225  630 -631  -654 656  -601 603
  646  17 -0.001225  628 -631  -651 654  -601 603
  647  17 -0.001225  628 -629  -654 659  -601 603
c
  648  17 -0.001225  626 -629  -659 661  -601 603
  649  17 -0.001225  626 -627  -655 659  -601 603
  650  17 -0.001225  625 -626  -652 661  -601 603
  651  17 -0.001225  624 -625  -653 660  -602 603
  652  17 -0.001225  623 -624  -653 657  -602 603
  653  17 -0.001225  623 -624  -658 660  -602 603
  654  17 -0.001225  622 -623  -653 660  -602 603
  655  17 -0.001225  621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656  17 -0.001225  623 -624  -657 658  -602 201
  657  17 -0.001225  623 -624  -657 658  -599 605
  658  17 -0.001225  422 -624  -657 658  -201 599
  659  17 -0.001225  420 -422  -657 412  -201 599
  660  17 -0.001225  420 -422  -410 658  -201 599
  661  17 -0.001225  623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                -606
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c **********************
c  END Cell Descriptions
c **********************
c
```

```
c
c *********************
c  Surface Descriptions
c *********************
c
c ------------------------------------
c  Target, Primary Collimator & Filters
c ------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c  Collimator opening set for a 10x10 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  461  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.0000000E-03
  465  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.5001000E+01
c
  470  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  471  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.0000000E-03
  475  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -2.7100000E+01
  481  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -2.7100000E+01
  491  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  521  pz   -92.0
  522  pz  -108.0
  523  px   -15.0
  524  px    15.0
  525  py   -15.0
  526  py    15.0
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Geometry Without Block or Ingot

```
c *******************
c  Cell Descriptions
c *******************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199  17 -0.001225 102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291  17 -0.001225 312  -311    214  -211
  298  17 -0.001225 319  -313    215  -212
  299  17 -0.001225 319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391  17 -0.001225 321  -111    233  -219
  392  17 -0.001225 111  -112    231  -219
  393  17 -0.001225 112  -113    232  -219
  394  17 -0.001225 113  -121    239  -219
c
```

```
c  Air surrounding target and filters
  399  17 -0.001225       121    239  -201
                    410  -412    420  -422
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402  17 -0.001225 ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412  17 -0.001225 ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422  17 -0.001225 ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432  17 -0.001225 ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  599  17 -0.001225  -401 599  410 -412  420 -422
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644  17 -0.001225  630 -632  -656 662  -601 603
  645  17 -0.001225  630 -631  -654 656  -601 603
  646  17 -0.001225  628 -631  -651 654  -601 603
  647  17 -0.001225  628 -629  -654 659  -601 603
  648  17 -0.001225  626 -629  -659 661  -601 603
  649  17 -0.001225  626 -627  -655 659  -601 603
  650  17 -0.001225  625 -626  -652 661  -601 603
  651  17 -0.001225  624 -625  -653 660  -602 603
  652  17 -0.001225  623 -624  -653 657  -602 603
  653  17 -0.001225  623 -624  -658 660  -602 603
  654  17 -0.001225  622 -623  -653 660  -602 603
  655  17 -0.001225  621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656  17 -0.001225  623 -624  -657 658  -602 201
  657  17 -0.001225  623 -624  -657 658  -599 605
  658  17 -0.001225  422 -624  -657 658  -201 599
  659  17 -0.001225  420 -422  -657 412  -201 599
  660  17 -0.001225  420 -422  -410 658  -201 599
  661  17 -0.001225  623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                -606
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c ***********************
c  END Cell Descriptions
c ***********************
c

c
c **********************
c  Surface Descriptions
c **********************
c
c --------------------------------------
c  Target, Primary Collimator & Filters
c --------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
```

```
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631          -1
  312  kz   -7.84   0.416           -1
  313  kz   -9.94   1.653           -1
  319  kz  -12.32   0.1914          +1
c
  321  kz  -13.26   1.83376736112  -1
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c  Collimator opening set for a 10x10 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  461  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.0000000E-03
  465  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.5001000E+01
c
  470  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  471  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.0000000E-03
  475  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -2.7100000E+01
  481  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -2.7100000E+01
  491  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  599  pz  -130.0
c
c ------
c  Room
```

```
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
c *************************
c  END Surface Descriptions
c *************************
c
```

Electron-Photoatomic Description of the Materials

```
m11     plib=02p  elib=03e
        74000 -0.90  75000 -0.10
m12     plib=02p  elib=03e
        29000 1
m13     plib=02p  elib=03e
        74000 -0.95  28000 -0.035  29000 -0.015
m14     plib=02p  elib=03e
```

```
        13027 1
m15     plib=02p  elib=03e
        26000 -0.70  24000 -0.18  28000 -0.09
        25000 -0.02  14000 -0.01
m16     plib=02p  elib=03e
        82000 1
m17     plib=02p  elib=03e
        7000 0.784403  8000 0.210747  18000 0.004691
        6000 0.000159
m18     plib=02p  elib=03e
         1000 -0.0055   8000 -0.4984  14000 -0.3157
        20000 -0.0826  11000 -0.017   12000 -0.0026
        13000 -0.0455  16000 -0.0013  19000 -0.0191
        26000 -0.0123
m19     plib=02p  elib=03e
        79000 1
m20     plib=02p  elib=03e
        6000 -0.768   1000 -0.102  8000 -0.059
        7000 -0.036  20000 -0.018  9000 -0.017
```

Electron-Photon-Neutron Description of the Materials

```
m11     nlib=60c  plib=02p  elib=01e  pnlib=03u
        74182  0.237294
        74183  0.129157
        74184  0.276674
        74186  0.258020
        75185  0.036972
        75187  0.061883
mpn11   74184 5r
m12     nlib=60c  plib=02p  elib=01e  pnlib=03u
        29063  0.6917
        29065  0.3083
mpn12   29063 1r
m13     nlib=60c  plib=02p  elib=01e  pnlib=03u
        74182  0.226794
        74183  0.123441
        74184  0.264431
        74186  0.246603
        28058  0.067660
        28060  0.026063
        28061  0.001133
        28062  0.003612
        28064  0.000920
        29063  0.027213
        29065  0.012129
mpn13   74184 3r  29063 6r
m14     nlib=60c  plib=02p  elib=01e  pnlib=03u
        13027  1
m15     nlib=60c  plib=02p  elib=01e  pnlib=03u
        26054  0.039836
        26056  0.629963
        26057  0.015110
        26058  0.001923
        24050  0.008242
        24052  0.158937
        24053  0.018022
        24054  0.004486
        28058  0.057199
        28060  0.022033
        28061  0.000958
        28062  0.003053
        28064  0.000778
        25055  0.019948
        14000  0.019510
mpn15   26056 7r  29063 4r 26056 13027
m16     nlib=60c  plib=02p  elib=01e  pnlib=03u
        82206  0.245667
        82207  0.225666
        82208  0.528667
```

```
m17     nlib=60c  plib=02p  elib=01e  pnlib=03u
         7014      0.781532
         8016      0.210747
        18000.35c  0.004691
         7015      0.002871
         6000      0.000159
mpn17   0 4r
m18     nlib=60c  plib=02p  elib=01e  pnlib=03u
         1001  0.103128
         8016  0.585223
        14000  0.211247
        20000  0.038705
        11023  0.013912
        12000  0.001974
        13027  0.031709
        16000  0.000748
        19000  0.009203
        26054  0.000245
        26056  0.003812
        26057  0.000095
mpn18   0 0 13027 20040 0 0 13027 0 0 26056 2r
m19     nlib=60c  plib=02p  elib=01e  pnlib=03u
        79197  1
mpn19       0
m20     nlib=60c  plib=02p  elib=01e  pnlib=03u
         1001  0.585827
         6000  0.370166
         8016  0.021348
         7014  0.014879
        20000  0.002600
         9019  0.005180
mpn20   0 3r 20040 0
```

Electron-Photoatomic Options For Tally Detectors

```
mode  e p
phys:p 3j
cut:e   j  5.7   0   0
cut:p   j  5.7   0   0
print  -85 -120
prdmp 3j 3
```

Electron-Photoatomic Options For Volume Detectors

```
mode  e p
phys:p 2j 1
cut:e   j  5.7   0   0
cut:p   j  5.7   0   0
print  -85 -120
prdmp 3j 3
```

Electron-Photon-Neutron Options For Tally Detectors

```
mode  e p n
phys:p 2j 1 -1
cut:e   j  5.7   0   0
cut:p   j  5.7   0   0
cut:n   j    j   0   0
print  -85 -120
prdmp 3j 3
```

Electron-Photon-Neutron Options For Volume Detectors

```
mode  e p n
phys:p 3j -1
cut:e   j  5.7   0   0
```

```
cut:p   j  5.7   0   0
cut:n   j    j   0   0
print  -85 -120
prdmp 3j 3
```

Weight-Windows for Electron-Photon Simulation With Block and Ingot

```
wwp:e,p   4 3 10 0 0
wwe:e,p   1e20
wwn1:e,p  0.25 80r  -1 5r
```

Weight-Windows for Electron-Photon Simulation With Block But Without Ingot

```
wwp:e,p   4 3 10 0 0
wwe:e,p   1e20
wwn1:e,p  0.25 76r  -1 5r
```

Weight-Windows for Electron-Photon Simulation Without Block But With Ingot

```
wwp:e,p   4 3 10 0 0
wwe:e,p   1e20
wwn1:e,p  0.25 77r  -1 5r
```

Weight-Windows for Electron-Photon Simulation Without Block or Ingot

```
wwp:e,p   4 3 10 0 0
wwe:e,p   1e20
wwn1:e,p  0.25 75r  -1 5r
```

Weight-Windows for Electron-Photon-Neutron Simulation With Block and Ingot

```
wwp:e,p,n  4  3 10  0  0
wwe:e,p,n  1e20
wwn1:e,p   0.25  80r  -1 5r
wwn1:n     0.002 80r  -1 5r
```

Weight-Windows for Electron-Photon-Neutron Simulation With Block But Without Ingot

```
wwp:e,p,n  4  3 10  0  0
wwe:e,p,n  1e20
wwn1:e,p   0.25  76r  -1 5r
wwn1:n     0.002 76r  -1 5r
```

Weight-Windows for Electron-Photon-Neutron Simulation Without Block But With Ingot

```
wwp:e,p,n  4  3 10  0  0
wwe:e,p,n  1e20
wwn1:e,p   0.25  77r  -1 5r
wwn1:n     0.002 77r  -1 5r
```

Weight-Windows for Electron-Photon-Neutron Simulation Without Block or Ingot

```
wwp:e,p,n  4  3 10  0  0
wwe:e,p,n  1e20
wwn1:e,p   0.25  75r  -1 5r
wwn1:n     0.002 75r  -1 5r
```

Incident Electron Source – 19 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 19 MeV)
#       si2    sp2
         l      d
       17.9   0.0054
       18.0   0.0141
       18.1   0.0335
       18.2   0.0725
       18.3   0.1433
       18.4   0.2589
       18.5   0.4268
       18.6   0.6426
       18.7   0.8833
       18.8   1.1087
       18.9   1.2707
       19.0   1.3298
       19.1   1.2707
       19.2   1.1087
       19.3   0.8833
       19.4   0.6426
       19.5   0.4268
       19.6   0.2589
       19.7   0.1433
       19.8   0.0725
       19.9   0.0335
       20.0   0.0141
       20.1   0.0054
```

Incident Electron Source – 20 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 20 MeV)
#       si2    sp2
         l      d
       18.9   0.0054
       19.0   0.0141
       19.1   0.0335
       19.2   0.0725
       19.3   0.1433
       19.4   0.2589
       19.5   0.4268
       19.6   0.6426
       19.7   0.8833
       19.8   1.1087
       19.9   1.2707
       20.0   1.3298
       20.1   1.2707
       20.2   1.1087
       20.3   0.8833
       20.4   0.6426
       20.5   0.4268
       20.6   0.2589
       20.7   0.1433
       20.8   0.0725
       20.9   0.0335
       21.0   0.0141
       21.1   0.0054
```

Incident Electron Source – 21 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 21 MeV)
#       si2    sp2
         l      d
       19.9   0.0054
       20.0   0.0141
       20.1   0.0335
       20.2   0.0725
       20.3   0.1433
       20.4   0.2589
       20.5   0.4268
       20.6   0.6426
       20.7   0.8833
       20.8   1.1087
       20.9   1.2707
       21.0   1.3298
       21.1   1.2707
       21.2   1.1087
       21.3   0.8833
       21.4   0.6426
       21.5   0.4268
       21.6   0.2589
       21.7   0.1433
       21.8   0.0725
       21.9   0.0335
       22.0   0.0141
       22.1   0.0054
```

Incident Electron Source – 22 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 22 MeV)
#       si2    sp2
         l      d
       20.9   0.0054
       21.0   0.0141
       21.1   0.0335
       21.2   0.0725
       21.3   0.1433
       21.4   0.2589
       21.5   0.4268
       21.6   0.6426
       21.7   0.8833
       21.8   1.1087
       21.9   1.2707
       22.0   1.3298
       22.1   1.2707
       22.2   1.1087
       22.3   0.8833
       22.4   0.6426
       22.5   0.4268
       22.6   0.2589
       22.7   0.1433
       22.8   0.0725
       22.9   0.0335
       23.0   0.0141
       23.1   0.0054
```

Incident Electron Source – 23 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 23 MeV)
#       si2    sp2
         l      d
       21.9   0.0054
       22.0   0.0141
       22.1   0.0335
       22.2   0.0725
       22.3   0.1433
       22.4   0.2589
       22.5   0.4268
       22.6   0.6426
       22.7   0.8833
       22.8   1.1087
       22.9   1.2707
       23.0   1.3298
       23.1   1.2707
       23.2   1.1087
       23.3   0.8833
       23.4   0.6426
       23.5   0.4268
       23.6   0.2589
       23.7   0.1433
       23.8   0.0725
       23.9   0.0335
       24.0   0.0141
       24.1   0.0054
```

Incident Electron Source – 24 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 24 MeV)
#       si2    sp2
         l      d
       22.9   0.0054
       23.0   0.0141
       23.1   0.0335
       23.2   0.0725
       23.3   0.1433
       23.4   0.2589
       23.5   0.4268
       23.6   0.6426
       23.7   0.8833
       23.8   1.1087
       23.9   1.2707
       24.0   1.3298
       24.1   1.2707
       24.2   1.1087
       24.3   0.8833
       24.4   0.6426
       24.5   0.4268
       24.6   0.2589
       24.7   0.1433
       24.8   0.0725
       24.9   0.0335
       25.0   0.0141
       25.1   0.0054
```

Incident Electron Source – 25 MeV Mean Energy

```
sdef  par 3               $ electrons
      pos 0 0 0           $ starting at the origin
      sur 201     rad=d1  $ distributed uniformly on the surface within a spot
      vec 0 0 -1  dir=1   $ perpendicularly incident
      erg d2              $ Gaussian in energy
c
c  Distribute particles uniformly within spot size 0.05
si1     0.05
c
c  Gaussian in energy (About 25 MeV)
#       si2    sp2
         l      d
       23.9   0.0054
       24.0   0.0141
       24.1   0.0335
       24.2   0.0725
       24.3   0.1433
       24.4   0.2589
       24.5   0.4268
       24.6   0.6426
       24.7   0.8833
       24.8   1.1087
       24.9   1.2707
       25.0   1.3298
       25.1   1.2707
       25.2   1.1087
       25.3   0.8833
       25.4   0.6426
       25.5   0.4268
       25.6   0.2589
       25.7   0.1433
       25.8   0.0725
       25.9   0.0335
       26.0   0.0141
       26.1   0.0054
```

^{196}Au Production in Ingot 1

```
dxt:p 0 0 -100 2.5 r 0.1 0.0001
f204:p 501
e204   7.5 10 12.5 15 17.5 20 22.5 25 30
f214:p 501
fc214  Au-196 production in gold ingot at isocenter.
fm214 0.0595806783
de214 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df214 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
```

^{196}Au Production in Ingot 2

```
dxt:p 0 0 -100 30 r 0.1 0.0001
f204:p 501
e204   7.5 10 12.5 15 17.5 20 22.5 25 30
f224:p 501
fc224  Au-196 production in gold ingot at isocenter with A-150 moderator block.
fm224 0.0595806783
de224 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df224 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
```

^{196}Au Production in Ingot 3

```
dxt:p 564.081804406 33 -105.5 2.5 r 1e-08 1e-11
f204:p 501
e204   7.5 10 12.5 15 17.5 20 22.5 25 30
f234:p 670
fc234  Au-196 production in gold ingot in maze.
fm234 0.0595806783
de234 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df234 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
```

^{198}Au Production in Ingot 1

```
dxt:n 0 0 -100 2.5 r 1e-5 1e-8
f104:n 501
e104  0.01 0.05 0.1 0.25 0.5 0.75 1 2.5 5 10 15 30
f114:n 501
fc114  Au-198 production in gold ingot at isocenter.
fm114 0.0595806783 19 102
f124:n 501
fc124  Au-196 production in gold ingot at isocenter.
fm124 0.0595806783 19 16
```

^{198}Au Production in Ingot 2

```
dxt:n 0 0 -100 30 r 1e-4 1e-7
f134:n 501
e134  0.01 0.05 0.1 0.25 0.5 0.75 1 2.5 5 10 15 30
f144:n 501
fc144  Au-198 production in gold ingot at isocenter with A-150 moderator block.
fm144 0.0595806783 19 102
f154:n 501
fc154  Au-196 production in gold ingot at isocenter with A-150 moderator block.
fm154 0.0595806783 19 16
```

^{198}Au Production in Ingot 3

```
dxt:n 564.081804406 33 -105.5 2.5 r 1e-7 1e-10
f164:n 670
e164  0.01 0.05 0.1 0.25 0.5 0.75 1 2.5 5 10 15 30
f174:n 670
fc174  Au-198 production in gold ingot in maze.
fm174 0.0595806783 19 102
f184:n 670
fc184  Au-196 production in gold ingot in maze.
fm184 0.0595806783 19 16
```

^{196}Au Production by Point Detectors

```
f205:p  0 0 -100 0
e205    7.5 10 12.5 15 17.5 20 22.5 25 30
f215:p  0 0 -100 0
fc215 Au-196 production (pt. est.) at isocenter
fm215 0.0595806783
de215 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df215 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f225:p  0 -3 -100 0
fc225 Au-196 production (pt. est.) at radius 3 cm (cross-plane)
fm225 0.0595806783
de225 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df225 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f235:p  0 -6 -100 0
fc235 Au-196 production (pt. est.) at radius 6 cm (cross-plane)
fm235 0.0595806783
de235 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df235 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f245:p  0 -9 -100 0
fc245 Au-196 production (pt. est.) at radius 9 cm (cross-plane)
fm245 0.0595806783
de245 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df245 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f255:p  0 -11.5 -100 0
fc255 Au-196 production (pt. est.) at radius 11.5 cm (cross-plane)
fm255 0.0595806783
de255 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df255 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f265:p  0 -14.5 -100 0
fc265 Au-196 production (pt. est.) at radius 14.5 cm (cross-plane)
fm265 0.0595806783
de265 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df265 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
       0.0436  0.0399  0.037   0.035   0.0
f275:p  564.081804406 33 -105.5 0
fc275 Au-196 production (pt. est.) in maze
fm275 0.0595806783
de275 lin
         8.071   8.08    8.35    8.62    8.89
         9.16    9.44    9.71    9.98   10.25
        10.52   10.8    11.07   11.34   11.61
        11.88   12.16   12.43   12.7    12.97
        13.24   13.52   13.79   14.06   14.33
        14.6    14.88   15.42   15.69   15.96
        16.24   16.51   16.78   17.05   17.32
        17.6    17.87   18.14   18.41   18.68
        18.96   19.23    19.5   19.77   26.1
df275 lin
       0.0     0.0053  0.0223  0.0294  0.0399
       0.0496  0.0537  0.0736  0.0944  0.0941
       0.1117  0.1487  0.1741  0.2045  0.2636
       0.3126  0.3556  0.4135  0.4644  0.5064
       0.5249  0.5292  0.5268  0.5094  0.4961
       0.457   0.4207  0.3252  0.2762  0.2317
       0.1991  0.1662  0.1171  0.1031  0.0906
       0.0795  0.0697  0.0613  0.0542  0.0483
```

```
      0.0436  0.0399  0.037   0.035   0.0
```

^{198}Au Production by Point Detectors

```
f105:n  0 0 -100 0
e105  0.01 0.05 0.1 0.25 0.5 0.75 1 2.5 5 10 15 30
f115:n  0 0 -100 0
fc115 Au-198 production (pt. est.) at isocenter
fm115 0.0595806783 19 102
f125:n  0 -3 -100 0
fc125 Au-198 production (pt. est.) at radius 3 cm (cross-plane)
fm125 0.0595806783 19 102
f135:n  0 -6 -100 0
fc135 Au-198 production (pt. est.) at radius 6 cm (cross-plane)
fm135 0.0595806783 19 102
f145:n  0 -9 -100 0
fc145 Au-198 production (pt. est.) at radius 9 cm (cross-plane)
fm145 0.0595806783 19 102
f155:n  0 -11.5 -100 0
fc155 Au-198 production (pt. est.) at radius 11.5 cm (cross-plane)
fm155 0.0595806783 19 102
f165:n  0 -14.5 -100 0
fc165 Au-198 production (pt. est.) at radius 14.5 cm (cross-plane)
fm165 0.0595806783 19 102
f175:n  564.081804406 33 -105.5 0
fc175 Au-198 production (pt. est.) in maze
fm175 0.0595806783 19 102
```

Dose Calculations

Geometry For 5x5 Photon Field

```
c *******************
c  Cell Descriptions
c *******************
c
c --------------------------------------
c  Target, Primary Collimator & Filters
c --------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199   0           102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291   0           312  -311    214  -211
  298   0           319  -313    215  -212
  299   0           319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
```

```
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391   0           321  -111    233  -219
  392   0           111  -112    231  -219
  393   0           112  -113    232  -219
  394   0           113  -121    239  -219
c
c  Air surrounding target and filters
  399   0                 121    239  -201
                    410  -412    420  -422
c
c -------------------------
c  Collimator Jaw Assembly
c -------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402   0           ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412   0           ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422   0           ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432   0           ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  599   0            -401 599  410 -412  420 -422
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644   0            630 -632  -656 662  -601 603
  645   0            630 -631  -654 656  -601 603
  646   0            628 -631  -651 654  -601 603
  647   0            628 -629  -654 659  -601 603
  648   0            626 -629  -659 661  -601 603
  649   0            626 -627  -655 659  -601 603
  650   0            625 -626  -652 661  -601 603
  651   0            624 -625  -653 660  -602 603
  652   0            623 -624  -653 657  -602 603
  653   0            623 -624  -658 660  -602 603
  654   0            622 -623  -653 660  -602 603
  655   0            621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656   0            623 -624  -657 658  -602 201
  657   0            623 -624  -657 658  -599 605
  658   0            422 -624  -657 658  -201 599
  659   0            420 -422  -657 412  -201 599
  660   0            420 -422  -410 658  -201 599
  661   0            623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                -606
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c **********************
c  END Cell Descriptions
c **********************
c

c
c *********************
c  Surface Descriptions
c *********************
```

```
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c  Collimator opening set for a 05x05 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -2.4992258E-02   0.0000000E+00   9.9968764E-01  -4.0800000E+01
  461  p    -2.4992258E-02   0.0000000E+00   9.9968764E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9968764E-01   0.0000000E+00   2.4992258E-02   1.0000000E-03
  465  p     9.9968764E-01   0.0000000E+00   2.4992258E-02   1.5001000E+01
c
  470  p     2.4992258E-02   0.0000000E+00   9.9968764E-01  -4.0800000E+01
  471  p     2.4992258E-02   0.0000000E+00   9.9968764E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9968764E-01   0.0000000E+00  -2.4992258E-02  -1.0000000E-03
  475  p     9.9968764E-01   0.0000000E+00  -2.4992258E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -2.4992258E-02   9.9968764E-01  -2.7100000E+01
  481  p     0.0000000E+00  -2.4992258E-02   9.9968764E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9968764E-01   2.4992258E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9968764E-01   2.4992258E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   2.4992258E-02   9.9968764E-01  -2.7100000E+01
  491  p     0.0000000E+00   2.4992258E-02   9.9968764E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9968764E-01  -2.4992258E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9968764E-01  -2.4992258E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c ----------------------
c  Area around isocenter
c ----------------------
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Geometry For 10x10 Photon Field

```
c *******************
c  Cell Descriptions
c *******************
c
c --------------------------------------
c  Target, Primary Collimator & Filters
c --------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199   0           102  -121    209  -201
c
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291   0           312  -311    214  -211
  298   0           319  -313    215  -212
  299   0           319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391   0           321  -111    233  -219
  392   0           111  -112    231  -219
  393   0           112  -113    232  -219
  394   0           113  -121    239  -219
c
c  Air surrounding target and filters
  399   0                121    239  -201
                    410  -412    420  -422
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402   0           ( 480:-481 : -482: 483 : -484: 485 )
```

```
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412   0           ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422   0           ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432   0           ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c -----------------------
c  Area around isocenter
c -----------------------
c
  599   0            -401 599  410 -412  420 -422
c
c ------
c  Room
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
```

```
c
  644   0            630 -632  -656 662  -601 603
  645   0            630 -631  -654 656  -601 603
  646   0            628 -631  -651 654  -601 603
  647   0            628 -629  -654 659  -601 603
  648   0            626 -629  -659 661  -601 603
  649   0            626 -627  -655 659  -601 603
  650   0            625 -626  -652 661  -601 603
  651   0            624 -625  -653 660  -602 603
  652   0            623 -624  -653 657  -602 603
  653   0            623 -624  -658 660  -602 603
  654   0            622 -623  -653 660  -602 603
  655   0            621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656   0            623 -624  -657 658  -602 201
  657   0            623 -624  -657 658  -599 605
  658   0            422 -624  -657 658  -201 599
  659   0            420 -422  -657 412  -201 599
  660   0            420 -422  -410 658  -201 599
  661   0            623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
99995   0                -606
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c ***********************
c  END Cell Descriptions
c ***********************
c

c
c **********************
c  Surface Descriptions
c **********************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
```

```
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c  Collimator opening set for a 10x10 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  461  p    -4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.0000000E-03
  465  p     9.9875234E-01   0.0000000E+00   4.9937526E-02   1.5001000E+01
c
  470  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -4.0800000E+01
  471  p     4.9937526E-02   0.0000000E+00   9.9875234E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.0000000E-03
  475  p     9.9875234E-01   0.0000000E+00  -4.9937526E-02  -1.5001000E+01
c
  480  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -2.7100000E+01
  481  p     0.0000000E+00  -4.9937526E-02   9.9875234E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.0000000E-03
  483  p     0.0000000E+00   9.9875234E-01   4.9937526E-02   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -2.7100000E+01
  491  p     0.0000000E+00   4.9937526E-02   9.9875234E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.0000000E-03
  493  p     0.0000000E+00   9.9875234E-01  -4.9937526E-02  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c ----------------------
c  Area around isocenter
c ----------------------
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Geometry For 30x30 Photon Field

```
c *******************
c  Cell Descriptions
c *******************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
c  Tungsten/Rhenium electron target
  101  11  -19.47        -101    202  -201
c
c  Copper housing/cooling for electron target
  111  12   -8.96        -101    209  -202
  112  12   -8.96   101  -102    209  -201
c
c  Air around target assembly
  199   0           102  -121    209  -201
c
```

```
c  Primary (tungsten) collimator
  201  13  -18.78   311  -121    215  -209
c
c  Aluminum hardening filter
c    (within primary collimator)
  211  14   -2.7         -311    211  -209
c
c  Air above and below flattening filter
c    (within primary collimator)
  291   0           312  -311    214  -211
  298   0           319  -313    215  -212
  299   0           319  -121    219  -215
c
c  Flattening filter
c    (within primary collimator)
  221  15   -7.9         -312    212
  222  15   -7.9         -319         -212
  223  15   -7.9    313  -312    215  -212
  224  15   -7.9    312  -311    215  -214
c
c  Flattening filter
  301  15   -7.9         -321    233
  302  15   -7.9         -111    239  -233
  303  15   -7.9    111  -112    239  -231
  304  15   -7.9    112  -113    239  -232
c
c  Air surrounding flattening filter
  391   0           321  -111    233  -219
  392   0           111  -112    231  -219
  393   0           112  -113    232  -219
  394   0           113  -121    239  -219
c
c  Air surrounding target and filters
  399   0                 121    239  -201
                    410  -412    420  -422
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c
c  Positive Y collimator
  401  16  -11.35    -480  481    482 -483    484 -485
  402   0           ( 480:-481 : -482: 483 : -484: 485 )
                    (-239  400    411 -412    420 -422 )
c
c
c  Negative Y collimator
  411  16  -11.35    -490  491   -492  493    494 -495
  412   0           ( 490:-491 :  492:-493 : -494: 495 )
                    (-239  400    410 -411    420 -422 )
c
c
c  Positive X collimator
  421  16  -11.35    -460 461     462 -463    464 -465
  422   0           ( 460:-461 : -462: 463 : -464: 465 )
                    (-400  401    410 -412    421 -422 )
c
c
c  Negative X collimator
  431  16  -11.35    -470 471     472 -473   -474  475
  432   0           ( 470:-471 : -472: 473 :  474:-475 )
                    (-400  401    410 -412    420 -421 )
c
c ----------------------
c  Area around isocenter
c ----------------------
c
  599   0            -401 599  410 -412  420 -422
c
c ------
c  Room
```

```
c ------
c
c  Ceiling slab
c
  600  18   -2.35    620 -632  -650 662  -600 601
  601  18   -2.35    622 -625  -653 660  -601 602
c
c  Concrete walls
c
  602  18   -2.35    620 -621  -650 662  -601 603
  603  18   -2.35    621 -622  -650 651  -601 603
  604  18   -2.35    622 -625  -650 653  -601 603
  605  18   -2.35    625 -626  -650 652  -601 603
  606  18   -2.35    626 -628  -650 655  -601 603
  607  18   -2.35    627 -628  -655 659  -601 603
  608  18   -2.35    628 -631  -650 651  -601 603
  609  18   -2.35    631 -632  -650 656  -601 603
  610  18   -2.35    621 -622  -661 662  -601 603
  611  18   -2.35    622 -625  -660 662  -601 603
  612  18   -2.35    625 -629  -661 662  -601 603
  613  18   -2.35    629 -630  -654 662  -601 603
c
  614  18   -2.35    633 -623  -663 664  -604 605
  615  18   -2.35    623 -624  -663 657  -604 605
  616  18   -2.35    623 -624  -658 664  -604 605
  617  18   -2.35    624 -634  -663 664  -604 605
c
c  Floor slabs
c
  618  18   -2.35    620 -623  -650 662  -603 604
  619  18   -2.35    623 -624  -650 657  -603 604
  620  18   -2.35    623 -624  -658 662  -603 604
  621  18   -2.35    624 -632  -650 662  -603 604
  622  18   -2.35    633 -634  -663 664  -605 606
c
c  Ground under slab (void)
c
  640   0            620 -633  -650 662  -604 606
  641   0            633 -634  -650 663  -604 606
  642   0            633 -634  -664 662  -604 606
  643   0            634 -632  -650 662  -604 606
c
c  Air inside room
c
  644   0            630 -632  -656 662  -601 603
  645   0            630 -631  -654 656  -601 603
  646   0            628 -631  -651 654  -601 603
  647   0            628 -629  -654 659  -601 603
  648   0            626 -629  -659 661  -601 603
  649   0            626 -627  -655 659  -601 603
  650   0            625 -626  -652 661  -601 603
  651   0            624 -625  -653 660  -602 603
  652   0            623 -624  -653 657  -602 603
  653   0            623 -624  -658 660  -602 603
  654   0            622 -623  -653 660  -602 603
  655   0            621 -622  -651 661  -601 603
c
c  Air above pit and around accelerator/phantom
c
  656   0            623 -624  -657 658  -602 201
  657   0            623 -624  -657 658  -599 605
  658   0            422 -624  -657 658  -201 599
  659   0            420 -422  -657 412  -201 599
  660   0            420 -422  -410 658  -201 599
  661   0            623 -420  -657 658  -201 599
c
c ---------------
c  Outside world
c ---------------
c
99994   0            600
```

```
99995   0                -606
99996   0            606 -600  -620
99997   0            606 -600       632
99998   0            606 -600  -632 620  -662
99999   0            606 -600  -632 620       650
c
c ***********************
c  END Cell Descriptions
c ***********************
c

c
c **********************
c  Surface Descriptions
c **********************
c
c -------------------------------------
c  Target, Primary Collimator & Filters
c -------------------------------------
c
  101  cz    0.2725
  102  cz    1.0
c
  111  cz    3.85
  112  cz    4.0
  113  cz    4.65
c
  121  cz   10.0
c
  201  pz   -0.0
  202  pz   -0.1
  209  pz   -1.5
c
  211  pz   -7.62
  212  pz  -10.6
  214  pz  -11.57
  215  pz  -11.79
  219  pz  -12.4
c
  231  pz  -14.86
  232  pz  -15.11
  233  pz  -15.46
  239  pz  -15.66
c
  311  kz   -1.028  0.0631         -1
  312  kz   -7.84   0.416          -1
  313  kz   -9.94   1.653          -1
  319  kz  -12.32   0.1914         +1
c
  321  kz  -13.26   1.83376736112  -1
c
c ------------------------
c  Collimator Jaw Assembly
c ------------------------
c  Collimator opening set for a 30x30 field size
c
  400  pz  -38.0
  401  pz  -51.0
c
  410  py  -50.0
  411  py    0.0
  412  py   50.0
c
  420  px  -50.0
  421  px    0.0
  422  px   50.0
c
  460  p    -1.4834105E-01   0.0000000E+00   9.8893626E-01  -4.0800000E+01
  461  p    -1.4834105E-01   0.0000000E+00   9.8893626E-01  -5.0800000E+01
  462  py   -1.1000000E+01
  463  py    1.1000000E+01
  464  p     9.8893626E-01   0.0000000E+00   1.4834105E-01   1.0000000E-03
  465  p     9.8893626E-01   0.0000000E+00   1.4834105E-01   1.5001000E+01
c
  470  p     1.4834105E-01   0.0000000E+00   9.8893626E-01  -4.0800000E+01
  471  p     1.4834105E-01   0.0000000E+00   9.8893626E-01  -5.0800000E+01
  472  py   -1.1000000E+01
  473  py    1.1000000E+01
  474  p     9.8893626E-01   0.0000000E+00  -1.4834105E-01  -1.0000000E-03
  475  p     9.8893626E-01   0.0000000E+00  -1.4834105E-01  -1.5001000E+01
c
  480  p     0.0000000E+00  -1.4834105E-01   9.8893626E-01  -2.7100000E+01
  481  p     0.0000000E+00  -1.4834105E-01   9.8893626E-01  -3.7100000E+01
  482  p     0.0000000E+00   9.8893626E-01   1.4834105E-01   1.0000000E-03
  483  p     0.0000000E+00   9.8893626E-01   1.4834105E-01   1.5001000E+01
  484  px   -1.1000000E+01
  485  px    1.1000000E+01
c
  490  p     0.0000000E+00   1.4834105E-01   9.8893626E-01  -2.7100000E+01
  491  p     0.0000000E+00   1.4834105E-01   9.8893626E-01  -3.7100000E+01
  492  p     0.0000000E+00   9.8893626E-01  -1.4834105E-01  -1.0000000E-03
  493  p     0.0000000E+00   9.8893626E-01  -1.4834105E-01  -1.5001000E+01
  494  px   -1.1000000E+01
  495  px    1.1000000E+01
c
c ----------------------
c  Area around isocenter
c ----------------------
c
  599  pz  -130.0
c
c ------
c  Room
c ------
c
c Z planes for floor and ceiling locations
c
  600  pz   333.8
  601  pz   237.28
  602  pz   135.68
c 201  pz     0.0
c 599  pz  -130.0
  603  pz  -225.0
  604  pz  -255.48
  605  pz  -476.46
  606  pz  -506.94
c
c X planes for walls
c
  620  px  -495.3
  621  px  -403.86
  622  px  -190.5
  623  px   -95.25
c 422  px   -50.0
c 421  px     0.0
c 420  px    50.0
  624  px    95.25
  625  px   190.5
  626  px   472.44
  627  px   518.16
  628  px   563.88
  629  px   723.9
  630  px   830.58
  631  px   990.6
  632  px  1036.32
  633  px  -125.73
  634  px   125.73
c
c Y planes for walls
c
  650  py   577.85
  651  py   486.41
  652  py   448.31
  653  py   372.11
  654  py   276.86
  655  py   158.75
  656  py   128.27
  657  py    95.25
c 412  py    50.0
c 411  py     0.0
c 410  py   -50.0
  658  py   -95.25
  659  py  -207.01
  660  py  -367.03
  661  py  -419.1
  662  py  -525.78
  663  py   125.73
  664  py  -125.73
c
c **************************
c  END Surface Descriptions
c **************************
c
```

Materials for Electron-Photon Simulation

```
m11     plib=02p  elib=03e
        74000 -0.90  75000 -0.10
m12     plib=02p  elib=03e
        29000 1
m13     plib=02p  elib=03e
        74000 -0.95  28000 -0.035  29000 -0.015
m14     plib=02p  elib=03e
        13027 1
m15     plib=02p  elib=03e
        26000 -0.70  24000 -0.18  28000 -0.09
        25000 -0.02  14000 -0.01
m16     plib=02p  elib=03e
        82000 1
m18     plib=02p  elib=03e
         1000 -0.0055   8000 -0.4984  14000 -0.3157
        20000 -0.0826  11000 -0.017   12000 -0.0026
        13000 -0.0455  16000 -0.0013  19000 -0.0191
        26000 -0.0123
```

Materials for Electron-Photon-Neutron Simulation

```
m11     nlib=60c  plib=02p  elib=01e  pnlib=03u
        74182  0.237294
        74183  0.129157
        74184  0.276674
        74186  0.258020
        75185  0.036972
        75187  0.061883
mpn11   74184 5r
m12     nlib=60c  plib=02p  elib=01e  pnlib=03u
        29063  0.6917
        29065  0.3083
mpn12   29063 1r
m13     nlib=60c  plib=02p  elib=01e  pnlib=03u
        74182  0.226794
        74183  0.123441
        74184  0.264431
        74186  0.246603
        28058  0.067660
        28060  0.026063
        28061  0.001133
        28062  0.003612
        28064  0.000920
        29063  0.027213
        29065  0.012129
```

```
mpn13   74184 3r  29063 6r
m14     nlib=60c  plib=02p  elib=01e  pnlib=03u
        13027  1
m15     nlib=60c  plib=02p  elib=01e  pnlib=03u
        26054  0.039836
        26056  0.629963
        26057  0.015110
        26058  0.001923
        24050  0.008242
        24052  0.158937
        24053  0.018022
        24054  0.004486
        28058  0.057199
        28060  0.022033
        28061  0.000958
        28062  0.003053
        28064  0.000778
        25055  0.019948
        14000  0.019510
mpn15   26056 7r  29063 4r 26056 13027
m16     nlib=60c  plib=02p  elib=01e  pnlib=03u
        82206  0.245667
        82207  0.225666
        82208  0.528667
m18     nlib=60c  plib=02p  elib=01e  pnlib=03u
         1001  0.103128
         8016  0.585223
        14000  0.211247
        20000  0.038705
        11023  0.013912
        12000  0.001974
        13027  0.031709
        16000  0.000748
        19000  0.009203
        26054  0.000245
        26056  0.003812
        26057  0.000095
mpn18   0 0 13027 20040 0 0 13027 0 0 26056 2r
```

Options for Electron-Photon Simulation

```
mode  e p
phys:p 2j 1
cut:e   j  0.5   0   0
cut:p   j  0.1   0   0
print  -85
prdmp 3j 3
```

Options for Electron-Photon-Neutron Simulation

```
mode  e p n
phys:p 2j 1 -1
cut:e   j  5.7   0   0
cut:p   j  5.7   0   0
cut:n   j    j   0   0
print  -85
prdmp 3j 3
```

Weight-Windows for Electron-Photon Simulation

```
wwp:e,p   5 3 10 0 0
wwe:e,p   1e20
wwn1:e,p  0.2 75r  -1 5r
```

Weight-Windows for Electron-Photon-Neutron Simulation

```
wwp:e,p,n   5  3 10  0  0
wwe:e,p,n  1e20
wwn1:e,p   0.2   75r  -1 5r
wwn1:n     0.002 75r  -1 5r
```

Energy Specification for 10 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 10
si1   0.05
```

Energy Specification for 15 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 15
si1   0.05
```

Energy Specification for 20 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 20
si1   0.05
```

Energy Specification for 25 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 25
si1   0.05
```

Energy Specification for 30 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 30
si1   0.05
```

Energy Specification for 40 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 40
si1   0.05
```

Energy Specification for 50 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 50
si1   0.05
```

Energy Specification for 75 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 75
si1   0.05
```

Energy Specification for 100 MeV

```
sdef  par 3 pos 0 0 0 sur 201 rad d1 vec 0 0 -1 dir=1 erg 100
si1   0.05
```

REFERENCES

1. Heaton, H.T. II, and Jacobs, R., eds. *Proceedings of a Conference on Neutrons from Electron Medical Accelerators*. NBS Special Publication 554. National Bureau of Standards, Department of Commerce: Gaithersburg, MD, 1979.

2. McCall, R.C., Almond, P. R., Devanney, J. A., Fuller, E. G., Holeman, G. R., Lanzl, L. H., Ing, H., and Swanson, W. P. *Neutron Contamination from Medical Electron Accelerators*. NCRP Report No. 79. National Council on Radiation Protection and Measurements: Bethesda, MD, 1984.

3. Breismeister, J.F., ed. *MCNP - A General Monte Carlo N-Particle Transport Code.* LA-12625-M. Los Alamos National Laboratory: Los Alamos, NM, 1997.

4. Metropolis, N. and Ulam, S. "The Monte Carlo Method," *Journal of the American Statistical Association*. Vol. 44, No. 247, pp. 335-341, 1949.

5. Bohr, A. and Mottelson, B.R. *Nuclear Structure*. 2nd Edition. World Scientific: Singapore, 1998.

6. Levinger, J.S. "Neutron Production by Complete Absorption of High-Energy Photons," *Nucleonics*. Vol. 6, No. 5, pp. 64-67, 1950.

7. Levinger, J.S. "The High Energy Nuclear Photoeffect," *Physical Review*. Vol. 84, No. 1, pp. 43-51, 1951.

8. Chadwick, M.B., Oblozinsky, P., Hodgson, P.E., and Reffo, G. "Pauli-Blocking in the Quasideuteron Model of Photoabsorption," *Physical Review C*. Vol. 44, No. 2, pp. 814-823, 1991.

9. Wu, J.R. and Chang, C.C. "Pre-Equilibrium Particle Decay in the Photonuclear Reactions," *Physical Review C*. Vol. 16, No. 5, pp. 1812-1824, 1977.

10. Blann, M., Berman, B.L., and Komoto, T.T. "Precompound-Model Analysis of Photoneutron Reaction," *Physical Review C*. Vol. 28, No. 6, pp. 2286-2298, 1983.

11. Chadwick, M.B., Young, P.G, and Chibas, S. "Photonuclear Angular-Distribution Systematics in the Quasideuteron Regime," *Journal of Nuclear Science and Technology*. Vol. 32, No. 11, pp. 1154-1158, 1995.

12. Fasso, A., Ferrari, A., and Sala, P.R. "Total Giant Resonance Photonuclear Cross Sections for Light Nuclei: A Database for the FLUKA Monte Carlo Transport Code," *Proceedings of the Third Specialists Meeting on Shielding Aspects of Accelerators, Targets, and Irradiation Facilities*. SATIF-3, Tohoku University, Sendai, Japan. Organization for Economic Cooperation and Development Nuclear Energy Agency: Paris, France, 1997.

13. Mughabghab, S.F., Divadeenam, M., Holden, N.E., McLane, V., Dunford, C.L., and Rose, P.F., eds. *Neutron Cross Sections*. 4th Edition. BNL-325. Brookhaven National Laboratory: Upton, NY, 1981.

14. Fuller, E.G., Gerstenberg, H.M., Vander Molen, H., and Dunn, T.C., eds. *Photonuclear Reaction Data*. NBS Special Publication 380. National Bureau of Standards, Department of Commerce: Gaithersburg, MD, 1973.

15. Fuller, E.G. and Gerstenberg, H.M., eds. *Photonuclear Data Index 1973-1977*. NBS Special Publication 380, Supplement 1. National Bureau of Standards, Department of Commerce: Gaithersburg, MD, 1978.

16. Berman, B.L. "Atlas of Photoneutron Cross Sections Obtained With Monoenergetic Photons," *Atomic Data and Nuclear Data Tables*. Vol. 15, No. 4, pp. 319-390, 1975.

17. Dietrich, S.S. and Berman, B.L. "Atlas of Photoneutron Cross Sections Obtained with Monoenergetic Photons," *Atomic Data and Nuclear Data Tables*. Vol. 38, No. 2, pp. 199-338, 1988.

18. Varlamov, A.V., Varlamov, V.V., Rudenko, D.S., and Stepanov, M.E. *Atlas of Giant Dipole Resonance Parameters and Graphs of Photonuclear Reaction Cross Sections*. INDC(NDS)-394. International Atomic Energy Association: Vienna, Austria, 1999.

19. Dunford, C.L. *Internet Connection to the National Nuclear Data Center (NNDC)*. http://www.nndc.bnl.gov/. Brookhaven National Laboratory: Upton, NY, 1998.

20. Alsmiller, R.G. Jr. and Moran, H.S. "Electron-Photon Cascade Calculations and Neutron Yields from Electron in Thick Targets," *Nuclear Instruments and Methods*. Vol. 48, pp. 109-116, 1967.

21. Alsmiller, R.G. Jr. and Moran, H.S. "Photoneutron Production from 34- and 100-MeV Electrons in Thick Uranium Targets," *Nuclear Instruments and Methods*. Vol. 51, pp. 339-340, 1967.

22. Alsmiller, R.G. Jr., Gabriel, T.A., and Guthrie, M.P. "The Energy Distribution of Photoneutrons Produced by 150 MeV Electrons in Thick Beryllium and Tantalum Targets," *Nuclear Science and Engineering*. Vol. 40, No. 3, pp. 365-374, 1970.

23. Gabriel, T.A. and Alsmiller, R.G. Jr. "Photonucleon and Photopion Production from 400 MeV Electrons in Thick Copper Targets," *Nuclear Physics B*. Vol. B14, No. 2, pp. 303-315, 1969.

24. Gabriel, T.A. and Alsmiller, R.G. Jr. "Photonuclear Disintegration at High Energies (<350 MeV)," *Physical Review*. Vol. 182, No. 4, pp. 1035-1050, 1969.

25. Hansen, E.C., Bartoletti, C.S., and Daitch, P.B. "Analog Monte Carlo Studies of Electron-Photon Cascades and the Resultant Production and Transport of Photoneutrons in Finite three-dimensional Systems," *Journal of Applied Physics*. Vol. 46, No. 3, pp. 1109-1123, 1975.

26. Kase, T. and Harada, H. "An Assessment of the Continuous Neutron Source Using a Low-Energy Electron Accelerator," *Nuclear Science and Engineering*. Vol. 126, No. 1, pp. 59-70, 1997.

27. Mokhov, N.V., Striganov, S.I., Van Ginneken, A., Mashnik, S.G., Sierk, A.J., and Ranft, J. "MARS Code Development," *Proceedings of the Fourth Specialists Meeting on Shielding Aspects of Accelerators, Targets and Irradiation Facilities*. SATIF-4, Knoxville, TN. Organization for Economic Cooperation and Development Nuclear Energy Agency: Paris, France, 1998.

28. Prokofiev, A.V., Mashnik, S.G., and Sierk, A.J. "Cascade-Exciton Model Analysis of Nucleon-Induced Fission Cross Sections of Lead and Bismuth at 45- to 500-MeV Energies," *Nuclear Science and Engineering*. Vol. 131, No. 1, pp. 78-95, 1999.

29. Swanson, W.P. "Activation of Aluminum Beam Dumps By High-Energy Electrons At SLAC," *Health Physics*. Vol. 28, No. 5, pp. 495-502, 1975.

30. Swanson, W.P. "Neutron Yields From Electrons Stopped in Selected Materials," *Health Physics*. Vol. 33, No. 6, pp. 686-687, 1977.

31. Swanson, W.P. "Calculation of Neutron Yields Released By Electrons Incident On Selected Materials," *Health Physics*. Vol. 35, No. 2, pp. 353-367, 1978.

32. Swanson, W.P. "Improved Calculation of Photo-Neutron Yields Released By Incident Electrons," *Health Physics*. Vol. 37, No. 3, pp. 347-358, 1979.

33. Swanson, W.P. "Estimate of the Risk in Radiation-Therapy Due to Unwanted Neutrons," *Medical Physics*. Vol. 7, No. 2, pp. 141-144, 1980.

34. Manfredotti, C., Nastasi, U., Ornato, E., and Zanini, A. "Evaluation of the Undesired Neutron Dose Equivalent to Critical Organs in Patients Treated By Linear-Accelerator Gamma-Ray Therapy," *Radiation Protection Dosimetry*. Vol. 44, No. 1-4, pp. 457-462, 1992.

35. Agosteo, S., Para, A.F., Silari, M., Torresin, A., and Tosi, G. "Monte-Carlo Simulations of Neutron-Transport in a Linac Radiotherapy Room," *Nuclear Instruments & Methods in Physics Research Section B*. Vol. 72, No. 1, pp. 84-90, 1992.

36. Agosteo, S., Para, A.F., Gerardi, F., Silari, M., Torresin, A., and Tosi, G. "Photoneutron Dose in Soft-Tissue Phantoms Irradiated By 25 MeV X-Rays," *Physics in Medicine and Biology*. Vol. 38, No. 10, pp. 1509-1528, 1993.

37. Gallmeier, F.X. "A Photoneutron Production Option for MCNP4A," *Proceedings of the Radiation Protection and Shielding Topical Conference on Advancements and Applications in Radiation Protection and Shielding*. No. Falmouth, MA. American Nuclear Society: La Grange Park, IL, 1996.

38. Liu, J.C., Nelson, W.R., Kase, K.R., and Mao, X.S. "Calculations of the Giant-Dipole-Resonance Photoneutrons Using a Coupled EGS4-MORSE Code," *Radiation Protection Dosimetry*. Vol. 70, No. 1-4, pp. 49-54, 1997.

39. Chadwick, M.B., Brown, T.H., and Little, R.C. "Photoneutron Production in Electron Beam Stop for Dual Axis Radiographic Hydrotest Facility (DARHT)," *Proceedings of the Radiation Protection and Shielding Topical Conference*. Nashville, TN. American Nuclear Society: La Grange Park, IL, 1998.

40. Chadwick, M.B., Young, P.G., Chiba, S., Frankle, S.C., Hale, G.M., Hughes, H.G., Koning, A.J., Little, R.C., MacFarlane, R.E., Prael, R.E., and Waters, L.S. "Cross-Section Evaluations to 150 MeV for Accelerator-Driven Systems and Implementation in MCNPX," *Nuclear Science and Engineering*. Vol. 131, No. 3, pp. 293-328, 1999.

41. Young, P.G. and Chadwick, M.B. "Comprehensive Nuclear Model Calculations: Theory and Use of the GNASH Code," *IAEA Workshop on Nuclear Reaction Data and Nuclear Reactors - Physics, Design and Safety*. Triest, Italy. World Scientific Publishing, Ltd: Singapore, 1996.

42. Hughes, H.G., Prael, R.E., and Little, R.C. *MCNPX - The LAHET/MCNP Code Merger*. XTM-RN-97-012. Los Alamos National Laboratory: Los Alamos, NM, 1997.

43. Oblozinsky, P., ed. *Summary Report of IAEA 1st Research Coordination Meeting*. INDC(NDS)-364. International Atomic Energy Association: Vienna, Austria, 1997.

44. Oblozinsky, P., ed. *Handbook of Photonuclear Data for Applications*. IAEA-TECDOC-In Press. International Atomic Energy Association: Vienna, Austria, 2000.

45. McLane, V., Dunford, C.L., and Rose, P.F., eds. *ENDF-102 Data Formats and Procedures for the Evaluated Nuclear Data File ENDF-6*. BNL-NCS-44945. Brookhaven National Laboratory: Upton, NY, 1997.

46. Werner, C.J. *Proposed Delayed Neutron Data Format for MCNP Libraries*. XCI:CJW-98-121. Los Alamos National Laboratory: Los Alamos, NM, 1998.

47. Frankle, S.C. *Proposed APT Data Library Formats and ZAID Specifications*. XTM:SCF-96-200. Los Alamos National Laboratory: Los Alamos, NM, 1996.

48. Frankle, S.C. *Follow-up to XTM:SCF-96-200, Proposed APT Data Library Formats*. XTM:SCF-96-312. Los Alamos National Laboratory: Los Alamos, NM, 1996.

49. MacFarlane, R.E. and Muir, D.W. *The NJOY Nuclear Data Processing System, Version 91*. LA-12740-M. Los Alamos National Laboratory: Los Alamos, NM, 1994.

50. Kalbach, C. *Systematics of Continuum Angular Distributions: Extensions to Higher Energies*. LA-UR-87-4139. Los Alamos National Laboratory: Los Alamos, NM, 1987.

51. Kalbach, C. "Systematics of Continuum Angular Distributions: Extensions to Higher Energies," *Physical Review C*. Vol. 37, No. 6, pp. 2350-2370, 1988.

52. White, M.C. *Modifications to the MCNP ACE Routines*. XCI:MCW-99-80. Los Alamos National Laboratory: Los Alamos, NM, 1999.

53. White, M.C. *Verification of the ACE Modifications to MCNP*. XCI:MCW-99-92. Los Alamos National Laboratory: Los Alamos, NM, 1999.

54. Hendricks, J.S. *MCNP4C ENDF65 Capability*. X-5:JSH-99-08. Los Alamos National Laboratory: Los Alamos, NM, 1999.

55. White, M.C. *MCNPX Modifications - Proton Table Loading and Mix & Match Issues*. XCI:MCW-99-2. Los Alamos National Laboratory: Los Alamos, NM, 1999.

56. Carter, L.L. and Cashwell, E.D. *Particle-Transport Simulation with the Monte Carlo Method*. Report No. TID-26607. National Technical Information Service: Springfield, VA, 1975.

57. Hendricks, J.S. *MCNP4C Delayed Neutrons*. X-5:JSH-99-10. Los Alamos National Laboratory: Los Alamos, NM, 1999.

58. Barber, W.C. and George, W.D. "Neutron Yields from Targets Bombarded by Electrons," *Physical Review*. Vol. 116, No. 6, pp. 1551-1559, 1959.

59. White, M.C. *Response to X-5:JSH-99-08 - MCNP4C ENDF65 Capability*. X-5:MCW-99-17. Los Alamos National Laboratory: Los Alamos, NM, 1999.

60. White, M.C. *Photonuclear Physics in MCNP(X) Progress Report*. XCI:MCW-99-17. Los Alamos National Laboratory: Los Alamos, NM, 1999.

61. Yung-Su, T. "Pair Production and Bremsstrahlung of Charged Leptons," *Reviews of Modern Physics*. Vol. 46, No. 4, pp. 815-851, 1974.

62. Veyssiere, A., Beil, H., Bergere, R., Carlos, P., Lepretre, A., and De Miniac, A. "A Study of the Photoneutron Contribution to the Giant Dipole Resonance of s-d Shell Nuclei," *Nuclear Physics A*. Vol. A227, No. 3, pp. 513-540, 1974.

63. Montalbetti, R., Katz, L., and Goldemberg, J. "Photoneutron Cross Sections," *Physical Review*. Vol. 91, No. 3, pp. 659-673, 1953.

64. Price, G.A. and Kerst, D.W. "Yields and Angular Distributions of Some Gamma-Neutron Processes," *Physical Review*. Vol. 77, No. 6, pp. 806-809, 1950.

65. Fultz, S.C., Bramblett, R.L., Caldwell, J.T., and Harvey, R.R. "Photoneutron Cross Sections for Natural Cu, Cu^{63} and Cu^{65}," *Physical Review*. Vol. B133, No. 5, pp. 1149-1154, 1964.

66. Bergere, R., Beil, H., and Veyssiere, A. "Photoneutron Cross Sections of La, Tb, Ho and Ta," *Nuclear Physics A*. Vol. A121, No. 2, pp. 463-480, 1968.

67. Veyssiere, A., Beil, H., Bergere, R., Carlos, P., Lepretre, A., and de Miniac, A. "Etude de la Resonance Geante Dipolaire dans la Region de Transitio autour de A = 190," *Le Journal de Physique*. Vol. 36, pp. L267-L270, 1975.

68. Harvey, R.R., Caldwell, J.T., Bramblett, R.L., and Fultz, S.C. "Photoneutron Cross Sections of Pb^{206}, Pb^{207}, Pb^{208} and Bi^{209}," *Physical Review*. Vol. B136, No. 1, pp. 126-131, 1964.

69. Veyssiere, A., Beil, H., Bergere, R., Carlos, P., and Lepretre, A. "Photoneutron Cross Sections of ^{208}Pb and ^{197}Au," *Nuclear Physics A*. Vol. A159, No. 2, pp. 561-576, 1970.

70. Hendricks, J.S., Frankle, S.C., and Court, J.D. *ENDF/B-VI Data for MCNP*. LA-12891. Los Alamos National Laboratory: Los Alamos, NM, 1994.

71. Ahrens, J., Borchert, H., Czock, K.H., Eppler, H.B., Gimm, H., Gundrum, H., Kroning, M., Riehn, P., Sita, G., Zieger, A., and Ziegler, B. "Total Nuclear Photon Absorption Cross Sections for Some Light Elements," *Nuclear Physics A*. Vol. A251, No. 3, pp. 479-492, 1975.

72. Fultz, S.C., Caldwell, J.T., Berman, B.L., Bramblett, R.L., and Harvey, R.R. "Photoneutron Cross Sections for C^{12} and Al^{27}," *Physical Review*. Vol. 143, No. 3, pp. 790-796, 1966.

73. Varlamov, V.V., Sapunenko, V.V., and Stepanov, M.E. *Internet Connection to the Photonuclear Data Index 1976-1995*. http://depni.npi.msu.su/cdfe/. Izdatel'stvo Muskovskogo Universiteta, Moscow State University: Moscow, Russia, 1996.

74. Kishida, N. *Nuclear Data Evaluation Methodology*. World Scientific: Singapore, 1993.

75. Costa, S., Ferrero, F., Manfredotti, C., Pasqualini, L., and Piragino, G. "Behavior of the (γ,Tn) Cross-Section in Selenium and Iron," *Nuovo Cimento*. Vol. 51B, Series 10, No. 1, pp. 199-201, 1967.

76. Dolbilkin, B.S., Isakov, A.I., Korin, V.I., Lazareva, L.E., Lin'kova, N.V., and Tulupov, B.A. "Absorption of Gamma Quanta by Iron Nuclei Near the Giant Resonance," *Yadernaya Fizika*. Vol. 9, No. 4, pp. 675-679, 1969.

77. Sund, R.E., Baker, M.P., Kull, L.A., and Walton, R.B. "Measurements of the ^{63}Cu (γ,n) and (γ,2n) Cross Sections," *Physical Review*. Vol. 176, No. 4, pp. 1366-1376, 1968.

78. Miller, J., Schuhl, C., and Tzara, C. "Mesure des Sections Efficaces (γ,n) de Cu, Ce, La, Ta, Au, Pb et Bi en Valeur Absolue," *Nuclear Physics*. Vol. 32, pp. 236-245, 1962.

79. Bramblett, R.L., Caldwell, J.T., Auchampaugh, G.F., and Fultz, S.C. "Photoneutron Cross Sections of Ta^{181} and Ho^{165}," *Physical Review*. Vol. 129, No. 6, pp. 2723-2729, 1963.

80. Lee, Y.O., Fukahori, T., and Chatt, J. "Evaluation of Photonuclear Reaction Data on Tantalum-181 up to 140 MeV," *Journal of Nuclear Science and Technology*. Vol. 35, No. 10, pp. 685-691, 1998.

81. Berman, B.L., Pywell, R.E., Thompson, M.N., McNeill, K.G., Jury, J.W., and Woodworth, J.G. *Bulletin of the American Physical Society*. Vol. 31, p. 855, 1986.

82. Barber, W.C. "Specific Ionization by High-Energy Electrons," *Physical Review*. Vol. 97, No. 4, pp. 1071-1077, 1955.

83. Udalesmith, M. "Monte-Carlo Calculations of Electron-Beam Parameters For 3 Philips Linear Accelerators," *Physics in Medicine and Biology*. Vol. 37, No. 1, pp. 85-105, 1992.

84. Rogers, D.W.O., Faddegon, B.A., Ding, G.X., Ma, C.M., We, J., and Mackie, T.R. "BEAM: A Monte Carlo Code to Simulate Radiotherapy Treatment Units," *Medical Physics*. Vol. 22, No. 5, pp. 503-524, 1995.

85. Hughes, G.H. *Information on the MCPLIB02 Photon Library*. X-6:HGH-93-77. Los Alamos National Laboratory: Los Alamos, NM, 1993.

86. Chetty, I. *A Photon Phase Space Source Model Incorporating Efficient Sampling Algorithms for Clinical Treatment Planning Using the Monte Carlo Method*. Dissertation. Department of Radiation Oncology, University of California at Los Angeles: Los Angeles, CA, 1999.

87. Alfassi, Z.B., ed. *Activation Analysis*. CRC Press: Boca Raton, FL, 1990.

88. Nath, R., Boyer, A.L., La Riviere, P.D., McCall, R.C., and Price, K.W. *Neutron Measurements Around High Energy X-ray Radiotherapy Machines*. AAPM Report No. 19. American Association of Physicists in Medicine: New York, NY, 1986.

89. Fultz, S.C., Bramblett, R.L., Caldwell, J.T., and Kerr, N.A. "Photoneutron Cross-Section Measurements on Gold Using Nearly Monochromatic Photons," *Physical Review*. Vol. 127, No. 4, pp. 1273-1279, 1962.

90. Berman, B.L., Pywell, R.E., Dietrich, S.S., Thompson, M.N., McNeill, K.G., and Jury, J.W. "Absolute photoneutron cross sections for Zr, I, Pr, Au, and Pb," *Physical Review C*. Vol. 36, No. 4, pp. 1286-1292, 1987.

91. DeMarco, J.J. *Modeling the Phillips SL Series MEA*. E-mail communication. 1998.

92. Chetty, I., DeMarco, J.J., and Solberg, T.D. "A Virtual Source Model for Monte Carlo Modeling of Arbitrary Intensity Distributions," *Medical Physics*. Vol. 27, No. 1, pp. 166-172, 2000.

93. Hansen Lind Meyer Inc. *Radiation Therapy Center for the University of Florida Health Science Center*. Project Number #87061.05. Hansen Lind Meyer Inc.: Orlando, FL, 1989.

94. White, R.C. *Typical Construction Materials Used in Finishing a Clinical Setting*. E-mail communication. 1999.

95. Bates, T. *Deflection System for Charged Particle Beams*. Patent No. 4,409,486. United States Patent and Trademark Office: Washington, D.C., 1983.

96. Halbleib, J.A., Kensek, R.P., Valdez, G.D., Seltzer, S.M., and Berger, M.J. "ITS: The Integrated TIGER Series of Electron/Photon Transport Codes - Version 3.0," *IEEE Transactions on Nuclear Science*. Vol. 39, No. 4, pp. 1025-1030, 1992.

97. Love, P.A., Lewis, D.G., AlAffan, I.A.M., and Smith, C.W. "Comparison of EGS4 and MCNP Monte Carlo Codes When Calculating Radiotherapy Depth Doses," *Physics in Medicine and Biology*. Vol. 43, No. 5, pp. 1351-1357, 1998.

98. Jeraj, R., Keall, P.J., and Ostwald, P.M. "Comparisons between MCNP ; EGS4 and experiment for clinical electron beams," *Physics in Medicine and Biology*. Vol. 44, No. 3, pp. 705-717, 1999.

99. Adams, K.J. *Integration into MCNP4C of the ITS3.0 Radiative and Collisional Stopping Power and Bremsstrahlung Production Model*. X-5:KJA-00-34. Los Alamos National Laboratory: Los Alamos, NM, 2000.

100. Adams, K.J. *Availability of MCNP4BNU*. E-mail communication. 1999.

101. *Internet Connection to the Blue Mountain Supercomputing Platform Homepage*. http://www.lanl.gov/projects/asci/bluemtn/bluemtn.htm. Los Alamos National Laboratory: Los Alamos, NM, 2000.

102. Frankle, S.C. *ENDF60 Information*. XTM:SCF-95-278. Los Alamos National Laboratory: Los Alamos, NM, 1995.

103. MacFarlane, R.E. *Internet Connection to the T-2 Nuclear Information Service*. http://t2.lanl.gov/. Los Alamos National Laboratory: Los Alamos, NM, 2000.

104. EG&G Ortec. *Nuclide Navigator*. Version 1.01. EG&G Ortec: Oak Ridge, TN, 1996.

105. EG&G Ortec. *GammaVision*. Version 4.10. EG&G Ortec: Oak Ridge, TN, 1997.

106. Rogers, D.W.O. "Fluence to Dose Equivalent Conversion Factors Calculated With EGS3 For Electrons From 100-Kev to 20-Gev and Photons From 11-Kev to 20-Gev," *Health Physics*. Vol. 46, No. 4, pp. 891-914, 1984.

107. Thomas, R.H., Brackenbush, L.W., Chartier, J-L., Clark, M.J., Dietze, G., Drexler, G., Menzel, H.G., Griffith, R., Grosswendt, B., Peroussi-Henss, N., Siebert, B.R.L., and Zankl, M. *Conversion Coefficients for use in Radiological Protection Against External Radiation*. ICRU Report 57. International Commission on Radiation Units and Measurements: Bethesda, MD, 1998.

108. Hughes, G.H. and Waters, L.S. *Many-Particle MCNP*. XTM:HGH-96-91. Los Alamos National Laboratory: Los Alamos, NM, 1996.

109. Hughes, G.H. and Waters, L.S. *Many-Particle MCNP Patch*. XTM:HGH-96-116. Los Alamos National Laboratory: Los Alamos, NM, 1996.

110. Hughes, G.H. and Waters, L.S. *Many-Particle MCNP Patch for 4XQ*. XTM:HGH-96-226. Los Alamos National Laboratory: Los Alamos, NM, 1996.

111. White, M.C. *ACE Tabular Angular Distributions*. XCI:MCW-99-81. Los Alamos National Laboratory: Los Alamos, NM, 1999.

Los Alamos
NATIONAL LABORATORY

Los Alamos, New Mexico 87545